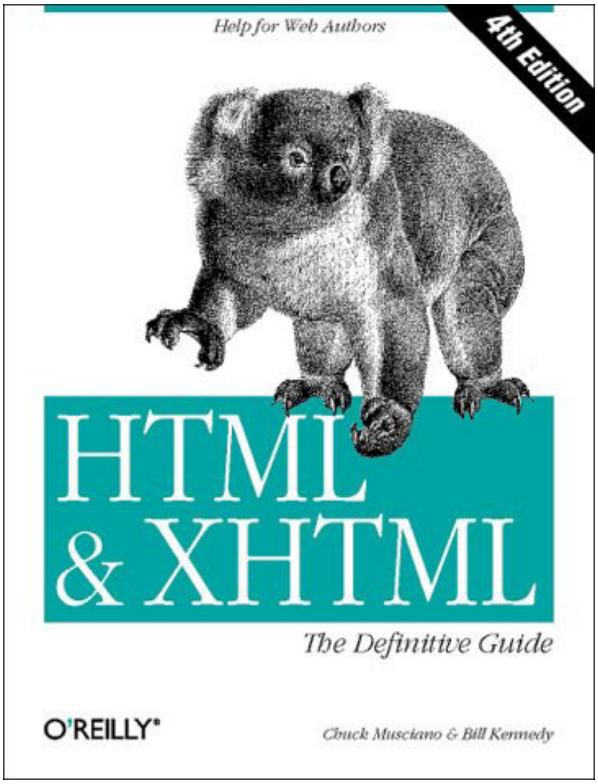

Help for Web Authors
4th Edition
HTML
& XHTML
The Definitive Guide
O'REILLY
Chuck Musciano & Bill Kennedy

HTML & XHTML: The Definitive Guide
4th edition

Chuck Musciano & Bill Kennedy

Fourth Edition August 2000

ISBN: 0-596-00026-X, 677 pages

This complete guide is full of examples, sample code, and practical hands-on advice for creating truly effective web pages and mastering advanced features.

Web authors learn how to insert images, create useful links and searchable documents, use Netscape extensions, design great forms, and much more.

The fourth edition covers XHTML 1.0, HTML 4.01, Netscape 6.0, and Internet Explorer 6.0, plus all the common extensions.

Table of Contents

Table of Contents (cont...)

Table of Contents (cont...)

Description

HTML is changing so fast it's almost impossible to keep up with developments. XHTML is HTML 4.0 rewritten in XML; it provides the precision of XML while retaining the flexibility of HTML. *HTML & XHTML: The Definitive Guide,* 4th Edition, brings it all together. It's the most comprehensive book available on HTML and XHTML today. It covers Netscape Navigator 6.0, Internet Explorer 5.0, HTML 4.01, XHTML 1.0, JavaScript, Style sheets, Layers, and all of the features supported by the popular web browsers.

Learning HTML and XHTML is like learning any new language, computer or human. Most students first immerse themselves in examples. Studying others is a natural way to learn, making learning easy and fun. Imitation can take learning only so far, though. It's as easy to learn bad habits through imitation as it is to acquire good ones. The better way to become HTML-fluent is through a comprehensive reference that covers the language syntax, semantics, and variations in detail and demonstrates the difference between good and bad usage.

HTML & XHTML: The Definitive Guide, 4th Edition, helps in both ways: the authors cover every element of HTML/XHTML in detail, explaining how each element works and how it interacts with other elements. Many hints about HTML/XHTML style smooth the way for writing documents that range from simple online documentation to complex presentations. With hundreds of examples, the book gives web authors models for writing their own effective web pages and for mastering advanced features, like style sheets and frames.

HTML & XHTML: The Definitive Guide, 4th Edition, shows how to:

- Implement the XHTML 1.0 standard and prepare web pages for the transition to XML browsers
- Use style sheets and layers to control a document's appearance
- Create tables, from simple to complex
- Use frames to coordinate sets of documents
- Design and build interactive forms and dynamic documents
- Insert images, sound files, video, Java applets, and JavaScript programs
- Create documents that look good on a variety of browsers
- Use new features to support multiple languages

Preface

Learning Hypertext Markup Language (HTML) and Extensible Hypertext Markup Language (XHTML) is like learning any new language, computer or human. Most students first immerse themselves in examples. Studying others is a natural way to learn, making learning easy and fun. Our advice to anyone wanting to learn HTML and XHTML is to get out there on the World Wide Web with a suitable browser and see for yourself what looks good, what's effective, what works for you. Examine others' documents and ponder the possibilities. Mimicry is how many of the current webmasters have learned the language.

Imitation can take you only so far, though. Examples can be both good and bad. Learning by example will help you talk the talk, but not walk the walk. To become truly conversant, you must learn how to use the language appropriately in many different situations. You could learn all that by example, if you live long enough.

Remember, too, that computer-based languages are more explicit than human languages. You've got to get the language syntax correct or it won't work. Then, too, there is the problem of "standards." Committees of academics and industry experts define the proper syntax and usage of a computer language like HTML. The problem is that browser manufacturers like Netscape Communications Corporation (now an America Online company) and Microsoft Corporation choose the parts of the standard they will use and which parts they will ignore. They even make up their own parts, which may eventually become standards.

Standards change, too. As we write this current edition, HTML is undergoing a conversion into XHTML, making it an application of the Extensible Markup Language (XML). HTML and XHTML are so similar that we often refer to them as a single language. But there are key differences; more about this later in the preface.

To be safe, the way to become fluent in HTML and XHTML is through a comprehensive, up-to-date language reference that covers the language syntax, semantics, and variations in detail to help you distinguish between good and bad usage.

There's one more step leading to fluency in a language. To become a true master of the language, you need to develop your own style. That means knowing not only what is appropriate, but what is effective. Layout matters. A lot. So does the order of presentation within a document, between documents, and between document collections.

Our goal in writing this book is to help you become fluent in HTML and XHTML, fully versed in their syntax, semantics, and elements of style. We take the natural learning approach, using examples: good ones, of course. We cover every element of the currently accepted versions (HTML 4.01 and XHTML 1.0) of the languages in detail, as well as all of the current extensions supported by the popular browsers, explaining how each element works and how it interacts with all the other elements.

And, with all due respect to Strunk and White, throughout the book we will give you suggestions for style and composition to help you decide how best to use HTML and XHTML to accomplish a variety of tasks, from simple online documentation to complex marketing and sales presentations. We'll show you what works and what doesn't, what makes sense to those who view your pages, and what might be confusing.

In short, this book is a complete guide to creating documents using HTML and XHTML, starting with basic syntax and semantics, and finishing with broad style guidelines to help you create beautiful, informative, accessible documents that you'll be proud to deliver to your browsers.

Our Audience

We wrote this book for anyone interested in learning and using the language of the Web, from the most casual user to the full-time design professional. We don't expect you to have any experience in HTML or XHTML before picking up this book. In fact, we don't even expect that you've ever browsed the World Wide Web, although we'd be very surprised if you haven't at least experimented with this technology by now. Being connected to the Internet is not necessary to use this book, but if you're not connected, this book becomes like a travel guide for the homebound.

The only things we ask you to have are a computer, a text editor that can create simple ASCII text files, and copies of the latest leading web browsers - preferably Netscape Navigator and Internet Explorer. Because HTML and XHTML documents are stored in a universally accepted format - ASCII text - and because the languages are completely independent of any specific computer, we won't even make an assumption about the kind of computer you're using. However, browsers do vary by platform and operating system, which means that your HTML or XHTML documents can look quite different depending on the computer and version of browser. We will explain how the various browsers use certain language features, paying particular attention to how they are different.

If you are new to HTML, the World Wide Web, or hypertext documentation in general, you should start by reading Chapter 1. In it, we describe how all the World Wide Web technologies come together to create webs of interrelated documents.

If you are already familiar with the Web, but not with HTML or XHTML specifically, or if you are interested in the new features in the latest standard version of HTML and XHTML, start by reading Chapter 2. This chapter is a brief overview of the most important features of the language and serves as a roadmap to how we approach the language in the remainder of the book.

Subsequent chapters deal with specific language features in a roughly top-down approach to HTML and XHTML. Read them in order for a complete tour through the language, or jump around to find the exact feature you're interested in.

Text Conventions

Throughout the book, we use a `constant-width` typeface to highlight any literal element of the HTML/XHTML standards, tags, and attributes. We always use lowercase letters for tags.[1] We use *italic* to indicate new concepts when they are defined and for those elements you need to supply when creating your own documents, such as tag attributes or user-defined strings.

> [1] HTML is case-insensitive with regard to tag and attribute names, but XHTML is case-sensitive. And some HTML items like source filenames, are case-sensitive, so be careful.

We discuss elements of the language throughout the book, but you'll find each one covered in depth (some might say in nauseating detail) in a shorthand, quick-reference definition box that looks like the following box. The first line of the box contains the element name, followed by a brief description of its function. Next, we list the various attributes, if any, of the element: those things that you may or must specify as part of the element.

<html>

Function:

Delimits a complete HTML document

Attributes:

DIR
VERSION
LANG

End tag:

</html>; may be omitted in HTML

Contains:

head_tag, *body_tag*, *frames*

We use the following symbols to identify tags and attributes that are not in the HTML 4.01 or XHTML 1.0 standards, but are additions to the languages:

Netscape Navigator extension to the standards

Internet Explorer extension to the standards

The description also includes the ending tag, if any, for the element, along with a general indication whether or not the end tag may be safely omitted in general use with HTML. With the few tags that do not have an end tag in HTML, but for which XHTML requires one, the language lets you indicate that ending with a forward slash (/) at the end of the tag, such as `<br />`. In these cases, the tag also may contain attributes, indicated with an intervening elipsis, such as `<br ... />`.

"Contains" names the rule in the HTML grammar that defines the elements to be placed within this tag. Similarly, "Used in" lists those rules that allow this tag as part of their content. These rules are defined in Appendix A.

Finally, HTML and XHTML are fairly intertwined languages. You will occasionally use elements in different ways depending on context, and many elements share identical attributes. Wherever possible, we place a cross-reference in the text that leads you to a related discussion elsewhere in the book. These cross-references, like the one at the end of this paragraph, serve as a crude paper model of hypertext documentation, one that would be replaced with a true hypertext link should this book be delivered in an electronic format. Section 3.3.1

We encourage you to follow these references whenever possible. Often, we'll only cover an attribute briefly and expect you to jump to the cross-reference for a more detailed discussion. In other cases, following the link will take you to alternative uses of the element under discussion or to style and usage suggestions that relate to the current element.

Versions and Semantics

The latest HTML standard is Version 4.01, but most updates and changes to the language standard were made in Version 4.0. Therefore, throughout the book, we generally refer to the HTML standard as HTML 4, encompassing all Versions 4.0 and later. We explicitly state the "dot" version number only when it is relevant.

The XHTML standard is currently in its first iteration, 1.0. For the most part, XHTML 1.0 is identical to HTML 4.01; we detail their differences in Chapter 16. Throughout the book, we specifically note cases where XHTML handles a feature or element differently than the original language, HTML.

The HTML and XHTML standards make very clear the distinction between "element types" of a document and the markup "tags" that delimit those elements. For example, the standard refers to the paragraph element type, which is not the same as the tag. The paragraph element consists of the accepted element-type name within the starting tag ( ), intervening content, and the ending paragraph ( ) tag. The tag is the starting tag for the paragraph element, and its contents, known as attributes, ultimately affect the paragraph element type's contents.

Although these are important distinctions, we're pragmatists. It is the markup tag that authors apply in their documents and that affects the intervening content, if any. Accordingly, throughout the book, we relax the distinction between element types and tags, most often talking about tags and all related contents, not necessarily using the term element-type when it would be technically appropriate to make the distinction. Forgive us the transgression, but we do so for the sake of clarity.

Is HTML Going Away?

Heavens, no. Why would we even think such a thing?

Well, actually, the language has reached middle age in standard Version 4.01 and is not expected to change again. Rather, HTML is being subsumed and modularized as part of Extensible Markup Language (XML). Its new name is XHTML, Extensible Hypertext Markup Language.

The emergence of XHTML is just another chapter in the often tumultuous history of HTML and the World Wide Web, where confusion for authors is the norm, not the exception. At the worst point, the elders of the World Wide Web Consortium (W3C) responsible for accepted and acceptable uses of the language - i.e., standards - lost control of the language in the browser "wars" between Netscape Communications and Microsoft. The abortive HTML+ standard never got off the ground, and HTML 3.0 became so bogged down in debate that the W3C simply shelved the entire draft standard. HTML 3.0 never happened, despite what some opportunistic marketers claimed in their literature. Instead, by late 1996, the browser manufacturers convinced the W3C to release HTML standard Version 3.2, which for all intents and purposes simply standardized most of the leading browser's (Netscape's) HTML extensions.

Fortunately for those of us who appreciate and strongly support standards, the W3C took back its primacy role with HTML 4.0, which stands today as HTML Version 4.01, released in December 1999. The standard is clearer and cleaner than any previous ones, establishes solid implementation models for consistency across browsers and platforms, provides strong supports and incentives for the companion Cascading Style Sheets (CSS) standard for HTML-based displays, and makes provisions for alternative (non-visual) user-agents, as well as for more universal language supports.

Cleaner and clearer aside, the W3C realized that HTML could never keep up with the demands of the web community for more ways to distribute, process, and display documents. HTML only offers a limited set of document creation primitives and is hopelessly incapable of handling non-traditional content like chemical formulae, musical notation, or mathematical expressions. Nor can it well support alternative display media, such as handheld computers or intelligent cellular phones, for instance.

To address these demands, the W3C developed the Extensible Markup Language (XML) standard. XML provides the way to create new, standards-based markup languages that don't take an act of the W3C to implement. XML-compliant languages deliver information that can be parsed, processed, displayed, sliced, and diced by the many different communication technologies that have emerged since the Web sparked the digital communication revolution a decade ago. XHTML is HTML reformulated to adhere to the XML standard. It is the foundation language for the future of the Web.

Why not just drop HTML for XHTML? For many reasons. First and foremost, don't expect everyone to just drop everything and start using XHTML standards (Version 1.0 just got recommended in January 2000). There's just too much current investment in HTML-based documentation and expertise for that to happen anytime soon. Besides, XHTML is HTML 4.01 reformulated as an application of XML. Know HTML 4 and you're all ready for the future.[2]

[2] We plumb the depths of XML and XHTML in Chapter 15 and Chapter 16.

The paradox in all this is that even the HTML 4.01 standard is not the definitive resource. There are many more features of HTML in popular use and supported by the popular browsers than are included in the latest language standard. And there are many parts of the standards that are ignored. We promise you, things can get downright confusing when you're trying to sort it all out.

We've managed to sort things out, so you don't have to sweat over what works with what browser and what doesn't work. This book, therefore, is the definitive guide to HTML and XHTML. We give details for all the elements of the HTML 4.01 and XHTML 1.0 standards, plus the variety of interesting and useful extensions to the language - some proposed standards - that the popular browser manufacturers have chosen to include in their products, such as:

- Cascading Style Sheets
- Java and JavaScript
- Layers
- Multiple columns

And while we tell you about each and every feature of the language, standard or not, we also tell you which browsers or different versions of the same browser implement a particular extension and which don't. That's critical knowledge when you want to create web pages that take advantage of the latest version of Netscape Navigator versus pages that are accessible to the larger number of people using Internet Explorer or even Lynx, a once-popular text-only browser for Unix systems.

In addition, there are a few things that are closely related but not directly part of HTML. For example, we touch, but do not handle, CGI and Java programming. CGI and Java programs work closely with HTML documents and run with or alongside browsers, but are not part of the language itself, so we don't delve into them. Besides, they are comprehensive topics that deserve their own books, such as *CGI Programming with Perl, by Scott Guelich, Shishir Gundavaram, and Gunther Birzneiks,* and *Java in a Nutshell*, by David Flanagan, both published by O'Reilly & Associates.

This is your definitive guide to HTML and XHTML as they are and should be used, including every extension we could find. Some extensions aren't documented anywhere, even in the plethora of online guides. But, if we've missed anything, certainly let us know and we'll put it in the next edition.

We'd Like to Hear from You

We have tested and verified all of the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
800-998-9938 (in the U.S. or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (FAX)

You can also send us messages electronically. To be put on the mailing list or request a catalog, send email to:

info@oreilly.com

To ask technical questions or comment on the book, send email to:

bookquestions@oreilly.com

Since the HTML and XHTML standards and browser additions to the languages are evolving so rapidly, some of the information in this book may be slightly out of date by the time you read it. We have a web site for the book, where we'll list errata and plans for future editions. Here you'll also find all the source code from the book available for download so you don't have to type it all in:

http://www.oreilly.com/catalog/html4/

For more information about this book and others, see the O'Reilly web site:

http://www.oreilly.com/

Acknowledgments

We did not compose, and certainly could not have composed, this book without generous contributions from many people. Our wives, Jeanne and Cindy, and our children, Eva, Ethan, Courtney, and Cole (they happened *before* we started writing), formed the front lines of support. And there are numerous neighbors, friends, and colleagues who helped by sharing ideas, testing browsers, and letting us use their equipment to explore HTML. You know who you are, and we thank you all.

In addition, we thank our technical reviewers, Robert Eckstein, Kane Scarlett, Eric Raymond, and Chris Tacy, for carefully scrutinizing our work. We took most of your keen suggestions. We especially thank Mike Loukides, our editor, who had to bring to bear his vast experience in book publishing to keep us two mavericks corralled. And special thanks to Deb Cameron for her perseverance and insight in bringing the Fourth Edition to fruition.

Chapter 1. HTML, XHTML, and the World Wide Web

Though it began as a military experiment and spent its adolescence as a sandbox for academics and eccentrics, recent events have transformed the worldwide network of computer networks - also known as the *Internet* - into a rapidly growing and wildly diversified community of computer users and information vendors. Today, you can bump into Internet users of nearly any and all nationalities, of any and all persuasions, from serious to frivolous individuals, from businesses to nonprofit organizations, and from born-again Christian evangelists to pornographers.

In many ways, the World Wide Web - the open community of hypertext-enabled document servers and readers on the Internet - is responsible for the meteoric rise in the network's popularity. You, too, can become a valued member by contributing: writing HTML and XHTML documents and then making them available to web surfers worldwide.

Let's climb up the Internet family tree to gain some deeper insight into its magnificence, not only as an exercise of curiosity, but to help us better understand just who and what it is we are dealing with when we go online.

1.1 The Internet, Intranets,and Extranets

Although popular media accounts are often confused and confusing, the concept of the Internet really is rather simple. It's a worldwide collection of computer networks - a network of networks - sharing digital information via a common set of networking and software protocols. Nearly anyone can connect a computer to the Internet and immediately communicate with other computers and users that are on the Net.

Networks are not new to computers. What makes the Internet global network unique is its worldwide collection of digital telecommunication links that share a common set of computer-network technologies, protocols, and applications. So whether you use a PC with Microsoft Windows 2000 or Linux or have an ancient Apple IIe, when connected to the Internet, the computers all speak the same networking language and use functionally identical programs so that you can exchange information - even multimedia pictures and sound - with someone next door or across the planet.

The common and now quite familiar programs people use to communicate and distribute their work over the Internet have also found their way into private and semi-private networks. These so-called *intranets* and *extranets* use the same software, applications, and networking protocols of the Internet. But unlike the Internet, intranets are private networks, usually unconnected to outside institutional boundaries and with restricted access to only members of the institution. Likewise, extranets restrict access, but use the Internet to provide services to members.

The Internet, on the other hand, seemingly has no restrictions. Anyone with a computer and the right networking software and connection can "get on the Net" and begin exchanging their words, sounds, and pictures with others around the world, day or night: no membership required. And that's precisely what is confusing about the Internet.

Like an oriental bazaar, the Internet is not well organized, there are few content guides, and it can take a lot of time and technical expertise to tap its full potential.That's because...

1.1.1 In the Beginning

The Internet began in the late 1960s as an experiment in the design of robust computer networks. The goal was to construct a network of computers that could withstand the loss of several machines without compromising the ability of the remaining ones to communicate. Funding came from the U.S. Department of Defense, which had a vested interest in building information networks that could withstand nuclear attack.

The resulting network was a marvelous technical success, but was limited in size and scope. For the most part, only defense contractors and academic institutions could gain access to what was then known as the ARPAnet (Advanced Research Projects Agency network of the Department of Defense).

With the advent of high-speed modems for digital communication over common phone lines, some individuals and organizations not directly tied to the main digital pipelines began connecting and taking advantage of the network's advanced and global communications. Nonetheless, it wasn't until these last few years (around 1993, actually) that the Internet really took off.

Several crucial events led to the meteoric rise in popularity of the Internet. First, in the early 1990s, businesses and individuals eager to take advantage of the ease and power of global digital communications finally pressured the largest computer networks on the mostly U.S. government-funded Internet to open their systems for nearly unrestricted traffic. (Remember, the network wasn't designed to route information based on content - meaning that commercial messages went through university computers that at the time forbade such activity.)

True to their academic traditions of free exchange and sharing, many of the original Internet members continued to make substantial portions of their electronic collections of documents and software available to the newcomers - free for the taking! Global communications, a wealth of free software and information: who could resist?

Well, frankly, the Internet was a tough row to hoe back then. Getting connected and using the various software tools, if they were even available for their computers, presented an insurmountable technology barrier for most people. And most available information was plain-vanilla ASCII about academic subjects, not the neatly packaged fare that attracts users to online services such as America Online, Prodigy, or CompuServe. The Internet was just too disorganized, and, outside of the government and academia, few people had the knowledge or interest to learn how to use the arcane software or the time to spend rummaging through documents looking for ones of interest.

1.1.2 HTML and the World Wide Web

It took another spark to light the Internet rocket. At about the same time the Internet opened up for business, some physicists at CERN, the European Particle Physics Laboratory, released an authoring language and distribution system they developed for creating and sharing multimedia-enabled, integrated electronic documents over the Internet. And so was born *Hypertext Markup Language* (HTML), browser software, and the World Wide Web. No longer did authors have to distribute their work as fragmented collections of pictures, sounds, and text. HTML unified those elements. Moreover, the World Wide Web's systems enabled *hypertext linking*, whereby documents automatically reference other documents, located anywhere around the world: less rummaging, more productive time online.

Lift-off happened when some bright students and faculty at the National Center for Supercomputing Applications (NCSA) at the University of Illinois, Urbana-Champaign wrote a web browser called Mosaic. Although designed primarily for viewing HTML documents, the software also had built-in tools to access the much more prolific resources on the Internet, such as FTP archives of software and Gopher-organized collections of documents.

With versions based on easy-to-use graphical-user interfaces familiar to most computer owners, Mosaic became an instant success. It, like most Internet software, was available on the Net for free. Millions of users snatched up a copy and began surfing the Internet for "cool web pages."

1.1.3 Golden Threads

There you have the history of the Internet and the World Wide Web in a nutshell: from rags to riches in just a few short years. The Internet has spawned an entirely new medium for worldwide information exchange and commerce, and its pioneers are profiting well. For instance, when the marketers caught on to the fact that they could cheaply produce and deliver eye-catching, wow-and-whizbang commercials and product catalogs to those millions of web surfers around the world, there was no stopping the stampede of blue suede shoes. Even the key developers of Mosaic and related web server technologies sensed potential riches. They left NCSA and formed Netscape Communications to produce commercial web browser and server software.

Business users and marketing opportunities have helped invigorate the Internet and fuel its phenomenal growth, particularly on the World Wide Web. But do not forget that the Internet is first and foremost a place for social interaction and information sharing, not a strip mall or direct advertising medium. Internet users, particularly the old-timers, adhere to commonly held, but not formally codified, rules of *netiquette* that prohibit such things as "spamming" special-interest newsgroups with messages unrelated to the topic at hand or sending unsolicited email. And there are millions of users ready to remind you of those rules should you inadvertently or intentionally ignore them.

Certainly, the power of HTML and network distribution of information go well beyond marketing and monetary rewards: serious informational pursuits also benefit. Publications, complete with images and other media like executable software, can get to their intended audience in a blink of an eye, instead of the months traditionally required for printing and mail delivery. Education takes a great leap forward when students gain access to the great libraries of the world. And at times of leisure, the interactive capabilities of HTML links can reinvigorate our otherwise television-numbed minds.

1.2 Talking the Internet Talk

Every computer connected to the Internet (even a beat-up old Apple II) has a unique address: a number whose format is defined by the *Internet Protocol* (IP), the standard that defines how messages are passed from one machine to another on the Net. An *IP address* is made up of four numbers, each less than 256, joined together by periods, such as 192.12.248.73 or 131.58.97.254.

While computers deal only with numbers, people prefer names. For this reason, each computer on the Internet also has a name bestowed upon it by its owner. There are several million machines on the Net, so it would be very difficult to come up with that many unique names, let alone keep track of them all. Recall, though, that the Internet is a network of networks. It is divided into groups known as *domains*, which are further divided into one or more *subdomains*.

So, while you might choose a very common name for your computer, it becomes unique when you append, like surnames, all of the machine's domain names as a period-separated suffix, creating a *fully qualified* domain name.

This naming stuff is easier than it sounds. For example, the fully qualified domain name *www.oreilly.com* translates to a machine named "www" that's part of the domain known as "oreilly," which, in turn, is part of the commercial (com) branch of the Internet. Other branches of the Internet include educational institutions (edu), nonprofit organizations (org), U.S. government (gov), and Internet service providers (net). Computers and networks outside the United States may have a two-letter abbreviation at the end of their names: for example, "ca" for Canada, "jp" for Japan, and "uk" for the United Kingdom.

Special computers, known as *name servers*, keep tables of machine names and their associated unique IP numerical addresses and translate one into the other for us and for our machines. Domain names must be registered and sometimes paid for through the nonprofit organization InterNIC. Once registered, the owner of the domain name broadcasts it and its address to other domain name servers around the world. Each domain and subdomain has an associated name server, so ultimately every machine is known uniquely by both a name and an IP address.

1.2.1 Clients, Servers, and Browsers

The Internet connects two kinds of computers: *servers*, which serve up documents, and *clients*, which retrieve and display documents for us humans. Things that happen on the server machine are said to be on the *server side*, while activities on the client machine occur on the *client side*.

To access and display HTML documents, we run programs called *browsers* on our client computers. These browser clients talk to special *web servers* over the Internet to access and retrieve electronic documents.

Several web browsers are available - most are free - each offering a different set of features. For example, browsers like Lynx run on character-based clients and display documents only as text. Then there are others that run on clients with graphical displays and render documents using proportional fonts and color graphics on a 1024 x 768, 24-bit-per-pixel display. Others still - Netscape Navigator, Microsoft's Internet Explorer, Opera, and Mozilla, to name a few - have special features that allow you to retrieve and display a variety of electronic documents over the Internet, including audio and video multimedia.

1.2.2 The Flow of Information

All web activity begins on the client side, when a user starts his or her browser. The browser begins by loading a *home page* document from either local storage or from a server over some network, such as the Internet, a corporate intranet, or a town extranet. In these latter cases, the client browser first consults a domain name system (DNS) server to translate the home page document server's name, such as *www.oreilly.com*, into an IP address, before sending a request to that server over the Internet. This request (and the server's reply) is formatted according to the dictates of the *Hypertext Transfer Protocol* (HTTP) standard.

A server spends most of its time listening to the network, waiting for document requests with the server's unique address stamped on it. Upon receipt, the server verifies that the requesting browser is allowed to retrieve documents from the server, and, if so, checks for the requested document. If found, the server sends (downloads) the document to the browser. The server usually logs the request, the client computer's name, document requested, and the time.

Back on the browser, the document arrives. If it's a plain-vanilla ASCII text file, most browsers display it in a common, plain-vanilla way. Document directories, too, are treated like plain documents, although most graphical browsers will display folder icons, which the user can select with the mouse to download the contents of subdirectories.

Browsers also retrieve binary files from a server. Unless assisted by a *helper* program or specially enabled by *plug-in* software or *applets*, which display an image or video file or play an audio file, the browser usually stores downloaded binary files directly on a local disk for later use.

For the most part, however, the browser retrieves a special document that appears to be a plain text file, but contains both text and special markup codes called *tags*. The browser processes these HTML or XHTML documents, formatting the text based upon the tags and downloading special accessory files, such as images.

The user reads the document, selects a hyperlink to another document, and the entire process starts over.

1.2.3 Beneath the World Wide Web

We should point out again that browsers and HTTP servers need not be part of the Internet's World Wide Web to function. In fact, you never need to be connected to the Internet, an intranet or extranet, or to any network, for that matter, to write documents and operate a browser. You can load up and display on your client browser locally stored documents and accessory files directly. This isolation is good: it gives you the opportunity to finish, in the editorial sense of the word, a document collection for later distribution. Diligent authors work locally to write and proof their documents before releasing them for general distribution, thereby sparing readers the agonies of broken image files and bogus hyperlinks.[1]

> [1] Vigorous testing of the HTML documents once they are made available on the Web is, of course, also highly recommended and necessary to rid them of various linking bugs.

Organizations, too, can be connected to the Internet and the World Wide Web, but also maintain private webs and document collections for distribution to clients on their local network, or intranet. In fact, private webs are fast becoming the technology of choice for the paperless offices we've heard so much about these last few years. With HTML, and especially with next-generation XHTML document collections, businesses and other enterprises can maintain personnel databases, complete with employee photographs and online handbooks, collections of blueprints, parts, and assembly manuals, and so on - all readily and easily accessed electronically by authorized users and displayed on a local computer.

1.2.4 Standards Organizations

Like many popular technologies, HTML started out as an informal specification used by only a few people. As more and more authors began to use the language, it became obvious that more formal means were needed to define and manage - to standardize - the language's features, making it easier for everyone to create and share documents.

1.2.4.1 The World Wide Web Consortium

The World Wide Web Consortium (W3C) was formed with the charter to define the standards for HTML. Members are responsible for drafting, circulating for review, and modifying the standard based on cross-Internet feedback to best meet the needs of the many.

Beyond HTML, the W3C has the broader responsibility of standardizing any technology related to the World Wide Web; they manage the HTTP, Cascading Style Sheet, and Extensible Markup Language (XML) standards, as well as related standards for document addressing on the Web. And they solicit draft standards for extensions to existing web technologies.

If you want to track HTML, XML, XHTML, CSS, and other exciting web development and related technologies, contact the W3C at http://www.w3.org .

Also, several Internet newsgroups are devoted to the Web, each a part of the *comp.infosystems.www* hierarchy. These include *comp.infosystems.www.authoring.html* and *comp.infosystems.www.authoring.images*.

1.2.4.2 The Internet Engineering Task Force

Even broader in reach than W3C, the Internet Engineering Task Force (IETF) is responsible for defining and managing every aspect of Internet technology. The World Wide Web is just one small part under the purview of the IETF.

The IETF defines all of the technology of the Internet via official documents known as Requests For Comment, or RFCs. Individually numbered for easy reference, each RFC addresses a specific Internet technology - everything from the syntax of domain names and the allocation of IP addresses to the format of electronic mail messages.

To learn more about the IETF and follow the progress of various RFCs as they are circulated for review and revision, visit the IETF home page, http://www.ietf.org.

1.3 HTML: What It Is

HTML is a document-layout and hyperlink-specification language. It defines the syntax and placement of special, embedded directions that aren't displayed by the browser, but tell it how to display the contents of the document, including text, images, and other support media. The language also tells you how to make a document interactive through special hypertext links, which connect your document with other documents - on either your computer or someone else's, as well as with other Internet resources, like FTP.

1.3.1 HTML Standards and Extensions

The basic syntax and semantics of HTML are defined in the HTML standard, currently Version 4.01. HTML has matured in barely eight years, having gone through at least four iterations in as many years. At one time, a new version would appear before you had a chance to finish reading this book. Today, the pace of change has slowed. Now the wait is for browser manufacturers to implement the standards.

Browser developers rely upon the HTML standard to program the software that formats and displays common HTML documents. Authors use the standard to make sure they are writing effective, correct HTML documents.

However, the standard is not always explicit; manufacturers have some leeway in how their browser might display an element. And to complicate matters, commercial forces have pushed developers to add into their browsers nonstandard extensions meant to improve the language.

In this book, we explore in detail the syntax, semantics, and idioms of HTML Version 4.01, along with the many important extensions that are supported in the latest versions of the most popular browsers, so that any aspiring HTML author can create fabulous documents with a minimum of effort.

1.4 XHTML: What It Is

You've certainly heard of HTML, but did you know that it is one of many other markup languages? Indeed, HTML is the black sheep in the family of document markup languages. HTML is based on SGML, the Standard Generalized Markup Language. The powers-that-be created SGML with the intent that it be the one and only markup metalanguage from which all other document markup elements would be created. Everything from hieroglyphics to HTML can be defined using SGML, negating any need for any other markup language.

The problem with SGML is that it is so broad and all-encompassing that mere mortals cannot use it. Using SGML effectively requires very expensive and complex tools that are completely beyond the scope of regular people who just want to bang out an HTML document in their spare time. As a result, HTML and other language standards adhere to some, but not all SGML standards,[2] eliminating many of the more esoteric features so that HTML is readily useable and used.

[2] The HTML DTD in Appendix D uses a subset of SGML to define the HTML 4.01 standard.

Recognizing that SGML is unwieldy and not well-suited to describing the very popular HTML in a useful way, and that there was a growing need to define other HTML-like markup languages to handle different network documents, the W3C defined the Extensible Markup Language (XML). Like SGML, XML is a separate formal markup metalanguage that uses select features of SGML to define markup languages. It eliminates many features of SGML that aren't applicable to languages like HTML and simplifies other SGML elements in order to make them easier to use and understand.

HTML Version 4.01 is not XML-compliant. Hence, the W3C offers XHTML, a reformulation of HTML to be compliant under XML. XHTML attempts to support every last nit and feature of HTML 4.01 using the more rigid rules of XML. It generally succeeds but has enough differences to make life difficult for the standards-conscious HTML author.

Confused? Don't be. Learning HTML is still the way to go for most authors and Web developers. The native language endures. Besides, by learning HTML, you learn the working bits of XHTML, effectively the same things. There are some differences, which we explore in Chapter 16, XHTML. But the differences should not affect your work in the foreseeable future.

1.5 HTML and XHTML: What They Aren't

With all their multimedia-enabling, new page layout features, and the hot technologies that give life to HTML/XHTML documents over the Internet, it is also important to understand the languages' limitations. They are not word-processing tools, desktop publishing solutions, or even programming languages. That's because their fundamental purpose is to define the structure and appearance of documents and document families so that they may be delivered quickly and easily to a user over a network for rendering on a variety of display devices. Jack of all trades, but master of none, so to speak.

1.5.1 Content Versus Appearance

Before you can fully appreciate the power of the language and begin creating effective documents, you must yield to one fundamental rule. These markup languages are designed to structure documents and make their content more accessible, not to format documents for display purposes.

HTML and its progeny XHTML do provide many different ways to let you define the appearance of your documents: font specifications, line breaks, and multicolumn text are all features of the language. And, of course, appearance is important, since it can have either detrimental or beneficial effects on how users access and use the information in your documents.

But with HTML and XHTML, content is paramount; appearance is secondary, particularly since it is less predictable, given the variety of browser graphics and text-formatting capabilities. Besides, these markup languages contain many more ways for structuring your document content without regard to the final appearance: section headers, structured lists, paragraphs, rules, titles, and embedded images are all defined by the standard languages without regard for how these elements might be rendered by a browser. Consider, for example, a browser for the blind, wherein graphics on the page come with audio descriptions and alternative rules for navigation. The HTML 4 standard defines such a thing: content over visual presentation.

If you treat HTML or XHTML as a document-generation tool, you will be sorely disappointed in your ability to format your document in a specific way. There is simply not enough capability built into the languages to allow you to create the kind of documents you might whip up with tools like FrameMaker or Microsoft Word. Attempts to subvert the supplied structuring elements to achieve specific formatting tricks seldom work across all browsers. In short, don't waste your time trying to force HTML and XHTML to do things they were never designed to do.

Instead, use HTML and XHTML in the manner for which they were designed: indicating the structure of a document so that the browser can then render its content appropriately. HTML and XHTML are rife with tags that let you indicate the semantics of your document content, something that is missing from tools like Frame or Word. Create your documents using these tags and you'll be happier, your documents will look better, and your readers will benefit immensely.

1.6 Nonstandard Extensions

It doesn't take an advanced degree in the obvious to know that many people vie for distinction to draw the attentions of others. So, too, with browsers. Extra whizbang features can give the edge in the otherwise standardized market. That can be a nightmare for authors. A lot of people want you to use the latest and greatest gimmick or even useful HTML extension. But it's not part of the standard, and not all browsers support it. In fact, on occasion, the popular browsers support different ways of doing the same thing.

1.6.1 Extensions: Pro and Con

Every software vendor adheres to the technological standards; it's embarrassing to be incompatible and your competitors will take every opportunity to remind buyers of your product's failure to comply, no matter how arcane or useless that standard might be. At the same time, vendors seek to make their products different and better than the competition's offerings. Netscape's and Internet Explorer's extensions to standard HTML are perfect examples of these market pressures.

Many document authors feel safe using these extended browsers' nonstandard extensions because of their combined and commanding share of users. For better or worse, extensions to HTML made by the folks at Netscape or Microsoft instantly become part of the street version of the language, much like English slang creeping into the vocabulary of most Frenchmen, despite all the best efforts of the Académie Française.

Fortunately, with HTML Version 4.0, the W3C standards caught up with the browser manufacturers. In fact, the tables turned somewhat. The many extensions to HTML that originally appeared as extensions in Netscape Navigator and Internet Explorer are now part of the HTML 4 and XHTML 1.0 standards, and there are other parts of the new standard that are not yet features of the popular browsers.

1.6.2 Avoiding Extensions

In general, we urge you to resist using an extension unless you have a compelling and overriding reason to do so. By using them, particularly in key portions of your documents, you run the risk of losing a substantial portion of your potential readership. Sure, the Internet Explorer community is large enough to make this point moot now, but even so, you are excluding several million people who use Netscape from your pages.

Of course, there are varying degrees of dependency on extensions. If you use some of the horizontal rule extensions, for example, most other browsers will ignore the extended attributes and render a conventional horizontal rule. On the other hand, reliance upon a number of font size changes and text alignment extensions to control your document appearance will make your document look terrible on many alternative browsers. It might not even display at all on browsers that don't support the extensions.

We admit that it is disingenuous of us to decry the use of extensions while presenting complete descriptions of their use. In keeping with the general philosophy of the Internet, we'll err on the side of handing out rope and guns to all interested parties while hoping you have enough smarts to keep from hanging yourself or shooting yourself in the foot.

Our advice still holds, though: only use an extension where it is necessary or very advantageous, and do so with the understanding that you are disenfranchising a portion of your audience. To that end, you might even consider providing separate, standards-based versions of your documents to accommodate users of other browsers.

1.6.3 Beyond Extensions: Exploiting Bugs

It is one thing to take advantage of an extension, and it is quite another to exploit known bugs in a particular version of a browser in order to achieve some unusual document effect.

A good example is the multiple-body bug in Version 1.1 of Netscape Navigator. The HTML standard insists that a compliant document have exactly one `<body>` tag, containing the body of the document. The now-obsolete browser allowed any number of `<body>` tags, processing and rendering each `<body>` in turn. By placing several `<body>` tags in an HTML document, an author could achieve crude animation effects when the document was first loaded into the browser. The most popular trick used several `<body>` tags, each with a slightly different background color. This trick results in a document fade-in effect.

The party ended when Version 1.2 of Netscape fixed the bug. Suddenly, thousands of documents lost their fancy fade-in effect. Although faced with some rather fierce complaints, to their credit, the people at Netscape stood by their decision to adhere to the standard, placing compliance higher on their list of priorities than nifty rendering hacks.

In that light, we can unequivocally offer this advice: *never* exploit a bug in a browser to achieve a particular effect in your documents.

1.7 Tools for the Web Designer

While you can use the barest of barebones text editors to create HTML and XHTML documents, most authors have a bit more elaborate toolbox of software utilities than a simple word processor. You also need a browser, so you can test and refine your work. Beyond the essentials are some specialized software tools for HTML document preparation and editing, and others for developing and preparing accessory multimedia files.

1.7.1 Essentials

At the very least, you'll need an editor, a browser to check your work, and ideally, a connection to the Internet.

1.7.1.1 Word processor or WYSIWYG editor?

Some authors use the word-processing capabilities of their specialized HTML/XHTML editing software. Others use the WYSIWYG (what-you-see-is-what-you-get) composition tools that come with their browser or the latest versions of the popular word processors. Others, such as ourselves, prefer to compose their work on a general word processor and later insert the markup tags and their attributes. Still others include markup as they compose.

We think the stepwise approach - compose, then mark up - is the better way. We find that once we've defined and written the document's content, it's much easier to make a second pass to judiciously and effectively add the HTML/XHTML tags to format the text. Otherwise, the markup can obscure the content. Note, too, that unless specially trained (if they can be), spellcheckers and thesauruses typically choke on markup tags and their various parameters. You can spend what seems to be a lifetime clicking the Ignore button on all those otherwise valid markup tags when syntax- or spell-checking a document.

When and how you embed markup tags into your document dictates the tools you need. We recommend that you use a good word processor, such as WordPerfect or Word, which comes with more and better writing tools than simple text editors or the browser-based markup-language editors. You'll find, for instance, that an outliner, spellchecker, and thesaurus will best help you craft the document's flow and content well, disregarding for the moment its look. The latest word processors encode your documents with HTML, too, but don't expect miracles. Except for boilerplate documents, you will probably need to nurse those automated HTML documents to full health. And it'll be a while before you'll see XHTML-specific markup tools in the popular word processors.

Another word of caution about automated composition tools: they typically change or insert content, such as replacing relative hyperlinks with full ones, and arrange your document in ways that will annoy you. Annoying, in particular, since they rarely give you the opportunity to do things your own way.

So become fluent in native HTML/XHTML. Be prepared to reverse some of the things a composition tool will do to your documents. And make sure you can wrest your document away from the tool so you can make it do your bidding.

1.7.1.2 Browser software

Obviously, you should view your newly composed documents and test their functionality before you release them for use by others. For serious authors, particularly those looking to push their documents beyond the HTML/XHTML standards, we recommend that you have several browser products, perhaps with versions running on different computers, just to be sure one's delightful display isn't another's nightmare.

The currently popular - and therefore most important - browsers are Netscape Navigator (the browser portion of Netscape Communicator) and Microsoft's Internet Explorer. Download the latest versions from their web sites.

1.7.1.3 Internet connection

We think you should have bona fide access to the Internet if you are really serious about learning and honing your document markup skills. Okay, it's not absolutely essential, since you can compose and view documents locally. And for some, a connection is perhaps not even possible or practical, but make the effort: sometimes there's no better way to learn than by example. Examples both good and bad abound on the Internet, and there are literally millions of Web pages whose source HTML you can download and examine, albeit fewer XHTML ones.

Moreover, an Internet connection *is* essential for development and testing if you include hypertext links to Internet services in your web documents. Most of all, an Internet connection gives you access to a wealth of tips and ongoing updates to the language through special-interest newsgroups, as well as much of the essential and accessory software you can use to prepare document collections.

1.7.2 An Extended Toolkit

If you're serious about creating documents, you'll soon find there are all sorts of nifty tools that make life easier. The list of freeware, shareware, and commercial products grows daily, so it's not very useful to provide a list here. This is, in fact, another good reason why you should get an Internet connection; various groups keep updated lists of HTML and XHTML resources on the Web. If you are really dedicated to writing in HTML and XHTML, you will visit those sites, and you will visit them regularly to keep abreast of the language, tools, and trends.

We think the following four web sites are the most useful for authors. Each contains dozens, sometimes hundreds, of hyperlinks to detailed descriptions of products and other important information. Go at it:

http://www.stars.com
http://msdn.microsoft.com
http://search.netscape.com
http://www.w3.org/MarkUp

Chapter 2. Quick Start

We didn't spend hours studiously poring over some reference book before we wrote our first HTML document. You probably shouldn't, either. HTML is simple to read and understand, and it's simple to write, too. And once you've written an HTML document, you've nearly completed your first XHTML one, too. So let's get started without first learning a lot of arcane rules.

To help you get that quick, satisfying start, we've included this chapter as a brief summary of the many elements of HTML and its progeny, XHTML. Of course, we've left out a lot of details and some tricks that you should know. Read the upcoming chapters to get the essentials for becoming fluent in HTML and XHTML.

Even if you are familiar with the languages, we recommend you work your way through this chapter before tackling the rest of the book. It not only gives you a working grasp of basic HTML and its jargon, but you'll also be more productive later, flush with the confidence that comes from creating attractive documents in such a short time.

2.1 Writing Tools

Use any text editor to create an HTML or XHTML document, as long as it can save your work on disk in ASCII text file format. That's because even though documents include elaborate text layout and pictures, they're all just plain old ASCII documents themselves. A fancier WYSIWYG editor or a translator for your favorite word processor are fine, too - although they may not support the many nonstandard features we discuss later in this book. You'll probably end up touching up the source text they produce, as well.

While not needed to compose documents, you should have at least one version of a popular browser installed on your computer to view your work, preferably Netscape Navigator or Microsoft's Internet Explorer. That's because the source document you compose on your text editor doesn't look anything like what gets displayed by a browser, even though it's the same document. Make sure what your readers actually see is what you intended by viewing the document yourself with a browser. Besides, the popular ones are free over the Internet.

Also note that you don't need a connection to the Internet or the World Wide Web to write and view your HTML or XHTML documents. You may compose and view your documents stored on a hard drive or floppy disk that's attached to your computer. You can even navigate among your local documents with the languages' hyperlinking capabilities without ever being connected to the Internet, or any other network, for that matter. In fact, we recommend that you work locally to develop and thoroughly test your documents before you share them with others.

We strongly recommend, however, that you *do* get a connection to the Internet if you are serious about composing your own documents. You may download and view others' interesting web pages and see how they accomplished some interesting feature - good or bad. Learning by example is fun, too. (Reusing others' work, on the other hand, is often questionable, if not downright illegal.) An Internet connection is essential if you include in your work hyperlinks to other documents on the Internet.

2.2 A First HTML Document

It seems every programming language book ever written starts off with a simple example on how to display the message, "Hello, World!" Well, you won't see a "Hello, World!" example in this book. After all, this is a style guide for the new millennium. Instead, ours sends greetings to the World Wide Web:

```
<html>
<head>
<title>My first HTML document</title>
</head>
<body>
<h2>My first HTML document</h2>
Hello, <i>World Wide Web!</i>
 <!-- No "Hello, World" for us -->
<p>
               Greetings from<br>
<a href="http://www.ora.com">O'Reilly & Associates</a>
<p>
Composed with care by:
<cite>(insert your name here)</cite>
<br>&copy;2000 and beyond
</body>
</html>
```

Go ahead: type in the example HTML source on a fresh word-processing page and save it on your local disk as *myfirst.html*. Make sure you select to save it in ASCII format; word processor-specific file formats like Microsoft Word's *.doc* files save hidden characters that can confuse the browser software and disrupt your HTML document's display.

After saving *myfirst.html* (or *myfirst.htm* if you are using archaic DOS- or Windows 3.11-based filenaming conventions) onto disk, start up your browser, locate, and then open the document from the program's File menu. Your screen should look like Figure 2-1.

Figure 2-1. A very simple HTML document

2.3 Embedded Tags

You have probably noticed right away, perhaps in surprise, that the browser displays less than half of the example source text. Closer inspection of the source reveals that what's missing is everything that's bracketed inside a pair of less-than (<) and greater-than (>) characters. Section 3.3.1

HTML and XHTML are embedded languages: you insert their directions or *tags* into the same document that you and your readers load into a browser to view. The browser uses the information inside those tags to decide how to display or otherwise treat the subsequent contents of your document.

For instance, the tag that follows the word "Hello" in the simple example tells the browser to display the following text in italics.[1] - Section 4.5

> [1] Italicized text is a very simple example and one that most browsers, except the text-only variety like Lynx, can handle. In general, the browser tries to do as it is told, but as we demonstrate in upcoming chapters, browsers vary from computer to computer and from user to user, as do the fonts that are available and selected by the user for viewing HTML documents. Assume that not all are capable or willing to display your HTML document exactly as it appears on your screen.

The first word in a tag is its formal name, which usually is fairly descriptive of its function, too. Any additional words in a tag are special *attributes*, sometimes with an associated value after an equal sign (=), which further define or modify the tag's actions.

2.3.1 Start and End Tags

Most tags define and affect a discrete region of your document. The region begins where the tag and its attributes first appear in the source document (a.k.a. the *start tag*) and continues until a corresponding *end tag*. An end tag is the tag's name preceded by a forward slash (/). For example, the end tag that matches the "start italicizing" tag is .

End tags never include attributes. In HTML, most tags, but not all, have an end tag. And, to make life a bit easier for HTML authors, the browser software often infers an end tag from surrounding and obvious context, so you needn't explicitly include some end tags in your source HTML document. (We tell you which are optional and which are never omitted when we describe each tag in later chapters.) Our simple example is missing an end tag that is so commonly inferred and hence not included in the source that some veteran HTML authors don't even know that it exists. Which one?

The XHTML standard is much more rigid, insisting that all tags have a corresponding end tag. Section 16.3.2 / Section 16.3.3

2.4 HTML Skeleton

Notice, too, in our simple example source that precedes Figure 2-1, the HTML document starts and ends with `<html>` and `</html>` tags. Of course, these tags tell the browser that the entire document is composed in HTML.[2] The HTML and XHTML standards require an `<html>` tag for compliant documents, but most browsers can detect and properly display HTML encoding in a text document that's missing this outermost structural tag. Section 3.6.1

[2] XHTML documents also begin with the <html> tag, but with additional information to differentiate them from common HTML documents. See Chapter 16 for details.

Like our example, all HTML and XHTML documents have two main structures: a *head* and a *body,* each bounded in the source by respectively named start and end tags. You put information about the document in the head and the contents you want displayed in the browser's window inside the body. Except in rare cases, you'll spend most of your time working on your document's body content. Section 3.7.1 / Section 3.8.1

There are several different document header tags you may use to define how a particular document fits into a document collection and into the larger scheme of the Web. Some nonstandard header tags even animate your document.

For most documents, however, the important header element is the title. Standards require that every HTML and XHTML document have a title, even though the currently popular browsers don't enforce that rule. Choose a meaningful title, one that instantly tells the reader what the document is about. Enclose yours, as we do for the title of our example, between the `<title>` and `</title>` tags in your document's header. The popular browsers typically display the title at the top of the document's window onscreen. Section 3.7.2

2.5 The Flesh on an HTML or XHTML Document

Except for the `<html>`, `<head>`, `<body>`, and `<title>` tags, the HTML and XHTML standards have few other required structural elements. You're free to include pretty much anything else in the contents of your document. (The web surfers among you know that authors have taken full advantage of that freedom, too.) Perhaps surprisingly, though, there are only three main types of HTML/XHTML content: tags (which we described previously), comments, and text.

2.5.1 Comments

A raw document with all its embedded tags can quickly become nearly unreadable, like computer-programming source code. We strongly recommend that you use comments to guide your composing eye.

Although it's part of your document, nothing in a comment, including the body of your comment that goes between the special starting tag `<!--` and ending tag delimiters `-->` gets included in the browser display of your document. Now you see a comment in the source, like in our simple HTML example, and now you don't on the display, as evidenced by our comment's absence in Figure 2-1. Anyone can download the source text of your documents and read the comments, though, so be careful what you write. Section 3.5.3

2.5.2 Text

If it isn't a tag or a comment, it's text. The bulk of content in most of your HTML/XHTML documents - the part readers see on their browser displays - is text. Special tags give the text structure, such as headings, lists, and tables. Others advise the browser how the content should be formatted and displayed.

2.5.3 Multimedia

What about images and other multimedia elements we see and hear as part of our web browser displays? Aren't they part of the HTML document? No. The data that comprise digital images, movies, sounds, and other multimedia elements that may be included in the browser display are in documents separate from the document. You include references to those multimedia elements via special tags. The browser uses the references to load and integrate other types of documents with your text.

We didn't include any special multimedia references in the previous example simply because they are separate, nontext documents you can't just type into a text processor. We do, however, talk about and give examples of how to integrate images and other multimedia in your documents later in this chapter, as well as in extensive detail in subsequent chapters.

2.6 Text

Text-related HTML/XHTML markup tags comprise the richest set of all in the standard languages. That's because the original language - HTML - emerged as a way to enrich the structure and organization of text.

HTML came out of academia. What was and still is important to those early developers was the ability of their mostly academic, text-oriented documents to be scanned and read without sacrificing their ability to distribute documents over the Internet to a wide diversity of computer display platforms. (ASCII text is the only universal format on the global Internet.) Multimedia integration is something of an appendage to HTML and XHTML, albeit an important one.

And page layout is secondary to structure. We humans visually scan and decide textual relationships and structure based on how it looks; machines can only read encoded markings. Because documents have encoded tags that relate meaning, they lend themselves very well to computer-automated searches and also to the recompilation of content - features very important to researchers. It's not so much *how* something is said as *what* is being said.

Accordingly, neither HTML nor XHTML are page-layout languages. In fact, given the diversity of user-customizable browsers as well as the diversity of computer platforms for retrieval and display of electronic documents, all these markup languages strive to accomplish is to *advise,* not dictate, how the document might look when rendered by the browser. You cannot force the browser to display your document in any certain way. You'll hurt your brain if you insist otherwise.

2.6.1 Appearance of Text

For instance, you cannot predict what font and what absolute size - 8- or 40-point Helvetica, Geneva, Subway, or whatever - will be used for a particular user's text display. Okay, so the latest browsers now support standard Cascading Style Sheets and other desktop publishing-like features that let you control the layout and appearance of your documents. But users may change their browser's display characteristics and override your carefully laid plans at will; quite a few of the older browsers out there don't support these new layout features; and some browsers are text-only with no nice fonts at all. What to do? Concentrate on content. Cool pages are a flash in the pan. Deep content will bring people back for more and more.

Nonetheless, style does matter for readability, and it is good to include it where you can, as long as it doesn't interfere with content presentation. You can attach common style attributes to your text with *physical style* tags like the italic `<i>` tag in the simple example. More importantly and truer to the language's original purpose, HTML and XHTML have *content-based* style tags that attach *meaning* to various text passages. And you can alter text display characteristics, such as font style and size, color, and so on, with Cascading Style Sheets.

Today's graphical browsers recognize the physical and content-related text style tags and change the appearance of their related text passage to visually convey meaning or structure. You can't predict exactly what that change will look like.

The HTML 4 standard, and particularly the XHTML 1.0 standard, stress that future browsers will not be so visually bound. Text contents may be heard or even felt, for example, not read by viewers. Context clues surely are better in those cases than physical styles.

2.6.1.1 Content-based text styles

Content-based style tags indicate to the browser that a portion of your HTML/XHTML text has a specific usage or meaning. The `<cite>` tag in our simple example, for instance, means the enclosed text is some sort of citation - the document's author, in this case. Browsers commonly, although not universally, display the citation text in italic, not as regular text. Section 4.4

While it may or may not be obvious to the current reader that the text is a citation, someday, someone might create a computer program that searches a vast collection of documents for embedded `<cite>` tags and compiles a special list of citations from the enclosed text. Similar software agents already scour the Internet for embedded information to compile listings, such as the infamous Webcrawler and the AltaVista database of web sites.

The most common content-based style used today is that of emphasis, indicated with the `<em>` tag. And if you're feeling really emphatic, you might use the `<strong>` content style. Other content-based styles include `<code>` , for snippets of programming code; `<kbd>`, to denote text entered by the user via a keyboard; `<samp>`, to mark sample text; `<dfn>`, for definitions; and `<var>`, to delimit variable names within programming code samples. All of these tags have corresponding end tags.

2.6.1.2 Physical styles

Even the barest of barebones text processors conform to a few traditional text styles, such as italic and bold characters. While not word-processing tools in the traditional sense, HTML and XHTML do provide tags that tell the browser explicitly to display (if it can) a character, word, or phrase in a particular physical style.

Although you should use related content-based tags for the reasons we argue earlier, sometimes form is more important than function. So use the `<i>` tag to italicize text, without imposing any specific meaning; the `<b>` tag to display text in boldface; or the `<tt>` tag so that the browser, if it can, displays the text in a teletype-style monospaced typeface. Section 4.5

It's easy to fall into the trap of using physical styles when you should really be using a content-based style instead. Discipline yourself now to use the content-based styles, because, as we argue earlier, they convey meaning as well as style, thereby making your documents easier to automate and manage.

2.6.1.3 Special text characters

Not all text characters available to you for display by a browser can be typed from the keyboard. And some characters have special meanings, such as the brackets around tags, which if not somehow differentiated when used for plain text - the less-than sign (<) in a math equation, for example - will confuse the browser and trash your document. HTML and XHTML give you a way to include any of the many different characters that comprise the ASCII character set anywhere in your text through a special encoding of its *character entity*.

Like the copyright symbol in our simple example, a character entity starts with an ampersand followed by its name, and terminated with a semicolon. Alternatively, you may also use the character's position number in the ASCII table of characters preceded by the pound or sharp sign (`#`) in lieu of its name in the character entity sequence. When rendering the document, the browser displays the proper character, if it exists in the user's font. Section 3.5.2

For obvious reasons, the most commonly used character entities are the greater-than (`>`), less-than (`<`), and ampersand (`&`) characters. Check Appendix F to find what symbol the character entity `¦` represents.

2.6.2 Text Structures

It's not obvious in our simple example, but the common carriage returns we use to separate paragraphs in our source document have no meaning in HTML or XHTML, except in special circumstances. You could have typed the document onto a single line in your text editor and it would still appear the same in Figure 2-1.[3]

> [3] We use a computer programming-like style of indentation so that our source HTML/XHTML documents are more readable. It's not obligatory, nor are there any formal style guidelines for source HTML/XHTML document text formats. We do, however, highly recommend that you adopt a consistent style, so that you and others can easily follow your source documents.

You'd soon discover, too, if you hadn't read it here first, that except in special cases, browsers typically ignore leading and trailing spaces, and sometimes more than a few in between. (If you look closely at the source example, the line "Greetings from" looks like it should be indented by leading spaces, but it isn't in Figure 2-1.)

2.6.2.1 Divisions, paragraphs, and line breaks

A browser takes the text in the body of your document and "flows" it onto the computer screen, disregarding any common carriage-return or line-feed characters in the source. The browser fills as much of each line of the display window as possible, beginning flush against the left margin, before stopping after the rightmost word and moving on to the next line. Resize the browser window, and the text reflows to fill the new space, indicating HTML's inherent flexibility.

Of course, readers would rebel if your text just ran on and on, so HTML and XHTML provide both explicit and implicit ways to control the basic structure of your document. The most rudimentary and common ways are with the division (`<div>`), paragraph (`<p>`), and line-break (`<br>`) tags. All break the text flow, which consequently restarts on a new line. The differences are that the `<div>` and `<p>` tags define an elemental region of the document and text, respectively, the contents of which you may specially align within the browser window, apply text styles to, and alter with other block-related features.

Without special alignment attributes, the `<div>` and `<br>` tags simply break a line of text and place subsequent characters on the next line. The paragraph tag adds more vertical space after the line break than either the `<div>` or `<br>` tags. Section 4.1.1 / Section 4.1.2 / Section 4.7.1

By the way, the HTML standard includes end tags for the paragraph and division tags, but not for the line-break tag.[4] Few authors ever include the paragraph end tag in their documents; the browser usually can figure out where one paragraph ends and another begins.[5] Give yourself a star if you knew that `</p>` even exists.

[4] With XHTML,
's start and end are between the same brackets:
. Browsers tend to be very forgiving and often ignore extraneous things, such as the forward slash in this case, so it's perfectly okay to get into the habit of adding that end-mark.

[5] The paragraph end tag is being used more commonly now that the popular browsers support the paragraph-alignment attribute.

2.6.2.2 Headings

Besides breaking your text into divisions and paragraphs, you also can organize your documents into sections with headings. Just as they do on this and other pages in this printed book, headings not only divide and title discrete passages of text: they also convey meaning visually. And headings also readily lend themselves to machine-automated processing of your documents.

There are six heading tags, `<h1>` through `<h6>`, with corresponding end tags. Typically, the browser displays their contents in, respectively, very large to very small font sizes, and sometimes in boldface. The text inside the `<h4>` tag is usually the same size as the regular text. Section 4.2.1

The heading tags also typically break the current text flow, standing alone on lines and separated from surrounding text, even though there aren't any explicit paragraph or line-break tags before or after a heading.

2.6.2.3 Horizontal rules

Besides headings, HTML and XHTML provide horizontal rule lines that help delineate and separate the sections of your document.

When the browser encounters an `<hr>` tag in your document, it breaks the flow of text and draws a line completely across the display window on a new line. The flow of text resumes immediately below the rule.[6] Section 5.1.1

[6] Similar to
, with XHTML the formal horizontal rule tag is <hr/>.

2.6.2.4 Preformatted text

Occasionally, you'll want the browser to display a block of text as-is: for example, with indented lines and vertically aligned letters or numbers that don't change even though the browser window might get resized. The `<pre>` tag rises to those occasions. All text up to the closing `</pre>` end tag appears in the browser window exactly as you type it, including carriage returns, line feeds, and leading, trailing, and intervening spaces. Although very useful for tables and forms, `<pre>` text turns out pretty dull; the popular browsers render the block in a monospace typeface. Section 4.7.5

2.7 Hyperlinks

While text may be the meat and bones of an HTML or XHTML document, the heart is hypertext. Hypertext gives users the ability to retrieve and display a different document in their own or someone else's collection simply by a click of the keyboard or mouse on an associated word or phrase (*hyperlink*) in the document. Use these interactive hyperlinks to help readers easily navigate and find information in your own or others' collections of otherwise separate documents in a variety of formats, including multimedia, HTML, XHTML, other XML, and plain ASCII text. Hyperlinks literally bring the wealth of knowledge on the whole Internet to the tip of the mouse pointer.

To include a hyperlink to some other document in your own collection or on a server in Timbuktu, all you need to know is the document's unique address and how to drop an *anchor* into your document.

2.7.1 URLs

While it is hard to believe, given the millions, perhaps billions, of them out there, every document and resource on the Internet has a unique address known as its *uniform resource locator* (URL; commonly pronounced "you-are-ell"). A URL consists of the document's name preceded by the hierarchy of directory names in which the file is stored (*pathname*), the Internet *domain name* of the server that hosts the file, and the software and manner by which the browser and the document's host server communicate to exchange the document (*protocol*):

protocol://server_domain_name/pathname

Here are some sample URLs:

http://www.kumquat.com/docs/catalog /price_list.html
price_list.html
http://www.kumquat.com/
ftp://ftp.netcom.com/pub/

The first example is an *absolute* or complete URL. It includes every part of the URL format: protocol, server, and the pathname of the document.

While absolute URLs leave nothing to the imagination, they can lead to big headaches when you move documents to another directory or server. Fortunately, browsers also let you use *relative* URLs and automatically fill in any missing portions with respective parts from the current document's *base* URL. The second example is the simplest relative URL of all; with it, the browser assumes that the *price_list.html* document is located on the same server, in the same directory as the current document, and uses the same network protocol.

Relative URLs are also useful if you don't know a directory or document's name. The third URL example, for instance, points to *kumquat.com*'s web home page. It leaves it up to the kumquat server to decide what file to send along. Typically, the server delivers the first file in the directory, one named *index.html*, or simply a listing of the directory's contents.

Although appearances may deceive, the last FTP example URL actually is absolute; it points directly at the contents of the */pub* directory.

2.7.2 Anchors

The anchor (<a>) tag is the HTML/XHTML feature for defining both the source and the destination of a hyperlink.[7] You'll most often see and use the <a> tag with its href attribute to define a source hyperlink. The value of the attribute is the URL of the destination.

> [7] The nomenclature here is a bit unfortunate: the "anchor" tag should mark just a destination, not the jumping off point of a hyperlink, too. You "drop anchor"; you don't jump off one. We won't even mention the atrociously confusing terminology the W3C uses for the various parts of a hyperlink except to say that someone got things all "bass ackwards."

The contents of the source <a> tag - the words and/or images between it and its end tag - is the portion of the document that is specially activated in the browser display and that users select to take a hyperlink. These *anchor* contents usually look different from the surrounding content (text in a different color or underlined, images with specially colored borders, or other effects), and the mouse pointer icon changes when passed over them. The <a> tag contents, therefore, should be text or an image (icons are great) that explicitly or intuitively tells users where the hyperlink will take them. Section 6.3.1

For instance, the browser will specially display and change the mouse pointer when it passes over the "Kumquat Archive" text in the following example:

```
For more information on kumquats, visit our
<a href="http://www.kumquat.com/archive.html">
Kumquat Archive</a>
```

If the user clicks the mouse button on that text, the browser automatically retrieves from the server *www.kumquat.com* a web *(http:)* page named *archive.html*, and then displays it for the user.

2.7.3 Hyperlink Names and Navigation

Pointing to another document in some collection somewhere on the other side of the world is not only cool, but it also supports your own web documents. Yet the hyperlink's chief duty is to help users navigate your collection in their search for valuable information. Hence, the concept of the home page and supporting documents has arisen.

None of your documents should run on and on. First, there's a serious performance issue: the value of your work suffers, no matter how rich it is, if the document takes forever to download and, if once retrieved, users must endlessly scroll up and down through the display to find a particular section.

Rather, design your work as a collection of several compact and succinct pages, like chapters in a book, each focused on a particular topic for quick selection and browsing by the user. Then use hyperlinks to organize that collection.

For instance, use your home page - the leading document of the collection - as a master index full of brief descriptions and respective hyperlinks to the rest of your collection.

Also use either the name variant of the <a> tag or the id attribute of nearly all tags to specially identify sections of your document. Tag ids and name anchors serve as internal hyperlink targets in your documents to help users easily navigate within the same document or jump to a particular section within another document. Refer to that id'd section in a hyperlink by appending a pound sign (#) and the section name as the suffix to the URL.

For instance, to reference a specific topic in an archive, such as "Kumquat Stew Recipes" in our example Kumquat Archive, first mark the section title with an id:

```
... preceding content...
<h3 id="Stews">Kumquat Stew Recipes</h3>
```

in the same or another document, then prepare a source hyperlink that points directly to those recipes by including the section's id value as a suffix to the document's URL, separated by a pound sign:

```
For more information on kumquats, visit our
<a href="http://www.kumquat.com/archive.html">
  Kumquat Archive</a>,
and perhaps try one or two of our
<a href="http://www.kumquat.com/archive.html#Stews">
  Kumquat Stew Recipes</a>.
```

If selected by the user, the latter hyperlink causes the browser to download the *archive.html* document and start the display at our "Stews" section.

2.7.4 Anchors Beyond

Hyperlinks are not limited to other HTML and XHTML documents. Anchors let you point to nearly any type of document available over the Internet, including other Internet services.

However, "let" and "enable" are two different things. Browsers can manage the various Internet services, like FTP and Gopher, so that users can download non-HTML documents. They don't yet fully or gracefully handle multimedia.

Today, there are few standards for the many types and formats of multimedia. Computer systems connected to the Web vary wildly in their abilities to display those sound and video formats. Except for some graphics images, standard HTML gives you no specific provision for display of multimedia documents except the ability to reference one in an anchor. The browser, which retrieves the multimedia document, must activate a special *helper* application, download and execute an associated *applet*, or have a *plug-in* accessory installed to decode and display it for the user right within the document's display.

Although HTML and most web browsers currently avoid the confusion by sidestepping it, that doesn't mean you can't or shouldn't exploit multimedia in your documents: just be aware of the limitations.

2.8 Images Are Special

Image files are multimedia elements you may reference with anchors in your document for separate download and display by the browser. But, unlike other multimedia, standard HTML and XHTML have an explicit provision for image display "inline" with the text, and images can serve as intricate maps of hyperlinks. That's because there is some consensus in the industry concerning image file formats - specifically, GIF and JPEG - and the graphical browsers have built-in decoders that integrate those image types into your document.[8]

> [8] Some browsers support other multimedia besides GIF and JPEG graphics for inline display. Internet Explorer, for instance, supports a tag that plays background audio. In addition, the HTML 4 and XHTML standards provide a way to display other types of multimedia inline with document text through a general tag.

2.8.1 Inline Images

The HTML/XHTML tag for inline images is <img>; its required src attribute is the URL of the GIF or JPEG image you want to insert in the document. Section 5.2.6

The browser separately loads images and places them into the text flow as if the image were some special, albeit sometimes very large, character. Normally, that means the browser aligns the bottom of the image to the bottom of the current line of text. You can change that with the special <img> align attribute whose value you set to put the image at the top , middle, or bottom of adjacent text. Examine Figures Figure 2-2 through Figure 2-4 for the image alignment you prefer. Of course, wide images may take up the whole line, and hence break the text flow. Or you may place an image by itself, by including preceding and following division, paragraph, or line-break tags.

Figure 2-2. An inline image aligned with the bottom of the text (default)

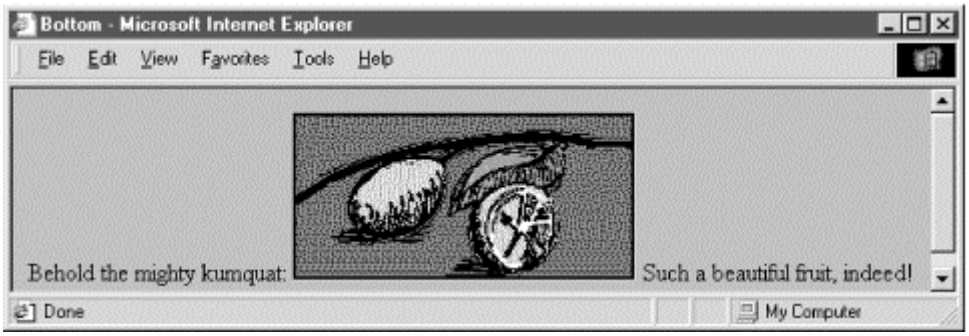


Figure 2-3. An inline image specially aligned with the middle of the text

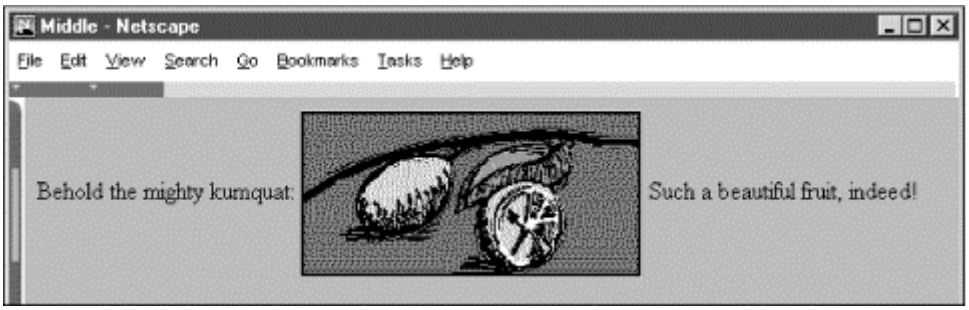
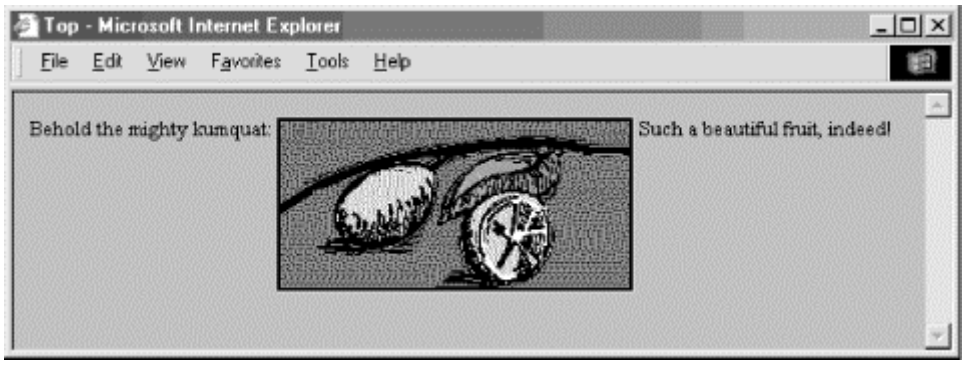


Figure 2-4. An inline image specially aligned with the top of the text



Experienced HTML authors use images not only as supporting illustrations, but also as quite small inline characters or glyphs, added to aid browsing readers' eyes and to highlight sections of the documents. Veteran HTML authors[9] commonly add custom list bullets or more distinctive section dividers than the conventional horizontal rules. Images, too, may be included in a hyperlink, so that users may select an inline thumbnail sketch to download a full-screen image. The possibilities with inline images are endless.

[9] XHTML is too new to call anyone a *veteran* or *experienced* XHTML author.

2.8.2 Image Maps

Image maps are images within an anchor with a special attribute: they may contain more than one hyperlink.

One way to enable an image map is by adding the `ismap` attribute to an `<img>` tag placed inside an anchor tag (`<a>`). When the user clicks somewhere in the image, the graphical browser sends the relative x,y coordinates of the mouse position to the server that is also designated in the anchor. A special server program then translates the image coordinates into some special action, such as downloading another document. Section 6.5.1.1

A good example of the use of an image map might be to locate a hotel while traveling. The user clicks on a map of the region they intend to visit, for instance, and your image map's server program might return the names, addresses, and phone numbers of local accommodations.

While they are very powerful and visually appealing, these so-called *server-side* image maps mean that authors must have some access to the map's coordinate-processing program on the server. Many authors don't even have access to the server, let alone a program on the server. A better solution is to take advantage of *client-side* image maps.

Rather than depending on a web server, the `usemap` attribute for the `<img>` tag along with the `<map>` and `<area>` tags allow authors to embed all the information the browser needs to process an image map in the same document as the image. Because of their reduced network bandwidth and server independence, the client-side image maps are popular among document authors and system administrators alike. Section 6.5.2

2.9 Lists, Searchable Documents, and Forms

Thought we'd exhausted text elements? Headers, paragraphs, and line breaks are just the rudimentary text-organizational elements of a document. The languages also provide several advanced text-based structures, including three types of lists, "searchable" documents, and forms. Searchable documents and forms go beyond text formatting, too; they are a way to interact with your readers. Forms let users enter text and click checkboxes and radio buttons to select particular items and then send that information back to the server. Once received, a special server application processes the form's information and responds accordingly, e.g., filling a product order or collecting data for a user survey.[10]

> [10] The server-side programming required for processing forms is beyond the scope of this book. We give some basic guidelines in the appropriate chapters, but please consult the server documentation and your server administrator for details.

The syntax for these special features and their various attributes can get rather complicated; they're not quick-start grist. So we mention them here and urge you to read on for details in later chapters.

2.9.1 Unordered, Ordered, and Definition Lists

The three types of lists match those we are most familiar with: unordered, ordered, and definition lists. An unordered list - one in which the order of items is not important, such as a laundry or grocery list - gets bounded by `<ul>` and `</ul>` tags. Each item in the list, usually a word or short phrase, is marked by the `<li>` (list-item) tag and, with XHTML, the end tag. When rendered, the list item typically appears indented from the left margin. The browser typically precedes each item with a leading bullet symbol. Section 7.1.1 Section 7.3

Ordered lists, bounded by the `<ol>` and `</ol>` tags, are identical in format to unordered ones, including the `<li>` tag (and `</li>` end tag with XHTML) for marking list items. However, the order of items is important - equipment assembly steps, for instance. The browser accordingly displays each item in the list preceded by an ascending number. Section 7.2.1

Definition lists are slightly more complicated than unordered and ordered lists. Within a definition list's enclosing `<dl>` and `</dl>` tags, each list item has two parts, each with a special tag: a short name or title, contained within a `<dt>` tag, followed by its corresponding value or definition, denoted by the `<dd>` tag (XHTML includes respective end tags). When rendered, the browser usually puts the item name on a separate line (although not indented), and the definition, which may include several paragraphs, indented below it. Section 7.5.1

The various types of lists may contain nearly any type of content normally allowed in the body of the document. So you can organize your collection of digitized family photographs into an ordered list, for example, or put them into a definition list complete with text annotations. The markup language standards even let you put lists inside of lists (nesting), opening up a wealth of interesting combinations.

2.9.2 Searchable Documents

The simplest type of user interaction provided by HTML and XHTML is the *searchable* document. You create a searchable document by including an `<isindex>` tag in its header or body. The browser automatically provides some way for the user to type one or more words into a text input box and to pass those keywords to a related processing application on the server.[11] Section 6.6.1

> [11] Few authors have used the tag, apparently. The `<isindex>` tag has been "deprecated" in HTML Version 4.0; sent out to pasture, so to speak, but not yet laid to rest.

The processing application on the server uses those keywords to do some special task, such as perform a database search or match the keywords against an authentication list to allow the user special access to some other part of your document collection.

2.9.3 Forms

Obviously, searchable documents are very limited - one per document and only one user input element. Fortunately, HTML and XHTML provide better, more extensive support for collecting user input through *forms*.

You create one or more special form sections in your document, bounded with the `<form>` and `</form>` tags. Inside the form, you may put predefined as well as customized text-input boxes allowing for both single and multiline input. You may also insert checkboxes and radio buttons for single- and multiple-choice selections, and special buttons that work to reset the form or send its contents to the server. Users fill out the form at their leisure, perhaps after reading the rest of the document, and click a special send button that makes the browser send the form's data to the server. A special server-side program you provide then processes the form and responds accordingly, perhaps by requesting more information from the user, modifying subsequent documents the server sends to the user, and so on. Section 9.2

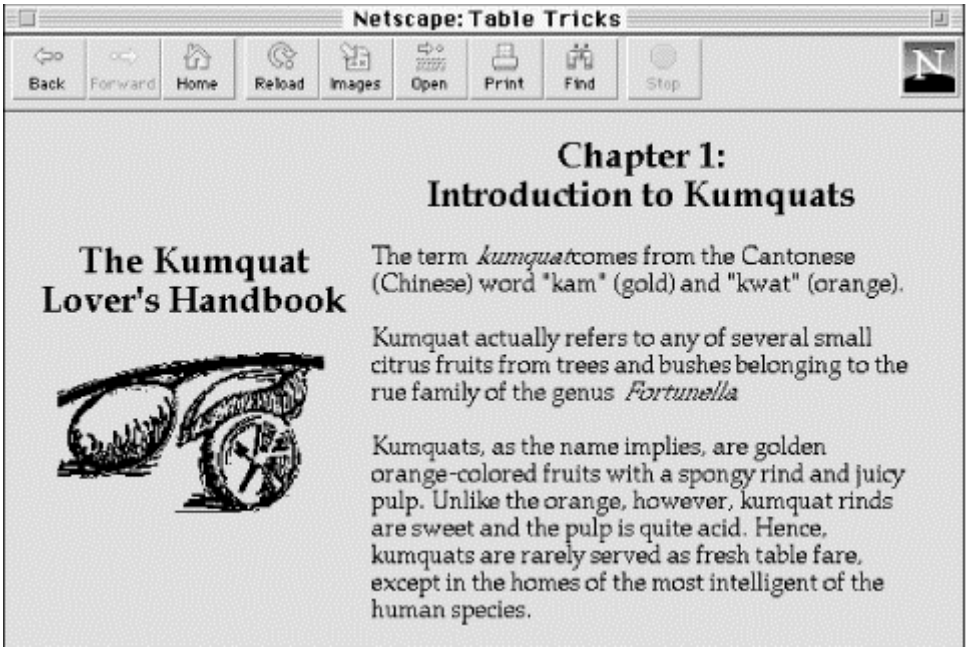
Forms provide everything you might expect of an automated form, including input area labels, integrated contents for instructions, default input values, and so on - except automatic input verification; your server-side program or client-side applets need to perform that function.

2.10 Tables

For a language that emerged from academia - a world steeped in data - it's not surprising to find that HTML, and now its progeny XHTML, support a set of tags for data tables that not only align your numbers, but can specially format your text, too.

Five tags enable tables, including the `<table>` tag itself and a `<caption>` tag for including a description of the table. Special tag attributes let you change the look and dimensions of the table. You create a table row by row, putting between the table row (`<tr>`) tag and its end tag (`</tr>`) either table header (`<th>`) or table data (`<td>`) tags and their respective contents for each cell in the table (end tags, too, with XHTML). Headers and data may contain nearly any regular content, including text, images, forms, and even another table. As a result, you can also use tables for advanced text formatting, such as for multicolumn text and sidebar headers (see Figure 2-5). For more information, see Chapter 10.

Figure 2-5. HTML tables let you perform page layout tricks, too

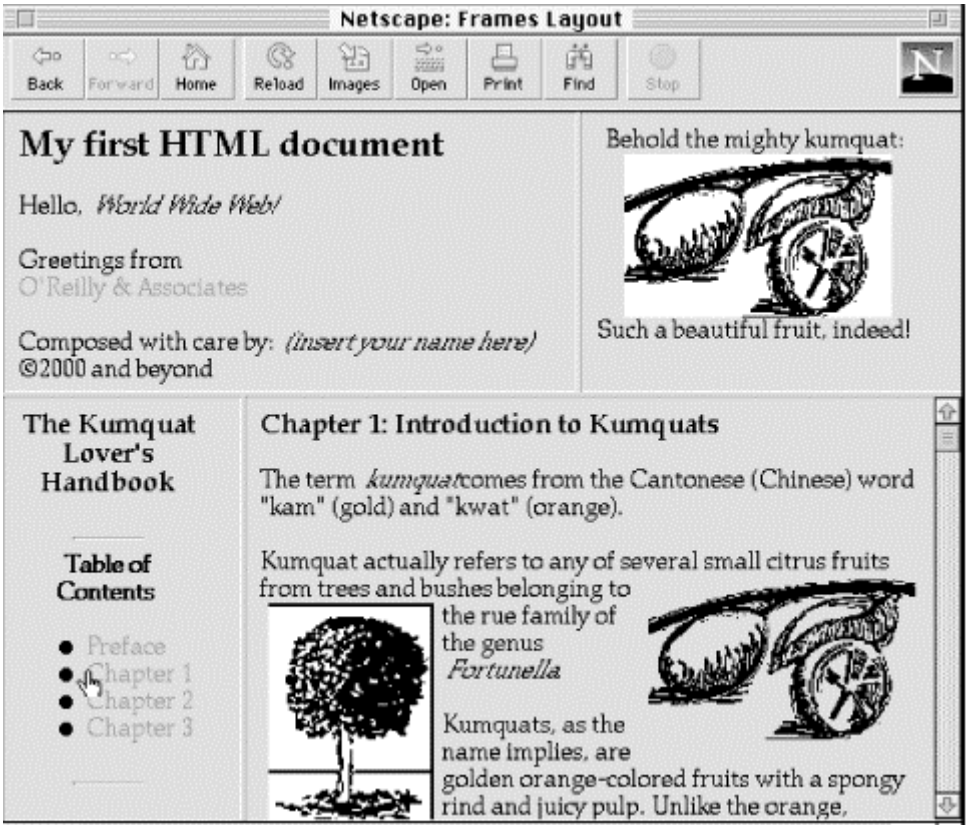


2.11 Frames

Anyone who has had more than one application window open on their graphical desktop at a time can immediately appreciate the benefits of frames. Frames let you divide the browser window into multiple display areas, each containing a different document.

Figure 2-6 is an example of a frame display. It shows how the document window may be divided into many individual windows separated by rule lines and scroll bars. What is not immediately apparent in the example, though, is that each frame may display an independent document, and not necessarily HTML or XHTML ones, either. A frame may contain any valid content that the browser is capable of displaying, including multimedia. If the frame's contents include a hypertext link the user selects, the new document's contents, even another frame document, may replace that same frame, another frame's content, or the entire browser window.

Figure 2-6. Frames divide the window into many document displays



Frames are defined in a special document in which you replace the `<body>` tag with one or more `<frameset>` tags that tell the browser how to divide its main window into discrete frames. Special `<frame>` tags go inside the `<frameset>` tag and point to the documents that go inside the frames.

The individual documents referenced and displayed in the frame document window act independently, to a degree; the frame document controls the entire window. You can, however, direct one frame's document to load new content into another frame. Selecting an item from a table of contents, for example, might cause the browser to load and display the referenced document into an adjacent frame for viewing. That way, the table of contents is always available to the user as he or she browses the collection. For more information on frames, see Chapter 11.

2.12 Style Sheets and JavaScript

Browsers also have support for two powerful innovations to HTML: style sheets and JavaScript. Like their desktop-publishing cousins, style sheets let you control how your web pages look - text font styles and sizes, colors, backgrounds, alignments, and so on. More importantly, style sheets give you a way to impose display characteristics uniformly over the entire document and over an entire collection of documents.

JavaScript is a programming language with functions and commands that let you control how the browser behaves for the user. Now, this is not a JavaScript programming book, but we do cover the language in fair detail in later chapters to show you how to embed JavaScript programs into your documents and achieve some very powerful and fun effects.

The W3C - the putative standards organization - prefers that you use the Cascading Style Sheets (CSS) model for document design. Since version 4, both Netscape and Internet Explorer support CSS and JavaScript. Netscape alone also supports a special JavaScript-based Style Sheet (JSS) model which we describe in Chapter 12 , but we do not recommend that you use it. CSS is the universally approved, universally supported way to control how your documents might (not will) usually get displayed on users' browsers.

To illustrate CSS, here's a way to make all the top-level (H1) header text in your HTML document appear in the color red:

```
<html>
<head>
<title>CSS Example</title>
<!-- Hide CSS properties within comments so old browsers
don't choke on or display the unfamiliar contents. -->
  <style type="text/CSS">
    <!--
    H1 {color: red}
    -->
  </style>
</head>
<body>
<H1>I'll be red if your browser supports CSS</H1>
Something in between.
<H1>I should be red, too!</H1>
</body>
</html>
```

Of course, you can't see red in this black & white book, so we won't show the result in a figure. Believe us or prove it to yourself by typing in and loading the example in your browser: the <H1>-enclosed text appears red on a color screen.

JavaScript is an object-based language. It views your document and the browser that displays your documents as a collection of parts ("objects") that have certain properties that you may change or compute. This is some very powerful stuff, but not something that most authors will want to handle. Rather, most of us probably will snatch the quick and easy, yet powerful JavaScript programs that proliferate across the Web and embed them in our own documents. We will tell you how in Chapter 12.

2.13 Forging Ahead

Clearly, this chapter represents the tip of the iceberg. If you've read this far, hopefully your appetite has been whetted for more. By now you've got a basic understanding of the scope and features of HTML and XHTML; proceed through subsequent chapters to expand your knowledge and learn more about each feature.

Chapter 3. Anatomy of an HTML Document

Most HTML and XHTML documents are very simple, and writing one shouldn't intimidate even the most timid of computer users. First, although you might use a fancy WYSIWYG editor to help you compose it, a document is ultimately stored, distributed, and read by a browser as a simple ASCII text file.[1] That's why even the poorest user with a barebones text editor can compose the most elaborate of web pages. (Accomplished webmasters often elicit the admiration of "newbies" by composing astonishingly cool pages using the crudest text editor on a cheap laptop computer and performing in odd places like on a bus or in the bathroom.) Authors should, however, keep several of the popular browsers on hand and alternate among them to view new documents under construction. Remember, browsers differ in how they display a page, not all browsers implement all of the language standards, and some have their own special extensions.

[1] Informally, both the text and the markup tags are ASCII characters. Technically, unless you specify otherwise, text and tags are made up of eight-bit characters as defined in the standard ISO-8859-1 Latin character set. The standards do support alternative character encoding, including Arabic and Cyrillic. See Appendix F for details.

3.1 Appearances Can Deceive

Documents never look alike when displayed by a text editor and when displayed by a browser. Take a look at any source document from the World Wide Web. At the very least, return characters, tabs, and leading spaces, although important for readability of the source text document, are ignored for the most part. There also is a lot of extra text in a source document, mostly from the display tags and interactivity markers and their parameters that affect portions of the document, but don't themselves appear in the display.

Accordingly, new authors are confronted with having to develop not only a presentation style for their web pages, but a different style for their source text. The source document's layout should highlight the programming-like markup aspects of HTML and XHTML, not their display aspects. And it should be readable not only by you, the author, but by others as well.

Experienced document writers typically adopt a programming-like style, albeit very relaxed, for their source text. We do the same throughout this book, and that style will become apparent as you compare our source examples with the actual display of the document by a browser.

Our formatting style is simple, but it serves to create readable, easily maintained documents:

- Except for the document structural tags like `<html>`, `<head>`, and `<body>`, any element we use to structure the content of a document is placed on a separate line and indented to show its nesting level within the document. Such elements include lists, forms, tables, and similar tags.
- Any element used to control the appearance or style of text is inserted in the current line of text. This includes basic font style tags like `<b>` (bold text) and document linkages like `<a>` (hypertext anchor).
- Avoid, where possible, the breaking of a URL onto two lines.
- Add extra newline characters to set apart special sections of the source document, for instance, around paragraphs or tables.

The task of maintaining the indentation of your source file ranges from trivial to onerous. Some text editors, like Emacs, manage the indentation automatically; others, like common word processors, couldn't care less about indentation and leave the task completely up to you. If your editor makes your life difficult, you might consider striking a compromise, perhaps by indenting the tags to show structure, but leaving the actual text without indentation to make modifications easier.

No matter what compromises or stands you make on source code style, it's important that you adopt one. You'll be very glad you did when you go back to that document you wrote three months ago searching for that really cool trick you did with... Now, where was that?

3.2 Structure of an HTML Document

HTML and XHTML documents consist of text, which defines the content of the document, and tags, which define the structure and appearance of the document. The structure of an HTML document is simple, consisting of an outer `<html>` tag enclosing the document head and body:[2]

[2] The structure of an XHTML document is slightly more complicated, as we detail in Chapter 16.

```
<html>
<head>
<title>Barebones HTML Document</title>
</head>
<body>
This illustrates, in a very <i>simp</i>le way,
the basic structure of an HTML document.
</body>
</html>
```

Each document has a *head* and a *body*, delimited by the `<head>` and `<body>` tags. The head is where you give your document a title and where you indicate other parameters the browser may use when displaying the document. The body is where you put the actual contents of the document. This includes the text for display and document control markers (tags) that advise the browser how to display the text. Tags also reference special-effects files, including graphics and sound, and indicate the hot spots (*hyperlinks* and *anchors*) that link your document to other documents.

3.3 Tags and Attributes

For the most part, tags - the markup elements of HTML and XHTML - are simple to understand and use, since they are made up of common words, abbreviations, and notations. For instance, the `<i>` and `</i>` tags tell the browser respectively to start and stop italicizing the text characters that come between them. Accordingly, the syllable "simp" in our barebones example above would appear italicized on a browser display.

The HTML and XHTML standards and their various extensions define how and where you place tags within a document. Let's take a closer look at that syntactic sugar that holds together all documents.

3.3.1 The Syntax of a Tag

Every tag consists of a tag *name*, sometimes followed by an optional list of tag *attributes*, all placed between opening and closing brackets (< and >). The simplest tag is nothing more than a name appropriately enclosed in brackets, such as `<head>` and `<i>`. More complicated tags contain one or more *attributes*, which specify or modify the behavior of the tag.

According to the HTML standard, tag and attribute names are not case-sensitive. There's no difference in effect between `<head>`, `<Head>`, `<HEAD>`, or even `<HeaD>`; they are all equivalent. With XHTML, case *is* important: all current standard tag and attribute names are in lowercase.

For both HTML and XHTML, the values that you assign to a particular attribute may be case-sensitive, depending on your browser and server. In particular, file location and name references - or uniform resource locators (URLs) - are case-sensitive. Section 6.2

Tag attributes, if any, belong after the tag name, each separated by one or more tab, space, or return characters. Their order of appearance is not important.

A tag attribute's value, if any, follows an equal sign (=) after the attribute name. You may include spaces around the equal sign, so that `width=6`, `width = 6`, `width =6`, and `width= 6` all mean the same. For readability, however, we prefer not to include spaces. That way, it's easier to pick out an attribute/value pair from a crowd of pairs in a lengthy tag.

With HTML, if an attribute's value is a single word or number (no spaces), you may simply add it after the equal sign. All other values should be enclosed in single or double quotation marks, especially those values that contain several words separated by spaces. With XHTML, all attribute values must be enclosed in double-quotes. The length of the value is limited to 1024 characters.

Most browsers are tolerant of how tags are punctuated and broken across lines. Nonetheless, avoid breaking tags across lines in your source document whenever possible. This rule promotes readability and reduces potential errors in your HTML documents.

3.3.2 Sample Tags

Here are some tags with attributes:

```
<a href="http://www.oreilly.com/catalog.html">
<ul compact>
<ul compact="compact">
<input type=text name=filename size=24 maxlength=80>
<link title="Table of Contents">
```

The first example is the `<a>` tag for a hyperlink to O'Reilly & Associates' World Wide Web-based catalog of products. It has a single attribute, `href`, followed by the catalog's address in cyberspace - its URL.

The second example shows an HTML tag that formats text into an unordered list of items. Its single attribute - `compact`, which limits the space between list items - does not require a value.

The third example demonstrates how the second example must be written in XHTML. Notice the `compact` attribute now has a value, albeit redundant, and that its value is enclosed in double quotes.

The fourth example shows an HTML tag with multiple attributes, each with a value that does not require enclosing quotation marks. Of course, with XHTML, each attribute value must be enclosed in double quotes.

The last example shows proper use of enclosing quotation marks when the attribute value is more than one word long.

What is not immediately evident in these examples is that while HTML attribute names are not case-sensitive (`href` works the same as `HREF` and `HreF` in HTML), most attribute values are case-sensitive. The value `filename` for the `name` attribute in the `<input>` tag example is not the same as the value `Filename`, for instance.

3.3.3 Starting and Ending Tags

We alluded earlier to the fact that most tags have a beginning and an end and affect the portion of content between them. That enclosed segment may be large or small, from a single text character, syllable, or word, such as the italicized "simp" syllable in our barebones example, to the `<html>` tag that bounds the entire document. The starting component of any tag is the tag name and its attributes, if any. The corresponding ending tag is the tag name alone, preceded by a slash. Ending tags have no attributes.

3.3.4 Proper and Improper Nesting

Tags can be put inside the affected segment of another tag (nested) for multiple tag effects on a single segment of the document. For example, a portion of the following text is both bold and included as part of an anchor defined by the `<a>` tag:

```
<body>
This is some text in the body, with a
<a href="another_doc.html">link, a portion of which
is <b>set in bold</b></a>
</body>
```

According to the HTML and XHTML standards, you must end nested tags starting with the most recent one and work your way back out. For instance in the example, we end the bold tag (`</b>`) before ending the link tag (`</a>`) since we started in the reverse order: `<a>` tag first, then `<b>` tag. It's a good idea to follow that standard, even though most browsers don't absolutely insist you do so. You may get away with violating this nesting rule for one browser, sometimes even with all current browsers. But eventually a new browser version won't allow the violation and you'll be hard pressed to straighten out your source HTML document. And, be aware that the XHTML standard explicitly forbids improper nesting.

3.3.5 Tags Without Ends

According to the HTML standard, a few tags do not have an ending tag. In fact, the standard forbids use of an end tag for these special ones, although most browsers are lenient and ignore the errant end tag. For example, the `<br>` tag causes a line break; it has no effect otherwise on the subsequent portion of the document and, hence, does not need an ending tag.

The HTML tags that do not have a corresponding end tags are:

```
<area>          <base>          <basefont>
<br>            <col>           <frame>
<hr>            <img>           <input>
<isindex>       <link>          <meta>
<param>
```

XHTML always requires end tags. Section 16.3.3

3.3.6 Omitting Tags

You often see documents in which the author seemingly has forgotten to include an ending tag in apparent violation of the HTML standard. Sometimes you even see a missing `<body>` tag. But your browser doesn't complain, and the document displays just fine. What gives? The HTML standard lets you omit certain tags or their endings for clarity and ease of preparation. The HTML standard writers didn't intend the language to be tedious.

For example, the `<p>` tag that defines the start of a paragraph has a corresponding end tag `</p>`, but the `</p>` ending tag rarely is used. In fact, many HTML authors don't even know it exists! Section 4.1.2

Rather, the HTML standard lets you omit a starting tag or ending tag whenever it can be unambiguously inferred by the surrounding context. Many browsers make good guesses when confronted with missing tags, leading the document author to assume that a valid omission was made.

We recommend that you most always add the ending tag. It'll make life easier for yourself as you transition to XHTML, as well as on the browser and anyone who might need to modify your document in the future.

3.3.7 Ignored or Redundant Tags

HTML browsers sometimes ignore tags. This usually happens with redundant tags whose effects merely cancel or substitute for themselves. The best example is a series of `<p>` tags, one after the other with no intervening content. Unlike the similar series of repeating return characters in a text-processing document, most browsers skip to a new line only once. The extra `<p>` tags are redundant and usually ignored by the browser.

In addition, most HTML browsers ignore any tag that they don't understand or that was incorrectly specified by the document author. Browsers habitually forge ahead and make some sense of a document, no matter how badly formed and error-ridden it may be. This isn't just a tactic to overcome errors; it's also an important strategy for extensibility. Imagine how much harder it would be to add new features to the language if the existing base of browsers choked on them.

The thing to watch out for with nonstandard tags that aren't supported by most browsers is their enclosed contents, if any. Browsers that recognize the new tag may process those contents differently than those that don't support the new tag. For example, Internet Explorer and Netscape Navigator now both support the `<style>` tag, whose contents serve to set the variety of display characteristics of your document. However, previous versions of the popular browsers, many of which are still in use by many people today, don't support styles. Hence, older browsers ignore the `<style>` tag and render its contents on the user's screen, effectively defeating the tag's purpose in addition to ruining the document's appearance. Section 8.1.2

3.4 Well-Formed Documents and XHTML

XHTML is HTML's prissy cousin. What would pass most beauty contests as a very proper and complete HTML document, done according to the book including end-paragraph tags, would get rejected by the XML judges as a malformed file.

To conform with XML, XHTML insists that documents be "well-formed." Among other things, that means every tag must have an ending tag, even the ones like `<br>` and `<hr>` that the HTML standard forbids the use of an end tag. With XHTML, the ending is placed inside the start tag: `<br />`, for example. Section 16.3.3

It also means that tag and attribute names are case-sensitive, and according to the current XHTML standard, must be in lowercase. Hence, only `<head>` is acceptable, and it is *not* the same as `<HEAD>` or `<HeAd>`, as it is with the HTML standard. Section 16.3.4

And, too, well-formed XHTML documents, like HTML standard ones, conform to proper nesting. No argument there. Section 16.3.1

In its defense, the XML standard and its offspring XHTML emphasize extensibility. That way, `<p>` can mean the beginning of a paragraph in HTML, whereas another variant of the language may define the contents of the `<P>` tag to be election-poll results, whose display is quite different, perhaps in tabular form with red, white, and blue stripes and accompanying patriotic music.

More about this in Chapter 15 and Chapter 16, in which we detail XML and XHTML standards (and the Forces of Conformity).

3.5 Document Content

Nearly everything else you put into your HTML or XHTML document that isn't a tag is by definition content, and the majority of that is text. Like tags, document content is encoded using a specific character set, the ISO-8859-1 Latin character set, by default. This character set is a superset of conventional ASCII, adding the necessary characters to support the Western European languages. If your keyboard does not allow you to directly enter the characters you need, you can use character entities to insert the desired characters.

3.5.1 Advice Versus Control

Perhaps the hardest rule to remember when marking up an HTML or XHTML document is that all the tags you insert regarding text display and formatting are only advice for the browser: they do not explicitly control how the browser will display the document. In fact, the browser can choose to ignore all of your tags and do what it pleases with the document content. What's worse, the user (of all people!) has control over the text-display characteristics of his or her own browser.

Get used to this lack of control. The best way to use markup to control the appearance of your documents is to concentrate on the content of the document, not on its final appearance. If you find yourself worrying excessively about spacing, alignment, text breaks, and character positioning, you'll surely end up with ulcers. You will have gone beyond the intent of HTML. If you focus on delivering information to users in an attractive manner, using the tags to advise the browser as to how best to display that information, you are using HTML or XHTML effectively, and your documents will render well on a wide range of browsers.

3.5.2 Character Entities

Besides common text, HTML and XHTML give you a way to display special text characters that you might not normally be able to include in your source document or that have other purposes. A good example is the less-than or opening bracket (<) symbol. In HTML, it normally signifies the start of a tag, so if you insert it simply as part of your text, the browser will get confused and probably misinterpret your document.

For both HTML and XHTML, the ampersand character instructs the browser to use a special character, formally known as a *character entity*. For example, the command `<` inserts that pesky less-than symbol into the rendered text. Similarly, `>` inserts the greater-than symbol, and `&` inserts an ampersand. There can be no spaces between the ampersand, the entity name, and the required, trailing semicolon. (Semicolons aren't special characters; you don't need to use an ampersand sequence to display a semicolon normally.) Section 16.3.7

You also may replace the entity name after the ampersand with a pound symbol (#) and a decimal value corresponding to the entity's position in the character set. Hence, the sequence `<` does the same thing as `<` and represents the less-than symbol. In fact, you could substitute all the normal characters within an HTML document with ampersand-special characters, such as `A` for a capital "A" or `a` for its lowercase version, but that would be silly. A complete listing of all characters, their names, and numerical equivalents can be found in Appendix F.

Keep in mind that not all special characters can be rendered by all browsers. Some browsers just ignore many of the special characters; with others, the characters aren't available in the character sets on a specific platform. Be sure to test your documents on a range of browsers before electing to use some of the more obscure character entities.

3.5.3 Comments

Comments are another type of textual content that appear in the source HTML document, but are not rendered by the user's browser. Comments fall between the special `<!--` and `-->` markup elements. Browsers ignore the text between the comment character sequences.

Here's a sample comment:

```
<!-- This is a comment -->
<!-- This is a
multiple line comment
that ends on this line -->
```

There must be a space after the initial `<!--` and preceding the final `-->`, but otherwise you can put nearly anything inside the comment. The biggest exception to this rule is that the HTML standard doesn't let you nest comments.[3]

[3] Netscape does let you nest comments, but the practice is tricky; you cannot always predict how other browsers will react to nested comments.

Internet Explorer also lets you place comments within a special <comment> tag. Everything between the <comment> and </comment> tag is ignored by Internet Explorer, but all other browsers will display the comment to the user. Because of this undesirable behavior, we do not recommend using the <comment> tag for comments. Instead, always use the <!-- and --> sequences to delimit comments.

Besides the obvious use of comments for source documentation, many web servers use comments to take advantage of features specific to the document server software. These servers scan the document for specific character sequences within conventional HTML comments and then perform some action based upon the commands embedded in the comments. The action might be as simple as including text from another file (known as a *server-side include*) or as complex as executing other commands on the server to generate the document contents dynamically.

3.6 HTML Document Elements

Every HTML document should conform to the HTML SGML DTD, the formal Document Type Definition that defines the HTML standard. The DTD defines the tags and syntax that are used to create an HTML document. You can inform the browser which DTD your document complies with by placing a special SGML (Standard Generalized Markup Language) command in the first line of the document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
```

This cryptic message indicates that your document is intended to be compliant with the HTML 4.01 final DTD defined by the World Wide Web Consortium (W3C). Other versions of the DTD define more restricted versions of the HTML standard, and not all browsers support all versions of the HTML DTD. In fact, specifying any other doctype may cause the browser to misinterpret your document when displaying it for the user. It's also unclear what doctype to use when including in the HTML document the various tags that are not standards, but are very popular features of a popular browser - the Netscape extensions, for instance, or even the deprecated HTML 3.0 standard, for which a DTD was never released.

Almost no one precedes their HTML documents with the SGML doctype command. Because of the confusion of versions and standards, we don't recommend that you include the prefix with your HTML documents either.

On the other hand, we do strongly recommend that you include the proper doctype statement in your XHTML documents, in conformance with XML standards. Read Chapter 15 and Chapter 16 for more about DTDs and the new Extensible Markup Language standards.

3.6.1 The <html> Tag

As we saw earlier, the <html> and </html> tags serve to delimit the beginning and ending of a document. Since the typical browser can easily infer from the enclosed source that it is an HTML document, you don't really need to include the tag in your source HTML document.

<html>

Function:

Delimits a complete HTML document

Attributes:

DIR
LANG
VERSION

End tag:

</html>; may be omitted in HTML

Contains:

head_tag, body_tag, frames

That said, it's considered good form to include this tag so that other tools, particularly more mundane text-processing ones, can recognize your document as an HTML document. At the very least, the presence of the beginning and ending <html> tags ensures that the beginning or the end of the document haven't been inadvertently deleted. Besides, XHTML requires the <html> tag.

Inside the <html> tag and its end tag are the document's head and body. Within the head, you'll find tags that identify the document and define its place within a document collection. Within the body is the actual document content, defined by tags that determine the layout and appearance of the document text. As you might expect, the document head is contained within a <head> tag and the body is within a <body> tag, both of which are defined later.

The <body> tag may be replaced by a <frameset> tag, defining one or more display frames that, in turn, contain actual document content. See Chapter 11 for more information. By far, the most common form of the <html> tag in HTML documents is simply:

```
<html>
document head and body content
</html>
```

When the <html> tag appears without the version attribute, the HTML document server and browser assume the version of HTML used in this document is supplied to the browser by the server.

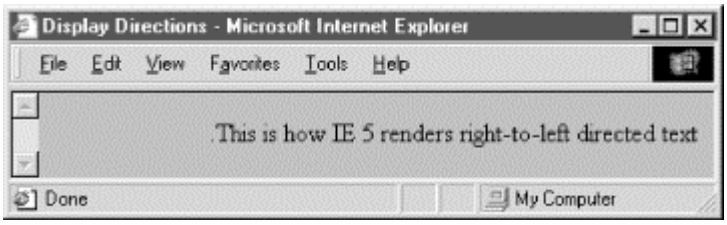
3.6.1.1 The dir attribute

The dir attribute specifies in which direction the browser should render text within the containing element. When used within the <html> tag, it determines how text will be presented within the entire document. When used within another tag, it controls the text's direction for just the content of that tag.

By default, the value of this tag is ltr, indicating that text is presented to the user left-to-right. Use the other value, rtl, to display text right-to-left for languages like Chinese or Hebrew.Of course, the results depend on your content and the browser's support of HTML 4.Netscape and Internet Explorer Versions 4 and earlier ignore the dir attribute. The HTML 4-compliant Internet Explorer Version 5 simply right-justifies dir=rtl text, although if you look in Figure 3-1, you'll notice the browser moves the punctuation (the period) to the other side of the sentence:

```
<html dir=rtl>
<head>
<title>Display Directions</title>
</head>
<body>
This is how IE 5 renders right-to-left directed text.
</body>
</html>
```

Figure 3-1. Internet Explorer 5 implements the dir attribute



3.6.1.2 The lang attribute

When included within the <html> tag, the lang attribute specifies the language you've generally used within the document. When used within other tags, the lang attribute specifies the language you used within that tag's content. Ideally, the browser will use lang to better render the text for the user.

Set the value of the lang attribute to an ISO-639 standard two-character language code. You may also indicate a dialect by following the ISO language code with a dash and a subcode name. For example, "en" is the ISO language code for English; "en-US" is the complete code for US English. Other common language codes include "fr" (French), "de" (German), "it" (Italian), "nl" (Dutch), "el" (Greek), "es" (Spanish), "pt" (Portuguese), "ar" (Arabic), "he" (Hebrew), "ru" (Russian), "zh" (Chinese), "ja" (Japanese), and "hi" (Hindi).

3.6.1.3 The version attribute

The version attribute defines the HTML standard version used to compose the document. Its value, for HTML Version 4.01, should read exactly:

```
version="-//W3C//DTD HTML 4.01//EN"
```

In general, version information within the <html> tag is more trouble than it is worth, and this attribute has been deprecated in HTML 4. Serious authors should instead use an SGML <!doctype> tag at the beginning of their documents, like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01//EN"
                  "http://www.w3c.org/TR/html4/strict.dtd">
```

3.7 The Document Header

The document header describes the various properties of the document, including its title, position within the Web, and relationship with other documents. Most of the data contained within the document header is never actually rendered as content visible to the user.

3.7.1 The <head> Tag

The <head> tag serves to encapsulate the other header tags. Place it at the beginning of your document, just after the <html> tag and before the <body> or <frameset> tag. Both the <head> tag and its corresponding ending </head> can be unambiguously inferred by the browser and so can be safely omitted from a document. Nonetheless, we do encourage you to include them in your documents, since they promote readability and support document automation.

<head>

Function:

Defines the document header

Attributes:

DIR
LANG
PROFILE

End tag:

</head>; rarely omitted in HTML

Contains:

head_content

Used in:

html_tag

The <head> tag may contain a number of other tags that help define and manage the document's content. These include, in any order of appearance: <base>, <isindex>, <link>, <meta>, <nextid>, <object>, <script>, <style>, and <title>.

3.7.1.1 The dir and lang attributes

The dir and lang attributes help extend HTML and XHTML to an international audience. Section 3.6.1.1, Section 3.6.1.2

3.7.1.2 The profile attribute

Often, the header of a document contains a number of <meta> tags used to convey additional information about the document to the browser. In the future, authors may use predefined profiles of standard document metadata to better describe their documents. The profile attribute supplies the URL of the profile associated with the current document.

The format of a profile and how it might be used by a browser are not yet defined; this attribute is primarily a placeholder for future development.

3.7.2 The <title> Tag

The <title> tag does exactly what you might expect: the words you place inside its start and end tags define the title for your document. (This stuff is pretty much self-explanatory and easier than you might think at first glance.) The title is used by the browser in some special manner, most often placed in the browser window's title bar or on a status line. Usually, too, the title becomes the default name for a link to the document if the document is added to a link collection or to a user's "hot list."

<title>

Function:

Defines the document title

Attributes:

DIR
LANG

End tag:

</title>; never omitted

Contains:

plain_text

Used in:

head_content

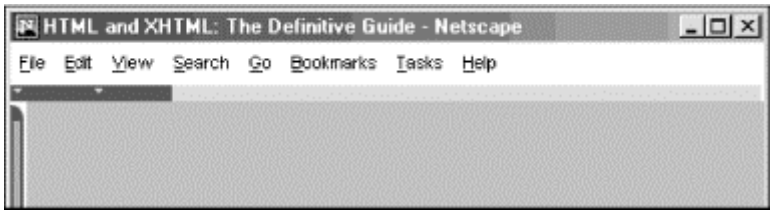
The <title> tag is the only thing required within the <head> tag. Since the <head> tag itself and even the <html> tag may be safely omitted, the <title> tag could be the first line within a valid HTML document. Beyond that, most browsers will even supply a generic title for documents lacking a <title> tag, such as the document's filename, so you don't even have to supply a title. That goes a bit too far even for our down-and-dirty tastes. No respectable author should serve up a document missing the <title> tag and a title.

Browsers do not specially format title text and ignore anything other than text inside the title start and end tags. For instance, they will ignore any images or links to other documents.

Here's an even barer barebones example of a valid HTML document to highlight the header and title tags; watch what happens when Netscape displays it in Figure 3-2:

```
<html>
<head>
<title>HTML and XHTML: The Definitive Guide</title>
</head>
</html>
```

Figure 3-2. What's in a <title>?



3.7.2.1 What's in a title?

Selecting the right title is crucial to defining a document and ensuring that it can be effectively used within the World Wide Web.

Keep in mind that users can access each of your documents in a collection in nearly any order and independently of one another. Each document's title should therefore define the document both within the context of your other documents as well as on its own merits.

Titles that include references to document sequencing are usually inappropriate. Simple titles, like "Chapter 2" or "Part VI" do little to help a user understand what the document might contain. More descriptive titles, such as "Chapter 2: Advanced Square Dancing" or "Part VI: Churchill's Youth and Adulthood," convey both a sense of place within a larger set of documents and specific content that invites the reader to read on.

Self-referential titles also aren't very useful. A title like "My Home Page" is completely content-free, as are titles like "Feedback Page" or "Popular Links." You want a title to convey a sense of content and purpose so that users can decide, based upon the title alone, whether to visit that page or not. "The Kumquat Lover's Home Page" is descriptive and likely to draw in lovers of the bitter fruit, as are "Kumquat Lover's Feedback Page" and "Popular Links Frequented by Kumquat Lovers."

People spend a great deal of time creating documents for the Web, often only to squander that effort with an uninviting, ineffective title. As special software that automatically collects links for users becomes more prevalent on the Web, the only descriptive phrase associated with your pages when they are inserted into some vast link database will be the title you choose for them. We can't emphasize this enough: take care to select descriptive, useful, context-independent titles for each of your documents.

3.7.2.2 The dir and lang attributes

The `dir` and `lang` attributes help extend HTML and XHTML to an international audience. Section 3.6.1.1, Section 3.6.1.2

3.7.3 Related Header Tags

Other tags you may include within the `<head>` tag deal with specific aspects of document creation, management, linking, automation, or layout. That's why we only mention them here and describe them in greater detail in other, more appropriate sections and chapters of this book.

Briefly, the special header tags are:

`<base>` and `<link>`

: Define the current document's base location and relationship to other documents. Section 6.7.1, Section 6.7.2

`<isindex>`

: Deprecated in HTML 4, the `<isindex>` tag at one time could be used to create automatic document indexing forms, allowing users to search databases of information using the current document as a querying tool. Section 6.6.1

`<nextid>`

: Not supported in HTML 4 or XHTML, the `<nextid>` tag makes creation of unique document labels easier when using document automation tools. Section 6.8.2

`<meta>`

: Provides additional document data not supplied by any of the other `<head>` tags. Section 6.8.1

`<object>`

: Defines methods by which nonstandard objects can be rendered by the browser. Section 12.2.1

`<script>`

: Defines one or more scripts that can be invoked by elements within the document. Section 12.3.1

`<style>`

: Lets you create Cascading Style Sheet properties to control body-content display characteristics for the entire document. Section 8.1.2

3.8 The Document Body

The document body is the meat of the matter; it's where you put the contents of your document. The `<body>` tag delimits the document body.

3.8.1 The <body> Tag

Within HTML 4 and XHTML, the `<body>` tag has a number of attributes that control the color and background of your document. Various browsers, have extended the tag to give even greater control over your document's appearance.

<body>

Function:

Defines the document body

Attributes:

ALINK	ONKEYUP
BACKGROUND	ONLOAD
BGCOLOR	ONMOUSEDOWN
BGPROPERTIES I	ONMOUSEMOVE
CLASS	ONMOUSEOUT
DIR	ONMOUSEOVER
ID	ONMOUSEUP
LANG	ONUNLOAD
LEFTMARGIN I	STYLE
LINK	TEXT
ONBLUE	TITLE
ONCLICK	TOPMARGIN I
ONDBLCLICK	VLINK
ONFOCUS	
ONKEYDOWN	
ONKEYPRESS	

End tag:

</body>; may be omitted in HTML

Contains:

body_content

Used in:

html_tag

Anything inside the <body> tag and its ending counterpart </body> is called *body content*. The simplest document might have only a sequence of text paragraphs within the <body> tag. More complex documents will include heavily formatted text, graphical figures, tables, and a variety of special effects.

Since the position of the <body> and </body> tags can be inferred by the browser, they can safely be omitted from the document. However, like the <html> and <head> tags, we recommend that you include the <body> tags in your document to make them more easily readable and maintainable.

The various attributes for the <body> tag can be loosely grouped into three sets: those that give you some control over the document's appearance, those that associate programmable functions with the document itself, and those that label and identify the body for later reference. We address the appearance attributes (alink, background, bgcolor, bgproperties, leftmargin, link, text, topmargin, and vlink) in Chapter 5; the class and style attributes for cascading style sheets in Chapter 8; JavaScript style sheets and the programmatic attributes (the "on-event" ones) in Chapter 12; the language attributes (dir and lang) earlier in this chapter in Section 3.5.1 and Section 3.5.2; and the identification attributes (id and title) in Chapter 4. Section 3.6.1.1, Section 3.6.1.2, Section 4.1.1.4, Section 4.1.1.5

3.8.2 Frames

The HTML and XHTML standards define a special type of document in which you replace the <body> tag with one or more <frameset> tags. This so-called *frame* document divides the display window into one or more independent windows, each displaying a different document. We thoroughly describe this innovation in Chapter 11.

3.9 Editorial Markup

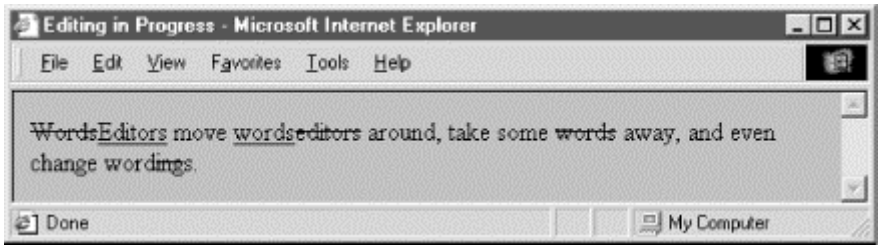
HTML 4.0 introduced two new tags that can help groups of authors collaborate in the development of documents and maintain some semblance of editorial and version control. The insert (<ins>) and delete (<del>) tags let you designate portions of your document's body as either new or added content, or designate old stuff that should be replaced. And with special attributes, you may indicate when you made the change (datetime) and a reference to a document that may explain the change (cite).

3.9.1 The <ins> and <del> Tags

The <ins> and <del> tags let authors set off portions of body contents they intend to add to or delete from the current version of their document. HTML 4/XHTML-compliant browsers display the contents of the <ins> or <del> tags in some special way so readers can quickly scan the document for the changes.

Netscape 4 and earlier versions ignore the tags, as did Internet Explorer 4 and earlier versions. The newest versions of Internet Explorer (Version 5) and Netscape (Version 6) use common editorial markings by underlining inserted text and striking out deleted text (Figure 3-3).

Figure 3-3. Internet Explorer Version 5 displays <ins> and <del>-tagged content



3.9.1.1 The cite attribute

The cite attribute lets you document the reasons for the insertion or deletion. Its value must be a URL that points to some other document that explains the inserted text. How cite gets treated by a browser is a question for the future.

<ins> and <del>

Function:

Defines inserted and deleted document content

Attributes:

CITE	ONKEYPRESS
CLASS	ONKEYUP
DATETIME	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE

End tag:

</ins> and </del>; never omitted

Contains:

body_content

Used in:

body_content

3.9.1.2 The datetime attribute

Although the reason for the change is important, knowing when a change was made is often more important. The `datetime` attribute for the `<ins>` and `<del>` tags takes a single value: a specially encoded date and time stamp.

The rigorous format for the datetime value is `YYYY-MM-DDThh:mm:ssTZD`. The components are:

- `YYYY` is the year, such as 1998 or 2001.
- `MM` is the month; 01 for January through 12 for December.
- `DD` is the day; 01 through 31.
- `T` is a required character designating the beginning of the time segment of the stamp.
- `hh` is the hour in 24-hour format; 00 (midnight) through 23 (11 P.M.). (Add a following colon if you include the minutes.)
- `mm` are the minutes on the hour; 00 through 59. (Add a following colon if you include the seconds.)
- `ss` are the seconds; 00 through 59.
- `TZD` is the time zone designator. It can be one of three values: `Z`, indicating Greenwich Mean Time,[4] or the hours, minutes, and seconds before (`-`) or after (`+`) Coordinated Universal Time (UTC) where time is relative to the time in Greenwich, England.

[4] Greenwich Mean Time is also known as "Zulu," thus the value of "Z."

For example:

```
1998-02-22T14:26Z
```

decodes to February 22, 1998 at 2:26 P.M. Greenwich Mean Time. To specify Eastern Standard Time, the code for the same time and date is:

```
1998-02-22T09:26-05:00
```

Notice that the local time zone may change depending on where the document gets edited, whereas the universal time will stay the same.

3.9.1.3 The class, dir, event, id, lang, style, title, and events attributes

There are several nearly universal attributes for the many HTML and XHTML tags. These attributes give you a common way to identify (`title`) and label (`id`) a tag's contents for later reference or automated treatment; to change the contents' display characteristics (`class`, `style`); and to reference the language used (`lang`) and related direction the text should flow (`dir`). There are also input events that may happen in and around the tagged contents that you may react to via an `on`-event attribute and some programming. Section 3.6.1.1, Section 3.6.1.2, Section 4.1.1.4, Section 4.1.1.5, Section 8.1.1, Section 8.3, Section 12.3.3

3.9.2 Using Editorial Markup

The uses of `<ins>` and `<del>` are obvious to anyone who has used a "boilerplate" document or form, or who has collaborated with others in the preparation of a document.

For example, law firms typically have a collection of online legal documents that are specially completed for each client. Law clerks usually do the "fill in," and the final document gets reviewed by a lawyer. To highlight where the clerk made changes in the document so that they are readily evident to the reviewer, use the `<ins>` tag to indicate the clerk's added text and the `<del>` tag to mark the text that was replaced. Optionally use the `cite` and `datetime` attributes to indicate when and why the changes were made.

For example, the clerk might fill in a boilerplate document with the law firm's and representative's names, indicating the time and source for the change:

```
The party of the first part, as represented by
<ins datetime=1998-06-22T08:30Z
     cite="http://www.mull+dull.com/tom_duller.html">
Thomas Muller of Muller and Duller
</ins>
<del>[insert representation here]</del>
```

The editorial markup tags could also be used by editing tools to denote how documents were modified as authors make changes over a period of time. With the correct use of the `cite` and `datetime` attributes, it would be possible to recreate a version of a document from a specific point in time.

3.10 The <bdo> Tag

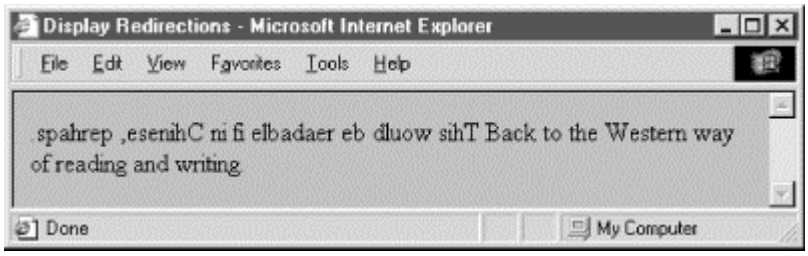
As we've mentioned earlier, the authors of the HTML 4 standard have made a concerted effort to include standard ways web agents (browsers) are supposed to treat and display the many different human languages and dialects. Accordingly, the HTML 4 standard and its progeny XHTML contain the universal `dir` and `lang` attributes that let you explicitly advise the browser that the whole document or specific tagged segments within it are in a particular language. These language-related attributes, then, may affect some display characteristics; for example, the `dir` attribute tells the browser to write the words across the display from either left to right (`dir=ltr`), as for most Western languages, or right to left (`dir=rtl`), as for many Asian languages. Section 3.6.1.1, Section 3.6.1.2

The various Unicode and ISO standards for language encoding and display may conflict with your best intentions. In particular, the contents of some other documents, such as a MIME-encoded file, already may be properly formatted and your document may misadvise the browser to undo that encoding. Hence, the HTML 4 and XHTML standards have the `<bdo>` tag. With it, you override any current and inherited `dir` specifications. And with the tag's required `dir` attribute, you definitively specify the direction in which the tag's contents should be displayed.

For example, Figure 3-4 shows how Internet Explorer Version 5 handles the following HTML fragment containing a `<bdo>` redirection:

```
<bdo dir=rtl>This would be readable if in Chinese, perhaps.</bdo>
Back to the Western way of reading and writing.
```

Figure 3-4. Tricks with <bdo> redirected text flow



Admittedly, the effects of the `<bdo>` tag are a bit esoteric and the opportunities to use it currently are rare, particularly considering that the second most popular browser doesn't yet support it.

<bdo>

Function:

Overrides bidirectional algorithms for content display

Attributes:

CLASS	LANG
DIR	STYLE
ID	TITLE

End tag:

</bdo>; never omitted

Contains:

text

Used in:

body_content

Chapter 4. Text Basics

Any successful presentation, even a thoughtful tome, should have its text organized into an attractive, effective document. Organizing text into attractive documents is HTML and XHTML's forte. The languages give you a number of tools that help you mold your text and get your message across. They also help structure your document so that your target audience has easy access to your words.

Always keep in mind while designing your documents (here we go again!) that the markup tags, particularly in regard to text, only advise - they do not dictate - how a browser will ultimately render the document. Rendering varies from browser to browser. Don't get too entangled with trying to get just the right look and layout. Your attempts may and probably will be thwarted by the browser.

4.1 Divisions and Paragraphs

Like most text processors, a browser wraps the words it finds to fit the horizontal width of its viewing window. Widen the browser's window and words automatically flow up to fill the wider lines. Squeeze the window and words wrap downwards.

Unlike most text processors, however, HTML and XHTML use explicit division (`<div>`), paragraph (`<p>`), and line-break (`<br>`) tags to control the alignment and flow of text. Return characters, although quite useful for readability of the source document, typically are ignored by the browser - authors must use the `<br>` tag to explicitly force a common text line break. The `<p>` tag, while also performing the task, carries with it meaning and effects beyond a simple line break.

The `<div>` tag is a little different. Originally codified in the HTML 3.2 standard, `<div>` was included in the language to be a simple organizational tool - to divide the document into discrete sections - whose somewhat obtuse meaning meant few authors used it. But recent innovations - alignment, styles, and the `id` attribute for document referencing and automation - now let you more distinctly label and thereby define individual sections of your documents, as well as control the alignment and appearance of those sections. These features breathe real life and meaning into the `<div>` tag.

By associating an `id` and a `class` name with the various sections of your document, each delimited by a `<div id=name class=name>` tag and attributes (you can do the same with other tags like `<p>`, too), you not only label those divisions for later reference by a hyperlink and for automated processing and management (collect all the bibliography divisions, for instance), but you may also define different, distinct display styles for those portions of your document. For instance, you might define one divisional class for your document's abstract (`<div class=abstract>`, for example), another for the body, a third for the conclusion, and a fourth divisional class for the bibliography (`<div class=biblio>`, for example).

Each class, then, might be given a different display definition in a document-level or externally related style sheet: the abstract indented and in an italic typeface (such as `div.abstract {left-margin: +0.5in; font-style: italic}`); the body in a left-justified roman typeface; the conclusion similar to the abstract; and the bibliography automatically numbered and formatted appropriately.

We provide a detailed description of style sheets, classes, and their applications in Chapter 8.

4.1.1 The Tag

As defined in the HTML 4.01 and XHTML 1.0 standards, the `<div>` tag divides your document into separate, distinct sections. It may be used strictly as an organizational tool, without any sort of formatting associated with it; it becomes more effective if you add the `id` and `class` attributes to label the division. The `<div>` tag may also be combined with the `align` attribute to control the alignment of whole sections of your document's content in the display and with the many programmatic "on" attributes for user interaction.

4.1.1.1 The align attribute

The `align` attribute for `<div>` positions the enclosed content to either the `left` (default), `center`, or `right` of the display. In addition, you can specify `justify` to align both the left and right margins of the text. The `<div>` tag may be nested, and the alignment of the nested `<div>` tag takes precedence over the containing `<div>` tag. Further, other nested alignment tags, such as `<center>`, aligned paragraphs (see `<p>` in Section 4.1.2), or specially aligned table rows and cells, override the effect of `<div>`. Like the `align` attribute for other tags, it is deprecated in the HTML and XHTML standards in deference to style sheet-based layout controls.

Function:

Defines a block of text

Attributes:

ALIGN	ONKEYPRESS
CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
NOWRAP	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE

End tag:

; usually omitted in HTML

Contains:

body_content

Used in:

block

4.1.1.2 The nowrap attribute

Supported only by Internet Explorer, the `nowrap` attribute suppresses automatic word wrapping of the text within the division. Line breaks will only occur where you have placed carriage returns in your source document.

While the `nowrap` attribute probably doesn't make much sense for large sections of text that would otherwise be flowed together on the page, it can make things a bit easier when creating blocks of text with many explicit line breaks: poetry, for example, or addresses. You don't have to insert all those explicit `<br>` tags in a text flow within a `<div nowrap>` tag. On the other hand, all other browsers ignore the `nowrap` attribute and merrily flow your text together anyway. If you are targeting only Internet Explorer with your documents, consider using `nowrap` where needed, but otherwise, we can't recommend this attribute for general use.

4.1.1.3 The dir and lang attributes

The `dir` attribute lets you advise the browser as to which direction the text ought to be displayed, and the `lang` attribute lets you specify the language used within the division. Section 3.6.1.1 / Section 3.6.1.2

4.1.1.4 The id attribute

Use the `id` attribute to label the document division specially for later reference by a hyperlink, style sheet, applet, or other automated process. An acceptable `id` value is any quote-enclosed string that uniquely identifies the division and that later can be used to reference that document section unambiguously. Although we're introducing it within the context of the `<div>` tag, this attribute can be used with almost any tag.

When used as an element label, the value of the `id` attribute can be added to a URL to address the labelled element uniquely within the document. You can label both large portions of content (via a tag like `<div>`) or small snippets of text (using a tag like `<i>` or `<span>`). For example, you might label the abstract of a technical report using `<div id="abstract">`. A URL could jump right to that abstract by referencing `report.html#abstract`. When used in this manner, the value of the `id` attribute must be unique with respect to all other `id` attributes within the document, and all the names defined by any `<a>` tags with the `name` attribute. Section 6.3.3

When used as a style-sheet selector, the value of the `id` attribute is the name of a style rule that can be associated with the current tag. This provides a second set of definable style rules, similar to the various style classes you can create. A tag can use both the `class` and `id` attributes to apply two different rules to a single tag. In this usage, the name associated with the `id` attribute must be unique with respect to all other style IDs within the current document. A more complete description of style classes and IDs can be found in Chapter 8.

4.1.1.5 The title attribute

Use the optional `title` attribute and quote-enclosed string value to associate a descriptive phrase with the division. Like the `id` attribute, the `title` attribute can be used with almost any tag and behaves similarly for all tags.

There is no defined usage for the value of the `title` attribute, and many browsers simply ignore it. Internet Explorer, however, will display the title associated with any element when the mouse pauses over that element. Nifty. Used correctly, the `title` attribute could be used in this manner to provide spot help for the various elements within your document.

4.1.1.6 The class and style attributes

Use the `style` attribute with the `<div>` tag to create an inline style for the content enclosed by the tag. The `class` attribute lets you apply the style of a predefined class of the `<div>` tag to the contents of this division. The value of the `class` attribute is the name of a style defined in some document-level or externally defined style sheet. In addition, class-identified divisions also lend themselves well for computer processing of your documents, such as extraction of all divisions whose class name is "biblio," for example, for the automated assembly of a master bibliography. Section 8.1.1 / Section 8.3

4.1.1.7 Event attributes

The many user-related events that may happen in and around a division, such as when a user clicks or double-clicks the mouse within its display space, are recognized by the browser if it conforms to the current HTML or XHTML standards. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box, or activating some multimedia event. Section 12.3.3

4.1.2 The <p> Tag

The `<p>` tag signals the start of a paragraph. That's not well-known even by some veteran webmasters, because it runs counterintuitive to what we've come to expect from experience. Most word processors we're familiar with use just one special character, typically the return character, to signal the *end* of a paragraph. In HTML and XHTML, each paragraph should start with `<p>` and ends with the corresponding `</p>` tag. And while a sequence of newline characters in a text processor-displayed document creates an empty paragraph for each one, browsers typically ignore all but the first paragraph tag.

In practice, with HTML you can ignore the starting `<p>` tag at the beginning of the first paragraph, and the `</p>` tag at the end of paragraphs: they can be implied from other tags that occur in the document, and hence safely omitted.[1]

[1] XHTML, on the other hand, requires explicit starting and ending tags.

For example:

```
<body>
This is the first paragraph, at the very beginning of the
body of this document.
<p>
The tag above signals the start of this second paragraph.
When rendered by a browser, it will begin slightly below the
end of the first paragraph, with a bit of extra white space
between the two paragraphs.
<p>
This is the last paragraph in the example.
</body>
```

Notice that we haven't included the paragraph start tag (`<p>`) for the first paragraph or any end paragraph tags at all in the HTML example; they can be unambiguously inferred by the browser and are therefore unnecessary.

Function:

Defines a paragraph of text

Attributes:

ALIGN	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	

End tag:

; often omitted in HTML

Contains:

text

Used in:

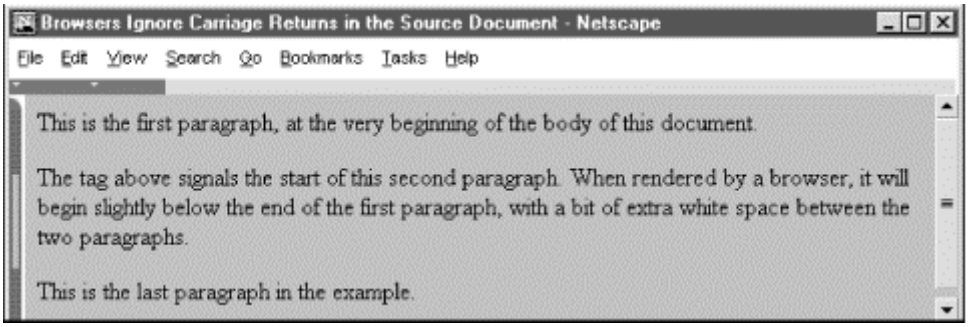
block

In general, you'll find that human document authors tend to omit postulated tags whenever possible while automatic document generators tend to insert them. That may be because the software designers didn't want to run the risk of having their product chided by competitors as not adhering to the HTML standard, even though we're splitting letter-of-the-law hairs here. Go ahead and be defiant: omit that first paragraph's tag and don't give a second thought to paragraph ending tags, provided, of course, that your document's structure and clarity are not compromised. That is, as long as you are aware that XHTML frowns severely on such laxity.

4.1.2.1 Paragraph rendering

When encountering a new paragraph ( ) tag, a browser typically inserts one blank line plus some extra vertical space into the document before starting the new paragraph. The browser then collects all the words and, if present, inline images into the new paragraph, ignoring leading and trailing spaces (not spaces between words, of course) and return characters in the source text. The browser software then flows the resulting sequence of words and images into a paragraph that fits within the margins of its display window, automatically generating line breaks as needed to wrap the text within the window. For example, compare how a browser arranges the text into lines and paragraphs (Figure 4-1) to how the preceding example is printed on the page. The browser may also automatically hyphenate long words, and the paragraph may be full-justified to stretch the line of words out towards both margins.

Figure 4-1. Browsers ignore common return characters in the source HTML document



The net result is that you do not have to worry about line length, word wrap, and line breaks when composing your documents. The browser will take any arbitrary sequence of words and images and display a nicely formatted paragraph.

If you want to control line length and breaks explicitly, consider using a preformatted text block with the `<pre>` tag. If you need to force a line break, use the `<br>` tag. Section 4.7.5 / Section 4.7.1

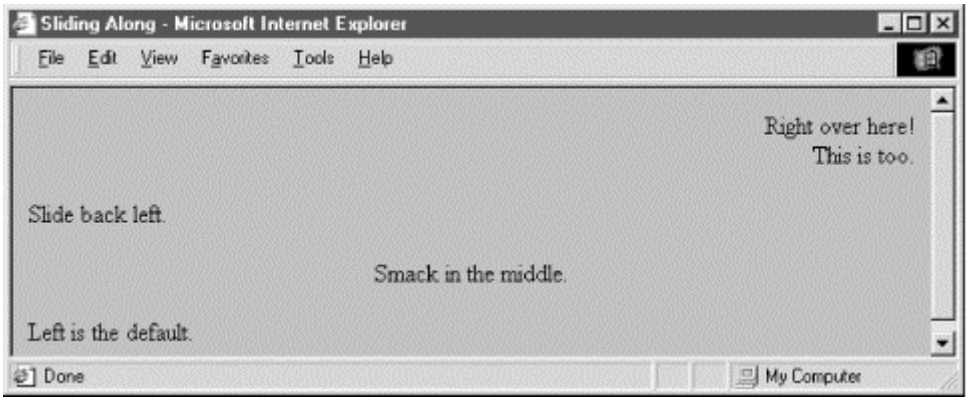
4.1.2.2 The align attribute

Most browsers automatically left-justify a new paragraph. To change this behavior, HTML 4 and XHTML give you the align attribute for the `<p>` tag and provide four kinds of content justification: `left`, `right`, `center`, or `justify`.

Figure 4-2 shows you the effect of various alignments as rendered from the following source:

```
<p align=right>
Right over here!
<br>
This is too.
<p align=left>
Slide back left.
<p align=center>
Smack in the middle.
</p>
Left is the default.
```

Figure 4-2. Effect of the align attribute on paragraph justification



Notice in the HTML example that the paragraph alignment remains in effect until the browser encounters another `<p>` tag or an ending `</p>` tag. We deliberately left out a final `<p>` tag in the example to illustrate the effects of the `</p>` end tag on paragraph justification. Other body elements may also disrupt the current paragraph alignment and cause subsequent paragraphs to revert to the default left alignment, including forms, headers, tables, and most other body content-related tags.

Note that the `align` attribute is deprecated in HTML 4 and XHTML in deference to style sheet-based alignments.

4.1.2.3 The dir and lang attributes

The `dir` lets you advise the browser as to which direction the text within the paragraph ought to be displayed, and the `lang` attribute lets you specify the language used within that paragraph. The `dir` and `lang` attributes are supported by the popular browsers, even though there are no behaviors defined for any specific language. Section 3.6.1.1 / Section 3.6.1.2

4.1.2.4 The class, id, style, and title attributes

Use the `id` attribute to create a label for the paragraph that can later be used to unambiguously reference that paragraph in a hyperlink target, for automated searches, as a style-sheet selector, and with a host of other applications. Section 4.1.1.4

Use the optional `title` attribute and quote-enclosed string value to provide a descriptive phrase for the paragraph. Section 4.1.1.5

Use the `style` attribute with the `<p>` tag to create an inline style for the paragraph's contents. The `class` attribute lets you label the paragraph with a name that refers to a predefined class of the `<p>` tag declared in some document-level or externally defined style sheet. And, class-identified paragraphs lend themselves well for computer processing of your documents, such as extraction of all paragraphs whose class name is "citation," for example, for automated assembly of a master list of citations. Section 8.1.1 / Section 8.3

4.1.2.5 Event attributes

Like with divisions, there are many user-initiated events, such as when a user clicks or double-clicks within its display space, that are recognized by the browser if it conforms to the current HTML or XHTML standards. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.1.2.6 Allowed paragraph content

A paragraph may contain any element allowed in a text flow, including conventional words and punctuation, links (`<a>`), images (`<img>`), line breaks (`<br>`), font changes (`<b>`, `<i>`, `<tt>`, `<u>`, `<strike>`, `<big>`, `<small>`, `<sup>`, `<sub>`, and `<font>`), and content-based style changes (`<acronym>`, `<cite>`, `<code>`, `<dfn>`, `<em>`, `<kbd>`, `<samp>`, `<strong>`, and `<var>`). If any other element occurs within the paragraph, it implies that the paragraph has ended, and the browser assumes that the closing `</p>` tag was not specified.

4.1.2.7 Allowed paragraph usage

You may specify a paragraph only within a *block*, along with other paragraphs, lists, forms, and preformatted text. In general, this means that paragraphs can appear where a flow of text is appropriate, such as in the body of a document, an element in a list, and so on. Technically, paragraphs cannot appear within a header, anchor, or other element whose content is strictly text-only. In practice, most browsers ignore this restriction and format the paragraph as a part of the containing element.

4.2 Headings

Users have a hard enough time reading what's displayed on a screen. A long flow of text, unbroken by title, subtitles, and other headers, crosses the eyes and numbs the mind, not to mention the fact that it makes it nearly impossible to scan the text for a specific topic.

You should always break a flow of text into several smaller sections within one or more headings (like this book!). There are six levels of headings that you can use to structure a text flow into a more readable, more manageable document. And, as we discuss in Chapter 5 and in Chapter 8, there are a variety of graphical and text-style tricks that help divide your document and make its contents more accessible as well as more readable to users.

4.2.1 Heading Tags

The six heading tags, written as `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`, indicate the highest (`<h1>`) to the lowest (`<h6>`) precedence a heading may have in the document.

The enclosed text within a heading typically is uniquely rendered by the browser, depending upon the display technology available to it. The browser may choose to center, embolden, enlarge, italicize, underline, or change the color of headings to make each stand out within the document. And in order to thwart the most tedious writers, users, as well, often can alter how a browser will render the different headings.

Fortunately, in practice most browsers use a diminishing character point size for the sequence of headers, so that `<h1>` text is quite large and `<h6>` text is quite minuscule (see Figure 4-3, for example).

By tradition, authors have come to use `<h1>` headers for document titles, `<h2>` headers for section titles, and so on, often matching the way many of us were taught to outline our work with heads, subheads, and sub-subheads.

Finally, don't forget to include the appropriate heading end tags in your document. The browser won't insert one automatically for you, and omitting the ending tag for a heading can have disastrous consequences for your document.

<h1>, <h2>, <h3>, <h4>, <h5>, <h6>

Function:

Define one of six levels of headers

Attributes:

ALIGN	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	

End tag:

</h1>, </h2>, </h3>, </h4>, </h5>, </h6>; never omitted

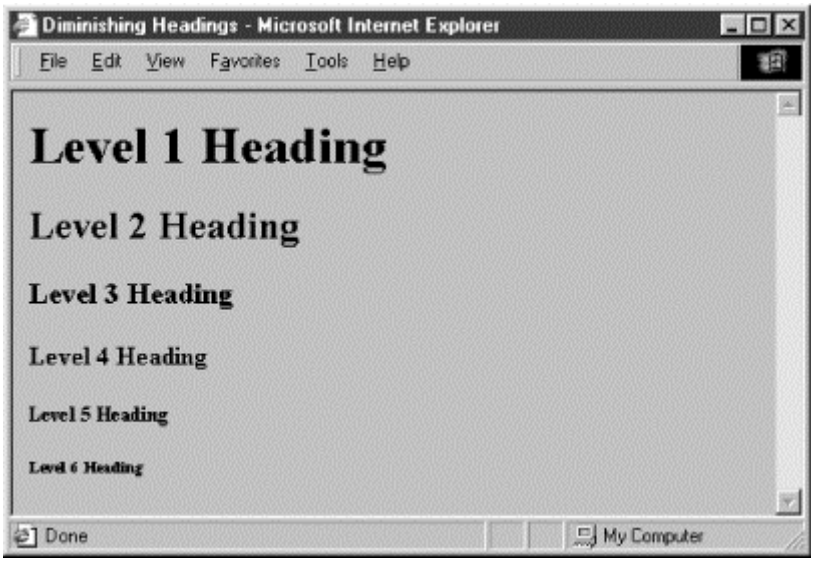
Contains:

text

Used in:

body_content

Figure 4-3. Browsers typically use diminishing text sizes for rendering headings

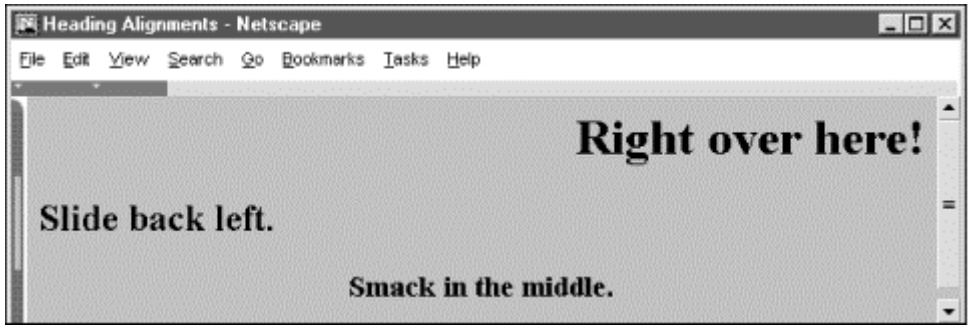


4.2.1.1 The align attribute

The default heading alignment for most browsers is `left`. Like the `<div>` and `<p>` tags, you can alter this alignment with the `align` attribute and one of the values `left`, `center`, `right`, or `justify`. Figure 4-4 shows these alternative alignments as rendered from the following source:

```
<h1 align=right>Right over here!</h1>
<h2 align=left>Slide back left.</h2>
<h3 align=center>Smack in the middle.</h3>
```

Figure 4-4. The headings align attribute in action



The `justify` value for align is not supported yet by any browser, and don't hold your breath. The `align` attribute is deprecated in HTML 4 and XHTML in deference to style sheet-based controls.

4.2.1.2 The dir and lang attributes

The `dir` attribute lets you advise the browser as to which direction the text within that paragraph ought to be displayed, and `lang` lets you specify the language used within the heading. Section 3.6.1.1 / Section 3.6.1.2

4.2.1.3 The class, id, style, and title attributes

Use the `id` attribute to create a label for the heading that can later be to used to unambiguously reference that heading in a hyperlink target, for automated searches, as a style-sheet selector, and with a host of other applications. Section 4.1.1.4

Use the optional `title` attribute and quote-enclosed string value to provide a descriptive phrase for the heading. Section 4.1.1.5

Use the `style` attribute with the heading tags to create an inline style for the headings' contents. The `class` attribute lets you label the heading with a name that refers to a predefined class declared in some document-level or externally defined style sheet. Section 8.1.1 / Section 8.3

4.2.1.4 Event attributes

Each user-initiated event that may happen in and around a heading each are recognized by the browser if it conforms to the HTML or XHTML standards. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.2.2 Appropriate Use of Headings

It's good form to repeat your document's title in the first heading tag, since the title you specify at the beginning of your document doesn't appear in the user's main display window. The title should match the one in the document's `<head>`. The following HTML segment is a good example of repeating the document's title in the header and in the body of the document:

```
<html>
<head>
<title>Kumquat Farming in North America</title>
</head>
<body>
<h3>Kumquat Farming in North America</h3>
<p>
Perhaps one of the most enticing of all fruits is the...
```

While the browser may place the title somewhere in the document window and may also use it to create bookmarks or hotlist entries, all of which vaguely are somewhere on the user's desktop, the level three title heading in the example will always appear at the very beginning of the document. It serves as a visible title to the document regardless of how the browser handles the <title> tag contents. And, unlike the <title> text, the heading title will appear at the beginning of the first page should the user elect to print the document. Section 3.7.2

In the example, we chose to use a level three heading (<h3>) whose rendered font typically is just a bit larger than the regular document text. Levels one and two are larger still and often a bit overbearing. You should choose a level of heading that you find useful and attractive and use that level consistently throughout your documents.

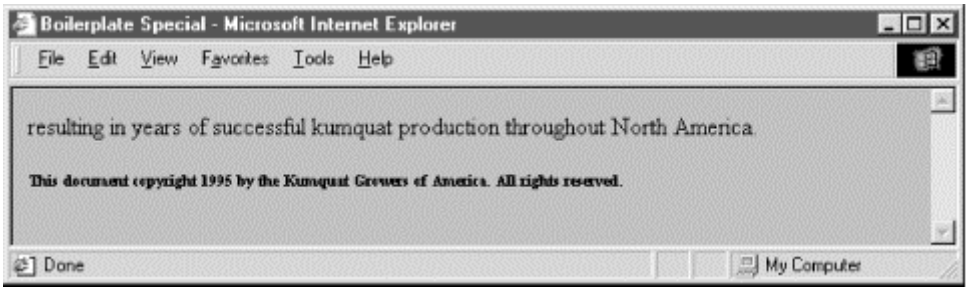
Once you have established the top-level heading for your document, use additional headings at the same or lower level throughout to add structure and "scanability" to the document. If you use a level three heading for the document title, break your document into several sections using level four headings. If you have the urge to subdivide your text further, consider using a level two heading for the title, level three for the section dividers, and level four for the subsections.

4.2.3 Using Headings for Smaller Text

For most graphical browsers, the fonts used to display <h1>, <h2>, and <h3> headers are larger, <h4> is the same, and <h5> and <h6> are smaller than the regular text size. Authors typically use the latter two sizes for boilerplate text, like a disclaimer or a copyright notice. It's become quite popular to use the smaller text in Tables of Contents or home pages that display a site's contents. Experiment with <h5> and <h6> to get the effect you want. See how a typical browser renders the copyright reference in the following sample HTML segment (see Figure 4-5):

```
resulting in years of successful kumquat production
throughout North America.
</p>
<h6>This document copyright 1995 by the Kumquat Growers of
America. All rights reserved. </h6>
</body>
</html>
```

Figure 4-5. HTML authors typically use heading level six for boilerplate text



4.2.4 Allowed Heading Content

A heading may contain any element allowed in *text*, including conventional text, link anchors (<a>), images (<img>), line breaks (
), font embellishments (<b>, <i>, <tt>, <u>, <strike>, <big>, <small>, <sup>, <sub>, and <font>), and content-based style changes (<acronym>, <cite>, <code>, <dfn>, , <kbd>, <samp>, , and <var>). In practice, however, font or style changes may not take effect within a heading, since the heading itself prescribes a font change within the browser.

There is widespread abuse of the heading tags as a mechanism for changing the font of an entire document. Technically, paragraphs, lists, and other block elements are not allowed within a heading and may be mistaken by the browser to indicate the implied end of the heading. In practice, most browsers apply the style of the heading to all contained paragraphs. We discourage this practice since it is not only a violation of HTML and XHTML standards but usually ugly to look at. Imagine if your local paper printed all the copy in headline type!

Designating large sections of text as heading content defeats the purpose of the tag. If you really want to change the entire font or type size of your document, consider instead defining a unique style for the <body> tag of your document. This style will be applied to all the content within the <body> and will make later modification of your document style much easier. See Chapter 8 for details.

And we strongly recommend that you carefully test your pages with more than one browser and at several different resolutions. As to be expected, your <h6> text may be readable at 320 x 480 resolution, but disappear on a 600 x 800 display.

4.2.5 Allowed Heading Usage

Formally, the HTML and XHTML standards allow headings only within *body content*. In practice, most browsers recognize headings almost anywhere, formatting the rendered text to fit within the current element. In all cases, the occurrence of a heading signifies the end of any preceding paragraph or other text element, so you can't use the heading tags to change font sizes in the same line. Rather, use styles to achieve those acute display effects. Section 8.1.1

4.2.6 Adding Images to Headings

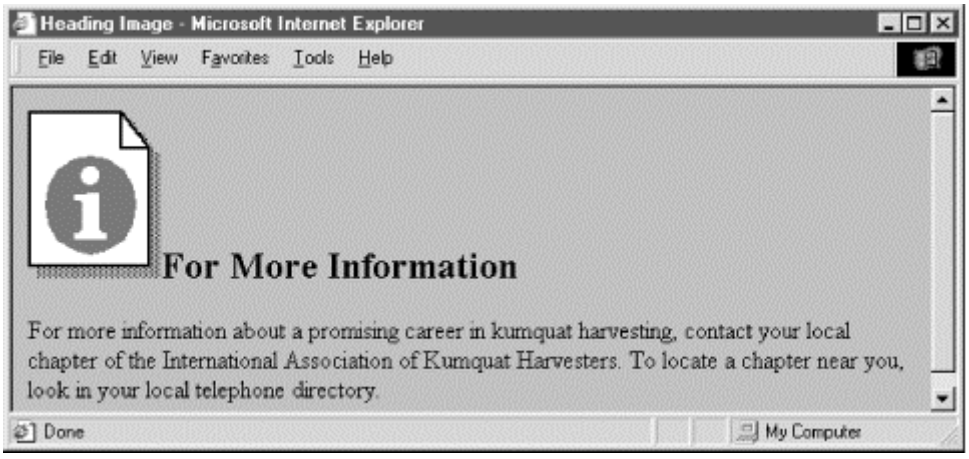
It is possible to insert one or more images within your headings, from small bullets or icons to full-sized logos. Combining a consistent set of headings with corresponding icons across a family of documents is not only visually attractive, but an effective way of aiding users' perusal of your document collection. Section 5.2.6

Adding an image to a heading is easy. For example, the following text puts an "information" icon inside the "For More Information" heading, as you can see in Figure 4-6:

```
<h2>
<img src="info.gif">
For More Information</h2>
```

In general, images within headings look best at the beginning of the heading, aligned with the bottom or middle of the heading text.

Figure 4-6. An image within a heading



4.3 Changing Text Appearance

A number of tags change the appearance of and associate hidden meaning with text. In general, these tags can be grouped into two flavors: content-based styles and physical styles.

In addition, the W3C standard for Cascading Style Sheets is now well-supported by the popular browsers, providing another, more comprehensive way for authors to control the look and layout of their document text. Netscape has also implemented style sheets through JavaScript. We describe the tag-based text styles in this chapter. See Chapter 8 for details about Cascading Style Sheets, and Chapter 12 for JavaScript-based style sheets.

4.3.1 Content-Based Styles

Content-based style tags inform the browser that the enclosed text has a specific meaning, context, or usage. The browser then formats the text in a manner consistent with that meaning, context, or usage.

Because font style is specified via semantic clues, the browser can choose a display style that is appropriate for the user. Since such styles vary by locale, using content-based styles helps ensure that your documents will have meaning to a broader range of readers. This is particularly important when a browser is targeted at blind or handicapped readers whose display options are radically different from conventional text or are extremely limited in some way.

The current HTML and XHTML standards do not define a format for each of the content-based styles except that they must be rendered in a manner different from the regular text in a document. The standard doesn't even insist that the content-based styles be rendered differently from one another. In practice, you'll find that many of these tags have fairly obvious relationships with conventional print, having similar meanings and rendered styles, and are rendered in the same style and fonts by most browsers.

4.3.2 Physical Styles

We use the word "intent" a lot when we talk about content-based style tags. That's because the meaning conveyed by the tag is more important than the way a browser displays the text. In some cases, however, you might want the text to appear explicitly in some special way - italic or bold, for example - perhaps for legal or copyright reasons. In those cases, use a physical style for the text.

While the tendency with other text-processing systems is to control style and appearance explicitly, with HTML or XHTML you should avoid explicit, physical tags except on rare occasions. Provide the browser with as much contextual information as possible. Use the content-based styles. Even though current browsers may do nothing more than display their text in italic or bold, future browsers and various document-generation tools may use the content-based styles in any number of creative ways.

4.4 Content-Based Style Tags

It takes discipline to use the content-based styles, since it is easier to simply think of how your text should look, not necessarily what it may also mean. Once you get started using content-based styles, your documents will be more consistent and better lend themselves to automated searching and content compilation.

Content-Based Style Tags

Function:

Alter the appearance of text based upon the meaning, context, or usage of the text

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tags:

Never omitted

Contains:

text

Used in:

text

4.4.1 The <abbr> Tag

The `<abbr>` tag indicates that the enclosed text is an abbreviated form of a longer word or phrase. The browser might use this information to change the way it renders the enclosed text. Since none of the popular browsers yet support this tag first introduced in HTML 4.0, we can't predict how they might implement it.

4.4.2 The <acronym> Tag

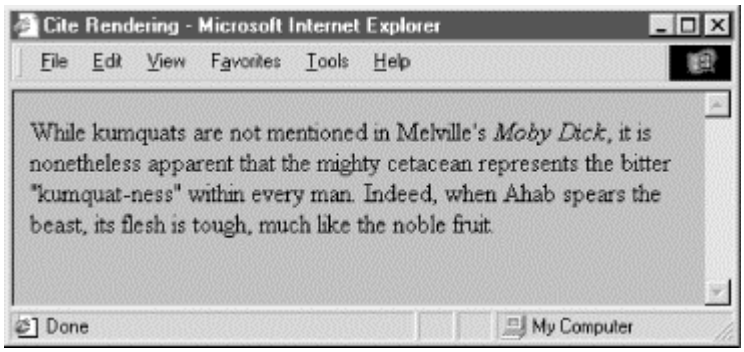
The <acronym> tag indicates that the enclosed text is an acronym, an abbreviation formed from the first letter of each word in a name or phrase, such as HTML and IBM. The popular browsers don't yet support this addition to content-based style tags, so we can't show you how they might render text tagged <acronym>.

4.4.3 The <cite> Tag

The <cite> tag usually indicates that the enclosed text is a bibliographic citation like a book or magazine title. By convention, the citation text is rendered in italic. See Figure 4-7 for how Internet Explorer renders this source text:

```
While kumquats are not mentioned in Melville's
<cite>Moby Dick</cite>, it is nonetheless apparent
that the mighty cetacean represents the bitter
"kumquat-ness" within every man. Indeed, when Ahab
spears the beast, its flesh is tough, much like the noble fruit.
```

Figure 4-7. Internet Explorer renders <cite> in italic



Use the <cite> tag to set apart any reference to another document, especially those in the traditional media, such as books, magazines, journal articles, and the like. If an online version of the referenced work exists, you also should enclose the citation within the <a> tag and make it a hyperlink to that online version.

The <cite> tag also has a hidden feature: it enables you or someone else to automatically extract a bibliography from your documents. It is easy to envision a browser that compiles tables of citations automatically, displaying them as footnotes or as a separate document entirely. The semantics of the <cite> tag go far beyond changing the appearance of the enclosed text; they enable the browser to present the content to the user in a variety of useful ways.

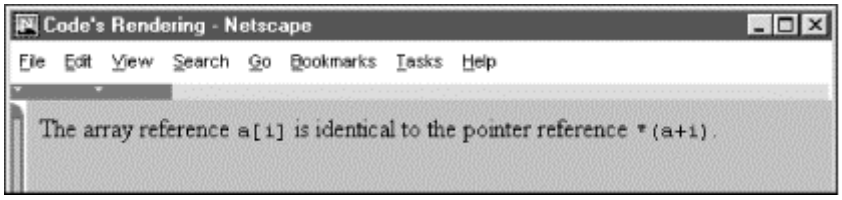
4.4.4 The <code> Tag

Software code warriors have become accustomed to a special style of text presentation for their source programs. The <code> tag is for them. It renders the enclosed text in a monospaced, teletype-style font like Courier, familiar to most programmers and readers of O'Reilly's series of books, including this one.

This following bit of en<code>ed text is rendered in monospaced font style by Netscape as shown in Figure 4-8:

```
The array reference <code>a[i]</code> is identical to
the pointer reference <code>*(a+i)</code>.
```

Figure 4-8. Use <code> to present computer-speak



You should use the <code> tag only for text that represents computer source code or other machine-readable content. While the <code> tag usually just makes text appear in a monospaced font, the implication is that it is source code and future browsers may add other display effects. For example, a programmer's browser might look for <code> segments and perform some additional text formatting like special indentation of loops and conditional clauses. If the only effect you desire is a monospaced font, use the <tt> tag instead.

4.4.5 The <dfn> Tag

Use <dfn> to tag those defining instances of special terms or phrases. It may not result in any formatting changes by the browser. Instead, <dfn> might assist in creating a document index or glossary.

For example, use the <dfn> tag to introduce a new phrase to the reader:

```
When analyzing annual crop yields, <dfn>rind spectroscopy</dfn> may prove useful.
By comparing the relative levels of saturated hydrocarbons in fruit from adjacent
trees, rind spectroscopy has been shown to be 87% effective in predicting an
outbreak of trunk dropsy in trees under four years old.
```

Notice that we delimit only the first occurrence of "rind spectroscopy" with a <dfn> tag in the example. Good style tells us not to clutter the text with highlighted text. As with the many other content-related and physical style tags, the fewer the better.[2] As a general style, especially in technical documentation, set off new terms when they are first introduced to help your readers better understand the topic at hand, but resist tagging the terms thereafter.

> [2] If you need convincing that less is better when applying the content-based and physical style tags, try reading a college textbook in which someone has highlighted what they considered important words and phrases with a yellow pen.

4.4.6 The Tag

The tag tells the client browser to present the enclosed text with emphasis. For nearly all browsers, this means the text is rendered in italic. For example, the popular browsers will emphasize by italicizing the words "always" and "never" in the following HTML source:

```
Kumquat growers must <em>always</em> refer to kumquats
as "the noble fruit," <em>never</em> as just a "fruit."
```

Adding emphasis to your text is a tricky business. Too little, and the emphatic phrases may be lost. Too much, and you lose the urgency. Like any seasoning, emphasis is best used sparingly.

Although invariably displayed in italic, the tag has broader implications as well and someday browsers may render emphasized text with a different special effect. The <i> tag explicitly italicizes text; use it if all you want is italic. Besides emphasis, also consider using when presenting new terms or as a fixed style when referring to a specific type of term or concept. For instance, one of O'Reilly's book styles is to specially format file and device names. might be used to differentiate those terms from simple italic for emphasis.

4.4.7 The <kbd> Tag

Speaking of special style for technical concepts, there is the <kbd> tag. As you probably already suspect, it is used to indicate text that is typed on a keyboard. Its enclosed text typically is rendered by the browser in monospaced font style.

The <kbd> tag is most often used in computer-related documentation and manuals, such as in this example:

```
Type <kbd>quit</kbd> to exit the utility, or type
<kbd>menu</kbd> to return to the main menu.
```

4.4.8 The <samp> Tag

The <samp> tag indicates a sequence of literal characters that should have no other interpretation by the user. This tag is most often used when a sequence of characters is taken out of its normal context. For example, the following source:

```
The <samp>ae</samp> character sequence may be converted
to the &aelig; ligature if desired.
```

is rendered by Netscape as shown in Figure 4-9.

Figure 4-9. Setting off sample text using the <samp> tag

The special HTML reference for the "ae" ligature entity is `æ` and is converted to its appropriate æ ligature character by most browsers. For more information, see Appendix F.

The `<samp>` tag is not used very often. It should be used in those few cases where special emphasis needs to be placed on small character sequences taken out of their normal context.

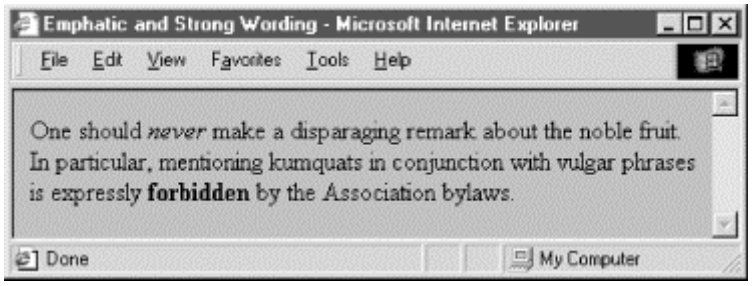
4.4.9 The Tag

Like the `<em>` tag, the `<strong>` tag is for emphasizing text, except with more gusto. Browsers typically display the `<strong>` tag differently than the `<em>` tag, usually by making the text bold (versus italic), so that users can distinguish between the two. For example, in the following text, the emphasized "never" appears in italic with Internet Explorer, while the `<strong>` "forbidden" is rendered in bold characters (see Figure 4-10):

```
One should <em>never</em> make a disparaging remark
about the noble fruit. In particular, mentioning
kumquats in conjunction with vulgar phrases is
expressly <strong>forbidden</strong> by the Association
bylaws.
```

If common sense tells us that the `<em>` tag should be used sparingly, the `<strong>` tag should appear in documents even more infrequently. `<em>` text is like shouting. `<strong>` text is nothing short of a scream. Like a well-chosen epithet voiced by an otherwise taciturn person, restraint in the use of `<strong>` makes its use that much more noticeable and effective.

Figure 4-10. Strong and emphasized text are rendered differently by Internet Explorer

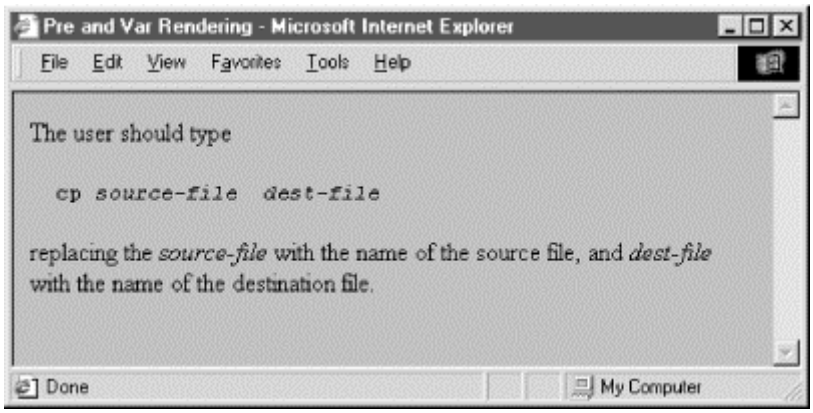


4.4.10 The <var> Tag

The `<var>` tag, another computer-documentation trick, indicates a variable name or a user-supplied value. The tag is often used in conjunction with the `<code>` and `<pre>` tags for displaying particular elements of computer programming code samples and the like. `<var>`-tagged text typically is rendered in italics, as shown in Figure 4-11, which displays Internet Explorer's rendering of the following example:

```
The user should type
<pre>
  cp <var>source-file</var>  <var>dest-file</var>
</pre>
replacing the <var>source-file</var> with the name of
the source file, and <var>dest-file</var> with the name
of the destination file.
```

Figure 4-11. The <var> tag typically appears in preformatted (<pre>) computer code



Like the other computer programming and documentation-related tags, the <var> tag not only makes it easy for users to understand and browse your documentation, but automated systems might someday use the appropriately tagged text to extract information and useful parameters mentioned in your document. Once again, the more semantic information you provide to your browser, the better it can present that information to the user.

4.4.11 The class, id, style, and title Attributes

Although each content-based tag has a defined style, you can override the style by defining a look for each tag. This new look can be applied to the content-based tags using either the style or class attributes. Section 8.1.1 / Section 8.3

You also may assign a unique id-entifier to the content-based style tag, as well as a less-rigorous title, using the respective attributes and their accompanying quote-enclosed string values. Section 4.1.1.4 / Section 4.1.1.5

4.4.12 The dir and lang Attributes

The dir attribute advises the browser as to which direction the text within the content-based style tag ought to be displayed, and lang lets you specify the language used within the tag. Section 3.6.1.1 / Section 3.6.1.2

4.4.13 Event Attributes

Things happen in and around a content-based tag's content, and, with the respective "on" attribute and value, you may react to that event by displaying a user dialog or activating some multimedia event. Section 12.3.3

4.4.14 Summary of Content-Based Tags

The various graphical browsers render text inside content-based tags in similar fashion; text-only browsers like Lynx have consistent styles for the tags. Table 4-1 summarizes these browsers' display styles for the native tags. However, style sheet definitions may override these native display styles.

Table 4-1, Content-Based Tags

Tag	Netscape	Internet Explorer	Lynx
<abbr>	n/a	n/a	n/a
<acronym>	n/a	n/a	n/a
<cite>	*italic*	*italic*	monospace
<code>	monospace	monospace	monospace
<dfn>	n/a	*italic*	n/a
	italic	*italic*	monospace
<kbd>	monospace	**monospace bold**	monospace
<samp>	monospace	monospace	monospace
	bold	**bold**	monospace
<var>	*italic*	*italic*	monospace

4.4.15 Allowed Content

Any content-based style tag may contain any item allowed in *text*, including conventional text, anchors, images, and line breaks. In addition, other content-based and physical style tags can be embedded within the content.

4.4.16 Allowed Usage

Any content-based style tag may be used anywhere an item allowed in *text* is used. In practice, this means you can use the , <code>, and other similar tags anywhere in your document except inside <title>, <listing>, or <xmp> tagged segments. You can use text style tags in headings, too, but their effect may be overridden by the effects of the heading tag itself.

4.4.17 Combining Content-Based Styles

It may have occurred to you to combine two or more of the various content-based styles to create interesting and perhaps even useful hybrids. Thus, an emphatic citation might be achieved with:

```
<cite><em>Moby Dick</em></cite>
```

In practice, Dr. Frankenstein, the browser usually ignores the monster; as you can test by typing and viewing the example yourself, Moby Dick gets the citation without emphasis.

The HTML and XHTML standards do not require the browser to support every possible combination of content-based styles and does not define how the browser should handle such combinations. Someday, maybe. For now, it's best to choose one tag and be satisfied.

4.5 Physical Style Tags

There are nine physical styles provided by the current HTML and XHTML standards, including bold, italic, monospaced, underlined, strike-through, larger, smaller, superscripted, and subscripted text. In addition, to our dismay, Netscape still supports "blinking" text.[3] All physical style tags require an ending tag.

4.5.1 The Tag

The tag is the physical equivalent of the content-based style tag, but without the latter's extended meaning. The tag explicitly boldfaces a character or segment of text that is enclosed between it and its corresponding () end tag. If a boldface font is not available, the browser may use some other representation, such as reverse video or underlining.

4.5.2 The <big> Tag

The <big> tag makes it easy to increase the size of text. It couldn't be simpler: the browser renders the text between the <big> tag and its matching </big> ending tag one font size larger than the surrounding text. If that text is already at the largest size, <big> has no effect. Section 4.6.3

Even better, you can nest <big> tags to enlarge the text. Each <big> tag makes the text one size larger, up to a limit of size seven, as defined by the font model.

Be careful with your use of the <big> tag, though. Because browsers are quite forgiving and try hard to understand a tag, those that don't support <big> often interpret it to mean bold.

4.5.3 The <blink> Tag

Text contained between the <blink> tag and its end tag </blink> does just that: blink on and off. Netscape for Macintosh, for example, simply and reiteratively reverses the background and foreground colors for the <blink> enclosed text. Neither the HTML nor the XHTML standard include <blink>; it is supported as an extension only by Netscape Navigator.

We cannot effectively reproduce the animated effect in these static pages, but it is easy to imagine and best left to the imagination, too. That's because blinking text has two primary effects: it gets your reader's attention, and then promptly annoys them to no end. Blinking text should be used sparingly in any context.

[3] Once programmed, always a feature, we guess. Internet Explorer has its warts, too.

As we discuss each physical tag in detail, keep in mind that they convey an acute styling for the immediate text. For more comprehensive, document-wide control of text display, use style sheets (see Chapter 8).

Physical Style Tags

Function:

Specify a physical style for text

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tags:

Never omitted

Contains:

text

Used in:

text

4.5.4 The <i> Tag

The `<i>` tag is like the `<em>` content-based style tag. It and its necessary (`</i>`) end tag tell the browser to render the enclosed text in an italic or oblique typeface. If the typeface is not available to the browser, highlighting, reverse video, or underlining might be used.

4.5.5 The <s> Tag (Deprecated)

The `<s>` tag is an abbreviated form of the `<strike>` tag supported by both Internet Explorer and Netscape. It is now a deprecated tag in HTML 4 and XHTML, meaning don't use it; it will eventually go away.

4.5.6 The <small> Tag

The `<small>` tag works just like its `<big>` counterpart (see previous description), except it decreases the size of text instead of increasing it. If the enclosed text is already at the smallest size supported by the font model, `<small>` has no effect.

Like `<big>`, you may also nest `<small>` tags to sequentially shrink text. Each `<small>` tag makes the text one size smaller than the containing `<small>` tag, down to a limit of size one.

4.5.7 The <strike> Tag (Deprecated)

Most browsers will put a line through ("strike through") text that appears inside the <strike> tag and its </strike> end tag. Presumably, it is an editing markup that tells the reader to ignore the text passage, reminiscent of the days before typewriter correction tape. You'll rarely, if ever, see the tag in use today, and probably never will: it is deprecated in HTML 4 and XHTML, just one version away from complete elimination from the standard.

4.5.8 The <sub> Tag

The text contained between the _{tag and its} end tag gets displayed half a character's height lower, but in the same font and size as the current text flow. Both <sub> and its <sup> counterpart are useful for math equations and in scientific notation, as well as with chemical formulæ.

4.5.9 The <sup> Tag

The ^{tag and its} end tag superscripts the enclosed text; it gets displayed half a character's height higher, but in the same font and size as the current text flow. This tag is useful for adding footnotes to your documents, along with exponential values in equations. In combination with the <a> tag, you can create nice, hyperlinked footnotes:

```
The larval quat
weevil<a href="footnotes.html#note74"><sup><small>74</small></sup></a> is a
```

This example assumes that *footnotes.html* contains all your footnotes, appropriately delimited as named document fragments.

4.5.10 The <tt> Tag

In a manner like the <code> and <kbd> tags, the <tt> tag and necessary </tt> end tag direct the browser to display the enclosed text in a monospaced typeface. For those browsers that already use a monospaced typeface, this tag may make no discernible change in the presentation of the text.

4.5.11 The <u> Tag (Deprecated)

This tag tells the browser to underline the text contained between the <u> and the corresponding </u> tag. The underlining technique is simplistic, drawing the line under spaces and punctuation as well as the text. This tag is deprecated in HTML 4 and XHTML and will be eliminated in the next version of the standard. The same effect can be achieved by using style sheets, covered in Chapter 8.

4.5.12 The dir and lang Attributes

The dir attribute lets you advise the browser as to which direction the text within the physical tag ought to be displayed, and lang lets you specify the language used within the tag. Section 3.6.1.1 / Section 3.6.1.2

4.5.13 The class, id, style, and title Attributes

Although each physical tag has a defined style, you can override that style by defining your own look for each tag. This new look can be applied to the physical tags using either the style or class attributes. Section 8.1.1 / Section 8.3

You also may assign a unique id to the physical style tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.5.14 Event Attributes

Like with content-based style tags, user-initiated mouse and keyboard events can happen in and around a physical-style tag's contents. Many of these events are recognized by the browser if it conforms to current standards, and, with the respective "on" attribute and value, you may react to the event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.5.15 Summary of Physical Style Tags

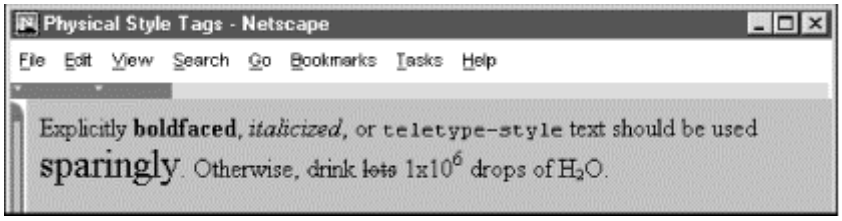
The various graphical browsers render text inside the physical style tags in similar fashion. Table 4-2 summarizes these browser's display styles for the native tags. Style sheet definitions may override these native display styles.

Table 4-2, Physical Style Tags		
Tag	**Meaning**	**Display Style**
<b>	Bold contents	**bold**
<big>	Increased font size	bigger text
<blink>	Alternating fore- and background colors	blinking text
<i>	Italic contents	*italic*
<small>	Decreased font size	smaller text
<s>, <strike>	Strike-through text	~~strike~~
<sub>	Subscripted text	subscript
<sup>	Superscripted text	superscript
<tt>	Teletypewriter style	monospaced
<u>	Underlined contents	underlined

The following HTML source example illustrates some of the various physical tags as rendered by Netscape for Figure 4-12:

```
Explicitly <b>boldfaced</b>, <i>italicized</i>, or
<tt>teletype-style</tt> text should be used
<big><big>sparingly</big></big>.
Otherwise, drink <strike>lots</strike> 1x10<sup>6</sup>
drops of H<sub><small><small>2</small></small></sub>O.
```

Figure 4-12. Use physical text tags with caution



4.5.16 Allowed Content

Any physical style tag may contain any item allowed in *text*, including conventional text, anchors, images, and line breaks. You also can combine physical style tags with other content-based ones.

4.5.17 Allowed Usage

Any physical style tag may be used anywhere an item allowed in *text* can be used. In general, this means anywhere within a document except in the <title>, <listing>, and <xmp> tags. You could use a physical style tag in a heading, but the browser will probably override and ignore its effect in lieu of the heading tag.

4.5.18 Combining Physical Styles

You probably will have better luck, Dr. Frankenstein, combining physical tags than you might have combining content-based tags to achieve multiple effects. For instance, Netscape renders the following in bold and italic typeface:

```
<b><i>Thar she blows!</i></b>
```

In practice, other browsers may elect to ignore such nesting. The HTML 4 standard does require the browser to "do its best" to support every possible combination of styles, but does not define how the browser should handle such combinations. Although most browsers make a good attempt at doing so, do not assume that all combinations will be available to you.

4.6 HTML's Expanded Font Handling

We agonized over including this section in a prominent position within this chapter, or relegating it to the end. It belongs here because the various tags associated with the extended font model for HTML were part of the 3.2 standard. And they remain very popular with HTML authors, besides being well-supported by all the popular browsers. Yet they have been deprecated in the HTML 4 and XHTML 1.0 standards, warranting banishing the whole section to the end of the chapter with all the implicit red flags.

We suspect the W3C wants authors to use style sheets, not acute tags, for explicit control of font styles, colors, and sizes of the text characters. That's why these extended font tags and related attributes have fallen into disfavor. We put this section here because we doubt that the majority of HTML authors will stop using, nor that the popular browsers will any time soon abandon support for, tags that are in such widespread use. Just be aware of their precarious position in the language.

4.6.1 The Extended Font Size Model

Instead of absolute point values, the Extended Font Model of HTML 3.2 as supported by the popular browsers uses a relative model for sizing fonts. Sizes range from 1, the smallest, to 7, the largest; the default (*basefont*) font size is 3.

It is almost impossible to state reliably the actual font sizes used for the various virtual sizes. Most browsers let the user change the physical font size, and the default sizes vary from browser to browser. It may be helpful to know, however, that each virtual size is successively 20 percent larger or smaller than the default font size 3. Thus, font size 4 is 20 percent larger, font size 5 is 40 percent larger, and so on, while font size 2 is 20 percent smaller and font size 1 is 40 percent smaller than font size 3.

4.6.2 The <basefont> Tag (Deprecated)

The `<basefont>` tag lets you define the basic size for the font that the browser will use to render normal document text. We can't recommend that you use it since it has been deprecated in the HTML 4 and XHTML standards.

<basefont>

Function:

Define base font size for relative font size changes

Attributes:

COLOR	NAME
FACE	SIZE
ID	

End tag:

</basefont>; often omitted in HTML

Contains:

Nothing

Used in:

block, head_content

The <basefont> tag has a single attribute recognized by all current browsers, size, whose value determines the document's base font size. It may be specified as an absolute value from 1 to 7, or as a relative value by placing a plus or minus sign before the value. In the latter case, the base font size is increased or decreased by that relative amount. The default base font size is 3.

Internet Explorer supports two additional attributes for the <basefont> tag: color and name. HTML 4 also defines the face attribute as a synonym for the name attribute. These attributes control the color and typeface used for the text in a document and are used just like the analogous color and face attributes for the <font> tag, described later.

HTML 4 also defines the id attribute for the <basefont> tag, allowing you to label the tag uniquely for later access to its contents. Section 4.1.1.4

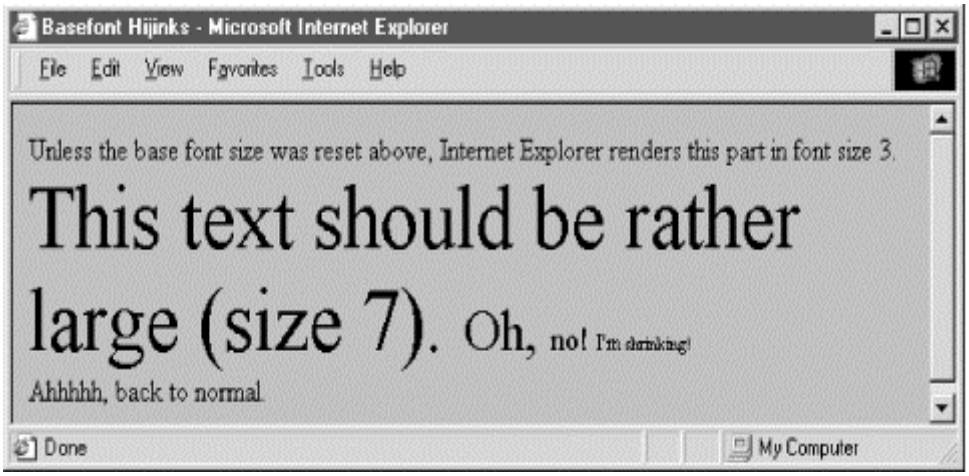
Authors typically include the <basefont> tag in the head of an HTML document, if at all, to set the base font size for the entire document. Nonetheless, the tag may appear nearly anywhere in the document, and it may appear several times throughout the document, each with a new size attribute. With each occurrence, the <basefont> tag's effects are immediate and hold for all subsequent text.

In an egregious deviation from the HTML and SGML standards, the browsers interpret the ending </basefont> tag *not* to terminate the effects of the most recent <basefont> tag. Instead, the </basefont> end tag resets the base font size to the default value of 3, which is the same as writing <basefont size=3>.

The following example source and Figure 4-13 illustrate how Internet Explorer responds to the <basefont> tag and </basefont> end tag:

```
Unless the base font size was reset above,
Inernet Explorer renders this part in font size 3.
<basefont size=7>
This text should be rather large (size 7).
<basefont size=6> Oh,
<basefont size=4> no!
<basefont size=2> I'm
<basefont size=1> shrinking!
</basefont><br>
Ahhhh, back to normal.
```

Figure 4-13. Playing with <basefont>



We recommend against using </basefont>; use <basefont size=3> instead.

4.6.3 The Tag (Deprecated)

The tag lets you change the size, style, and color of text. We don't recommend that you use it because it has been deprecated in the HTML 4 and XHTML standards. But should you decide to ignore our advice, then use it like any other physical or content-based style tag for changing the appearance of a short segment of text.

To control the color of text for the entire document, see the attributes for the <body> tag described in Section 5.3.1.

Function:

Set the font size for text

Attributes:

CLASS	LANG
COLOR	SIZE
DIR	STYLE
FACE	TITLE
ID	

End tag:

; always used

Contains:

text

Used in:

text

4.6.3.1 The size attribute

The value of the `size` attribute must be one of the virtual font sizes (1-7) described earlier, defined as an absolute size for the enclosed text or preceded by a plus or minus sign (+ or -) to define a relative font size that the browser adds to or subtracts from the base font size (see the `<basefont>` tag, section 4.6.2). The browsers automatically round the size to 1 or 7 if the calculated value exceeds either boundary.

In general, use absolute size values when you want the rendered text to be an extreme size, either very large or very small, or when you want an entire paragraph of text to be a specific size.

For example, using the largest font for the first character of a paragraph makes for a crude form of illuminated manuscript (see Figure 4-14):

```
<p>
<font size=7>C</font>all me Ishmael.
```

Figure 4-14. Exaggerating the first character of a sentence with the size attribute for <font>

Also, use an absolute font when inserting a delightfully unreadable bit of "fine" print - boilerplate or legalese - at the bottom of your documents (see Figure 4-15):

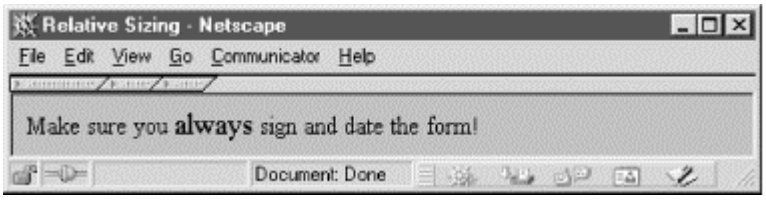
```
<p>
<font size=1>
All rights reserved. Unauthorized redistribution of this document is
prohibited. Opinions expressed herein are those of the authors, not the
Internet Service Provider.
```

Figure 4-15. Use the tiniest font for boilerplate text

Except for the extremes, use relative font sizes to render text in a size different than the surrounding text, to emphasize a word or phrase, for example (see Figure 4-16):

```
<p>
Make sure you <font size=+1>always</font> sign and date the form!
```

Figure 4-16. Use relative sizes for most text embellishments



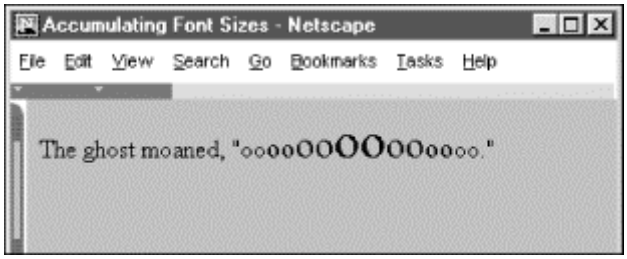
If your relative size change results in a size greater than 7, the browser uses font 7. Similarly, font sizes less than one are rendered with font 1.

Notice that specifying `size=+1` or `size=-1` is identical in effect to the `<big>` and `<small>` respectively. However, nested relative changes to the font size are not cumulative as they are for the alternative tags. Each `<font>` tag is relative to the base font size, not the current font size. For example (see Figure 4-17):

```
<p>
The ghost moaned, "oo<font size=+1>oo<font size=+2>oo<font
size=+3>oo</font>oo</font>oo</font>oo."
```

Contrast this with the `<big>` tag, which increases the size one level as you nest the `tags`. Section 4.5.2

Figure 4-17. Relative font sizes accumulate



4.6.3.2 The color attribute

The `color` attribute for the `<font>` tag sets the color of the enclosed text. The value of the attribute may be expressed in either of two ways: as the red, green, and blue (RGB) components of the desired color or as a standard color name. Enclosing quotes are recommended, but not required.

The RGB color value, denoted by a preceding pound sign, is a six-digit hexadecimal number. The first two digits are the red component, from 00 (no red) to FF (bright red). Similarly, the next two digits are the green component and the last two digits are the blue component. Black is the absence of color, `#000000`; white is all colors, `#FFFFFF`.

For example, to create basic yellow text, you might use:

```
Here comes the <font color="#FFFF00">sun</font>!
```

Alternatively, you may set the enclosed font color using any one of the many standard color names. See Appendix G, for a list of common ones. For instance, you could also have made the previous sample text yellow with the following source:

```
Here comes the <font color=yellow>sun</font>!
```

4.6.3.3 The face attribute

Internet Explorer and Netscape Navigator also let you change the font style in a text passage with the `face` attribute for the `<font>` tag.[4] The quote-enclosed value of `face` is one or more display font names separated with commas.

> [4] For the HTML purist, the once-powerful user who had ultimate control over their browser, this is egregious indeed. Form over function; look over content - what next? Embedded video commercials you can't stop?

The font face displayed by the browser depends on which fonts are available on the individual user's system. The browser parses the list of font names, one after the other, until it matches one with a font name supported by the user's system. If none match, the text display defaults to the font style set by the user in their browser's preferences. For example:

```
This text is in the default font. But,
<font face="Braggadocio, Machine, Zapf Dingbats">
heaven only knows</font>
what font face is this one?
```

If the Internet Explorer user has the Braggadocio, Machine, or none of the listed font typefaces installed in their system, they will be able to read the "heaven only knows" message in the respective or default font style. Otherwise, the message will be garbled because the Zapf Dingbats font contains symbols, not letters. Of course, the alternative is true, too; you may intend that the message be a symbol-encoded secret.

4.6.3.4 The dir and lang attributes

The `dir` attribute lets you advise the browser as to which direction the text within the tag ought to be displayed and `lang` lets you specify the language used for the tag's contents. Section 3.6.1.1 / Section 3.6.1.2

4.6.3.5 The class, id, style, and title attributes

You can associate additional display rules for the `<font>` tag using style sheets. The rules can be applied to the `<font>` tag using either the `style` or `class` attributes. Section 8.1.1 / Section 8.3

You also may assign a unique id to the `<font>` tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.7 Precise Spacing and Layout

Cascading Style Sheets notwithstanding, the original concept of HTML is for specifying document content without indicating format; to delineate the structure and semantics of a document, not how that document is to be presented to the user. Normally, you should leave word wrapping, character and line spacing, and other presentation details up to the browser. That way, the document's content - its rich information, not good looks - is what matters. When looks matter more, such as for commercial presentations, look to style sheets for layout control (see Chapter 8).

4.7.1 The
 Tag

The `<br>` tag interrupts the normal line filling and word wrapping of paragraphs within an HTML or XHTML document. It has no ending tag with HTML,[5] but simply marks the point in the flow where a new line should begin. Most browsers simply stop adding words and images to the current line, move down and over to the left margin, and resume filling and wrapping.

> [5] With XHTML, put the end inside the start tag: `<br />`. See Chapter 16 for details.

This effect is handy when formatting conventional text with fixed line breaks, such as addresses, song lyrics, or poetry. Notice, for example, the lyrical breaks when the following source is rendered by Internet Explorer:

```
<h3>
Heartbreak Hotel</h3>
<p>
Ever since my baby left me<br>
I've found a new place to dwell.<br>
It's down at the end of lonely street<br>
Called <cite>Heartbreak Hotel</cite>.
</p>
```

The results are shown in Figure 4-18.

Function:

Insert a line break into a text flow

Attributes:

CLASS	STYLE
CLEAR	TITLE
ID	

End tag:

None in HTML; </br> or <br ... /> in XHTML

Contains:

Nothing

Used in:

text

**Figure 4-18. Give lyrics their breaks (
)**

Also notice how the
 tag causes text simply to start a new line, while the browser, when encountering the <p> tag, typically inserts some vertical space between adjacent paragraphs. Section 4.1.2

4.7.1.1 The clear attribute

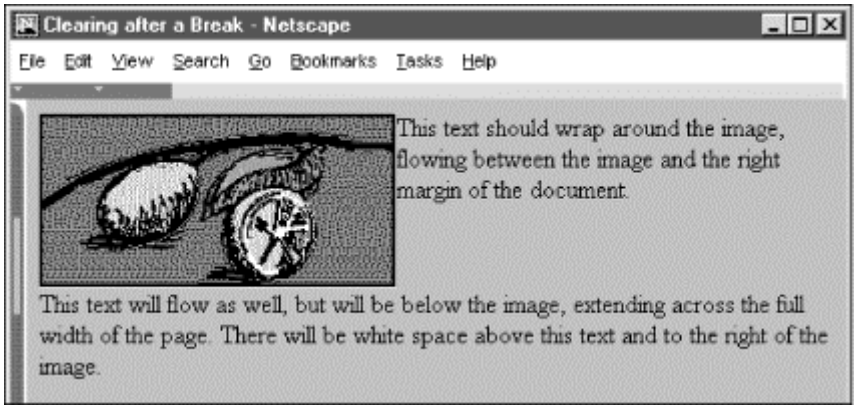
Normally, the
 tag tells the browser to stop the current flow of text immediately and resume at the left margin of the next line or against the right border of a left-justified inline graphic or table. Sometimes you'd rather the current text flow resume below any tables or images currently blocking the left or right margins.

HTML 4 and XHTML provide that capability with the `clear` attribute for the
 tag. It can have one of three values: `left`, `right`, or `all`, each related to one or both of the margins. When the specified margin or both margins are clear of images, the browser resumes the text flow.

Figure 4-19 illustrates the effects of the `clear` attribute when the browser renders the following HTML fragment:

```
<img src="kumquat.gif" align=left>
This text should wrap around the image, flowing between the
image and the right margin of the document.
<br clear=left>
This text will flow as well, but will be below the image,
extending across the full width of the page. There will
be white space above this text and to the right of the
image.
```

**Figure 4-19. Clearing images before resuming text flow after the
 tag**



Inline images are just that - normally in line with text, but usually only a single line of text. Additional lines of text flow below the image unless that image is specially aligned by `right` or `left` attribute values for the `<img>` tag (similarly for `<table>`). Hence, the `clear` attribute for the `<br>` tag only works in combination with left- or right-aligned images or tables.Section 5.2.6.4 / Section 10.2.1.1

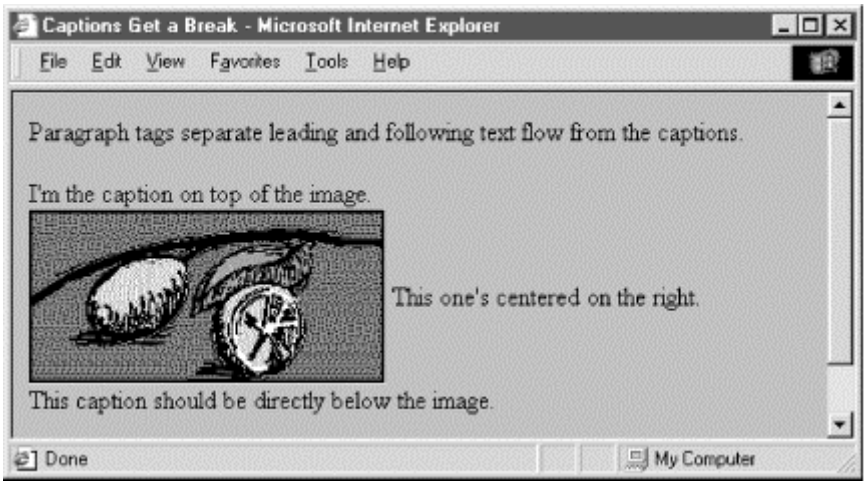
The following XHTML code fragment illustrates how to use the `<br>` tag and its `clear` attribute as well as the `<img>` tag's alignment attributes to place captions directly above, centered on the right, and below an image that is aligned against the left margin of the browser window:

```
Paragraph tags separate leading and following
text flow from the captions.
<p>
I'm the caption on top of the image.
<br />
<img src="kumquat.gif" align="absmiddle"/>
This one's centered on the right.
<br clear="left" />
This caption should be directly below the image.
</p>
<p>
```

Figure 4-20 illustrates the results of this example code.

You might also include a `<br clear=all>` tag just after an `<img>` tag or table that is at the very end of a section of your document. That way, you ensure that the subsequent section's text doesn't flow up and against that image and confuse the reader. Section 5.2.6

Figure 4-20. Captions placed on top, center-right, and below an image



4.7.1.2 The class, id, style, and title attributes

You can associate additional display rules for the
 tag using style sheets. The rules can be applied to the
 tag using either the style or class attributes. Section 8.1.1 / Section 8.3

You also may assign a unique id to the
 tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.7.2 The <nobr> Tag

Occasionally, you may have a phrase you want to appear unbroken on a single line in the user's browser window, even if that means the text extends beyond the visible region of the window. Computer commands are good examples. Typically, you type in a computer command - even a multiword one - on a single line. Because you cannot predict exactly how many words will fit inside an individual's browser window, the sequence of computer-command words may end up broken into two or more lines of text. Command syntax is confusing enough; it doesn't need the extra cross-eyed effect of being wrapped onto two lines.

With standard HTML and XHTML, the way to make sure text phrases stay intact across the browser display is to enclose those segments in a <pre> tag and format it by hand. That's acceptable and nearly universal for all browsers. However, <pre> alters the display font from the regular text, and manual line breaks inside the <pre> tag are not always rendered correctly. Section 4.7.5

<nobr>

Function:

Create a region of non-breaking text

Attributes:

None

End tag:

</nobr>; always used

Contains:

text

Used in:

block

The modern browsers offer the <nobr> tag alternative to <pre> so you can be sure enclosed text stays intact on a single line while retaining normal text style.[6] The effect is to make the browser treat the tag's contents as though they were a single, unbroken word. The tag contents retain the current font style, and you can change to another style within the tag.

[6] Be aware that <nobr> and its colleague <wbr> are extensions to the language and not part of the HTML standard.

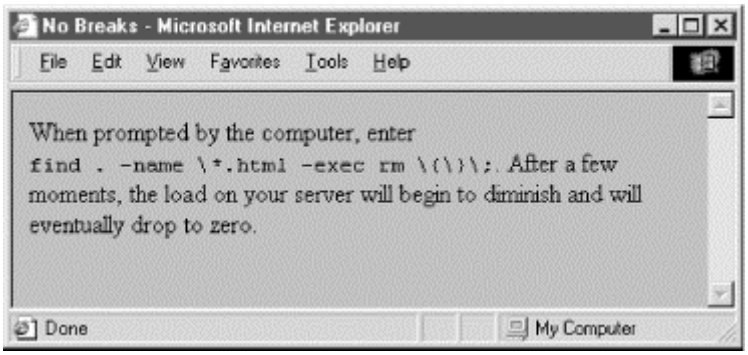
Here's the <nobr> tag in action with our computer command example:

```
When prompted by the computer, enter
<nobr>
<tt>find . -name \*.html -exec rm \{\}\;</tt>.
</nobr>
After a few moments, the load on your server will begin
to diminish and will eventually drop to zero.
```

Notice in the example source and its display (Figure 4-21) that we've included the special <tt> tag inside the <nobr> tag. If the <nobr>-tagged text cannot fit on a partially filled line of text, the extended browser precedes it with a line break, as shown in the figure. The <nobr> segment may then extend beyond the right window boundary. Section 4.5.10

The <nobr> tag does not suspend the browser's normal line-filling process; it still collects and inserts images and - believe it or not - asserts forced line breaks caused by the
 or <p> tags, for example. The <nobr> tag's only action is to suppress an automatic line break when the current line reaches the right margin.

Figure 4-21. The <nobr> extension suppresses text wrapping



In addition, you might think this tag is needed only to suppress line breaks for phrases, not a sequence of characters without spaces that can exceed the browser window's display boundaries. Today's browsers do not hyphenate words automatically, but someday soon they probably will. It makes sense to protect any break-sensitive sequence of characters with the <nobr> tag.

4.7.3 The <wbr> Tag

The <wbr> tag is the height of text-layout finesse, offered as an extension to the languages by the popular browsers. Used with the <nobr> tag, <wbr> advises the extended browser when it may insert a line break in an otherwise nonbreakable sequence of text. Unlike the
 tag, which always causes a line break even within a <nobr>-tagged segment, the <wbr> tag works only when placed inside a <nobr>-tagged content segment and causes a line break only if the current line already had extended beyond the browser's display window margins.

<wbr> NO

Function:

Define potential line break point if needed

Attributes:

None

End tag:

None in HTML; </wbr> or <wbr ... /> in XHTML

Contains:

Nothing

Used in:

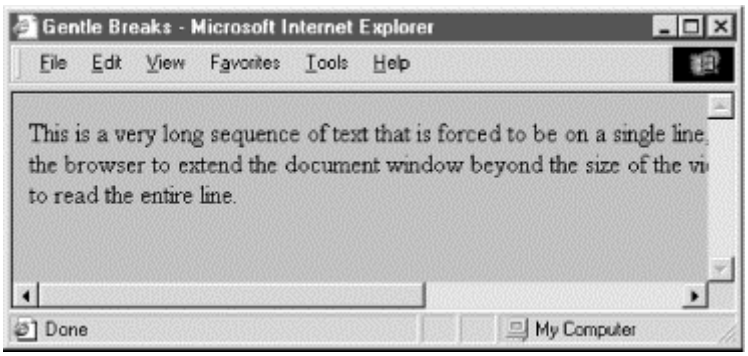
text

Now, <wbr> may seem incredibly esoteric to you, but scowl not. There may come a time when you want to make sure portions of your document appear on a single line, but you don't want to overrun the browser window margins so far that readers will have to camp on the horizontal scrollbar just to read your fine prose. By inserting the <wbr> tag at appropriate points in the nonbreaking sequence, you let the browser gently break the text into more manageable lines:

```
<p>
<nobr>
This is a very long sequence of text that is
forced to be on a single line, even if doing so causes
<wbr>
the browser to extend the document window beyond the
size of the viewing pane and the poor user must scroll right
<wbr>
to read the entire line.
</nobr>
```

You'll notice in our rendered version (Figure 4-22) that both <wbr> tags take effect. By increasing the horizontal window size or by reducing the font size, you may fit all of the segment before the first <wbr> tag within the browser window. In that case, only the second <wbr> would have an effect; all the text leading up to it would extend beyond the window's margins.

Figure 4-22. Gentle line breaks with <wbr>



4.7.4 Better Line-Breaking Rules

Unlike some browsers, and to their credit, Netscape Navigator and Internet Explorer do not consider tags to be a line-break opportunity. Consider the unfortunate consequences to your document's display if, while rendering the example segment below, the browser puts the comma adjacent to the "du" or the period adjacent to the word "df " on a separate line. Netscape and Internet Explorer will not:

```
Make sure you type <tt>du</tt>, not <tt>df</tt>.
```

4.7.5 The <pre> Tag

The <pre> tag and its required end tag (</pre>) define a segment inside which the browser renders text in exactly the character and line spacing defined in the source document. Normal word wrapping and paragraph filling are disabled, and extraneous leading and trailing spaces are honored. The browser displays all text between the <pre> and </pre> tags in a monospaced font.

Authors most often use the <pre> formatting tag when the integrity of columns and rows of characters must be retained; for instance, in tables of numbers that must line up correctly. Another application for <pre> is to set aside a blank segment - a series of blank lines - in the document display, perhaps to clearly separate one content section from another, or to temporarily hide a portion of the document when it first loads and is rendered by the user's browser.

Tab characters have their desired effect within the <pre> block, with tab stops defined at every eight character positions. We discourage their use, however, since tabs aren't consistently implemented among the various browsers. Use spaces to ensure correct horizontal positioning of text within <pre>-formatted text segments.

A common use of the <pre> tag is to present computer source code, as in the following example:

```
<p>
The processing program is:

<pre>
main(int argc, char **argv)

{   FILE *f;
    int i;

    if (argc != 2)
      fprintf(stderr, "usage: %s &lt;file&gt;\n",
          argv[0]);
    <a href="http:process.c">process</a>(argv[1]);
    exit(0);
}
</pre>
```

The result is displayed by Netscape Navigator as shown in Figure 4-23.

<pre>

Function:

Render a block of text without any formatting

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE
WIDTH	

End tag:

</pre>; never omitted

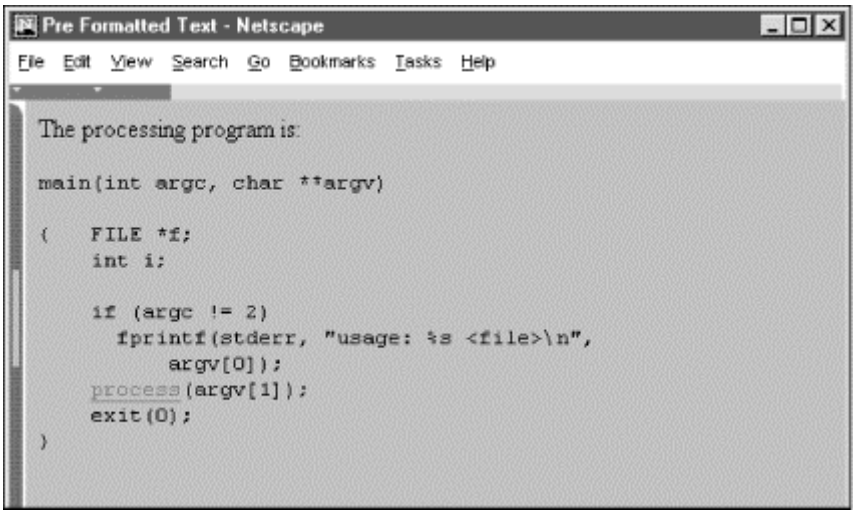
Contains:

pre_content

Used in:

block

Figure 4-23. Use the <pre> tag to preserve the integrity of columns and rows



4.7.5.1 Allowable content

The text within a <pre> segment may contain physical and content-based style changes, along with anchors, images, and horizontal rules. When possible, the browser should honor style changes, within the constraint of using a monospaced font for the entire <pre> block.

Tags that cause a paragraph break (heading, <p>, and <address> tags, for example), must not be used within the <pre> block. Although some browsers will interpret paragraph-ending tags as simple line breaks, this behavior is not consistent across all browsers.

Since style markup and other tags are allowed in a <pre> block, you must use entity equivalents for the literal characters: < for <, > for >, and & for the ampersand.

You place tags into the <pre> block as you would in any other portion of the HTML document. For instance, study the reference to the "process" function in the previous example. It contains a hyperlink (using the <a> tag) to its source file named *process.c*.

4.7.5.2 The width attribute

The <pre> tag has an optional attribute, width, that determines the number of characters to fit on a single line within the <pre> block. The browser may use this value to select a font or font size that fits the specified number of characters on each line in the <pre> block. It does not mean that the browser will wrap and fill text to the specified width. Rather, lines longer than the specified width simply extend beyond the visible region of the browser's window.

The width attribute is only advice for the user's browser; it may or may not be able to adjust the view font to the specified width.

4.7.5.3 The dir and lang attributes

The dir attribute lets you advise the browser as to which direction the text within the <pre> segment ought to be displayed, and lang lets you specify the language used within that tag. Section 3.6.1.1 / Section 3.6.1.2

4.7.5.4 The class, id, style, and title attributes

Although the browsers usually display <pre> content in a defined style, you can override that style and add special effects, such as a background picture, by defining your own style for the tag. This new look can be applied to the <pre> tags using either the style or class attributes. Section 8.1.1 / Section 8.3

You also may assign a unique id to the <pre> tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.7.5.5 Event attributes

Like with most other tagged segments of content, user-related events can happen in and around <pre> content, such as when a user clicks or double-clicks within its display space. Many of these events are recognized by current browsers. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.7.6 The Tag (Deprecated)

The tag is another of those whose effects are obvious: content, including text, graphics, tables, and so on, are each centered inside the browser's window. For text, this means that each line, individually, gets centered after the text flow is filled and wrapped. The alignment remains in effect until canceled with its end tag.

Line-by-line is a common, albeit primitive, way to center text, and it should be used judiciously. That's because the browsers do not attempt to balance a centered paragraph or other block-related elements, such as elements in a list. So keep your centered text short and sweet. Titles make good centering candidates; a centered list usually is difficult to follow.

Beyond that, you'll rarely see conventional text centered, except for some lyrical prose, so readers may react badly to large segments of centered prose in your documents. Rather, HTML authors more commonly use to center a table or image in the display window (there is no explicit center alignment option for inline images or tables, but there are styles-related ways to achieve the effect).

Function:

Center a section of text

Attributes:

ALIGN	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	

End tag:

; never omitted

Contains:

body_content

Used in:

block

Because users will have varying window widths, display resolutions, and so on, you may also want to employ the `<nobr>` and `<wbr>` extension tags (see previous descriptions) to keep your centered text intact and looking good. For example:

```
<center>
<nobr>
Copyright 1995 by QuatCo Enterprises.<wbr>
All rights reserved.
</nobr>
</center>
```

The `<nobr>` tags in the sample source help ensure that the text remains on a single line, and the `<wbr>` tag controls where the line may be broken if it exceeds the browser's display window width.

Centering is useful for creating distinctive section headers, although you may achieve the same effect with an explicit `align=center` attribute in the respective heading tag. You might also center text using `align=center` in conjunction with the `<div>` or `<p>` tags. In general, the `<center>` tag can be replaced by an equivalent `<div align=center>` or similar tag and its use should be discouraged.

Indeed, like `<font>` and other HTML 3.2 standard tags that have fallen in disfavor in the wake of style sheets, the `<center>` tag is deprecated in the HTML 4 and XHTML standards. Nonetheless, its use in HTML documents is nearly universal, and the popular browsers are sure to support it for many revisions to come. Still, be aware of its eventual demise.

4.7.6.1 The dir and lang attributes

The `dir` attribute lets you advise the browser as to which direction the text within the `<center>` segment ought to be displayed, and `lang` lets you specify the language used within the tag. Section 3.6.1.1 / Section 3.6.1.2

4.7.6.2 The class, id, style, and title attributes

Use the `style` attribute to specify an inline style for the `<center>` tag, or use the `class` attribute to apply a predefined style class to the tag. Section 8.1.1 Section 8.3

You may assign a unique id to the `<center>` tag, as well as a title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.7.6.3 Event attributes

Like with most other tagged segments of content, user-related events can happen in and around the `<center>` tag, such as when a user clicks or double-clicks within its display space. Many of these events are recognized by the current browsers. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.7.7 The <listing> Tag (Obsolete)

The `<listing>` tag is an obsolete tag, explicitly removed from the HTML 4 standard, meaning that you shouldn't use it. We include it here for historical reasons, since it is supported by some browsers and has the same effect on text formatting as the `<pre>` tag with a specified width of 132 characters.

<listing>

Function:

Render a block of text without any formatting

Attributes:

CLASS NO
STYLE NO

End tag:

</listing>; never omitted

Contains:

literal_text

Used in:

block

The only difference between `<pre>` and `<listing>` is that no other markup is allowed within the `<listing>` tag. So you don't have to replace the literal <, >, and & characters with their entity equivalents in a `<listing>` block as you must inside a `<pre>` block.

Since the `<listing>` tag is the same as a `<pre width=132>` tag, and because it might not be supported in later version of the language, we recommend that you stay away from using `<listing>`.

4.7.8 The <xmp> Tag (Obsolete)

Like the <listing> tag, the <xmp> tag is obsolete and should not be used. We include it here mostly for historical reasons.

<xmp>

Function:

Render a block of text without any formatting

Attributes:

CLASS NO
STYLE NO

End tag:

</xmp>; never omitted

Contains:

literal_text

Used in:

block

The <xmp> tag formats text just like the <pre> tag with a specified width of 80 characters. However, unlike the <pre> tag, you don't have to replace the literal <, >, and & characters with their entity equivalents within an <xmp> block. The name <xmp> is short for "example"; the language's designers intended that the tag be used to format examples of text originally displayed on 80-column wide displays. Because the 80-column display has mostly gone the way of green screens and teletypes, and since the effect of a <xmp> tag is basically the same as <pre width=80>, don't use <xmp>; it may disappear in subsequent versions of HTML.

4.7.9 The <plaintext> Tag (Obsolete)

Throw the <plaintext> tag out of your bag of HTML tricks; it's obsolete, like <listing> and <xmp>. Included here for historical reasons, authors once used <plaintext> to tell the browser to treat the rest of your document's text just as written with no markup allowed. There was no ending tag for <plaintext> (of course, no markup!), but there was an end to <plaintext>. Forget about it.

<plaintext>

Function:

Render a block of text without any formatting

Attributes:

None

End tag:

None

Contains:

literal_text

Used in:

block

4.8 Block Quotes

A common element in conventional documents is the block quote, a lengthy copy of text from another document. Traditionally, short quotes are set off with quotation marks, while block quotes are made entirely of separate paragraphs within the main document, typically with special indentation and sometimes italicized - features that you may change through style or class definitions (see Chapter 8).

4.8.1 The <blockquote> Tag

All of the text within the `<blockquote>` and `</blockquote>` tags is set off from the regular document text, usually with indented left and right margins, and sometimes in italicized typeface. Actual rendering varies from browser to browser, of course.

<blockquote>

Function:

Define a block quotation

Attributes:

CITE	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	

End tag:

</blockquote>; never omitted

Contains:

body_content

Used in:

block

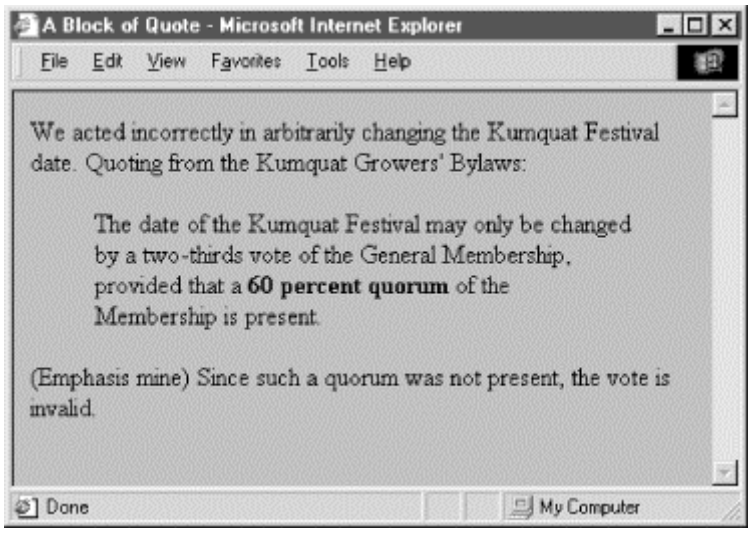
The HTML and XHTML standards allow any and all markup within the `<blockquote>`, although some physical and content-based styles may conflict with the font used by the browser for the block quote. Experimentation will reveal those little warts.

The <blockquote> tag is often used to set off long quotations from other sources. For example:

```
We acted incorrectly in arbitrarily changing the Kumquat
Festival date. Quoting from the Kumquat Growers' Bylaws:
<blockquote>
  The date of the Kumquat Festival may only be changed by
  a two-thirds vote of the General Membership, provided
  that a <strong>60 percent quorum</strong> of the Membership
  is present.
</blockquote>
(Emphasis mine) Since such a quorum was not present, the
vote is invalid.
```

gets displayed by Internet Explorer as an indented block of text. Figure 4-24 displays the results.

Figure 4-24. Blockquotes get their own space



4.8.1.1 The cite attribute

The cite attribute lets you indicate the source of a quote. The attribute's value should be a quote-enclosed URL that points to the online document and, if possible, the exact location in the document where the quote came from.

For instance, you could cite the specific section in the Kumquat Grower's Bylaws in our example. Presumably, someday the browser may actually let you click and view that specific citation via its embedded URL. Today, you must embed an explicit hyperlink to the document; see Chapter 6:

```
<blockquote cite="http://www.kumquat.com/growers/bylaws#s23.4">
```

4.8.1.2 The dir and lang attributes

The dir attribute lets you advise the browser as to which direction the text within the <blockquote> segment ought to be displayed, and lang lets you specify the language used within that tag. Section 3.6.1.1 / Section 3.6.1.2

4.8.1.3 The class, id, style, and title attributes

Use the style attribute to specify an inline style for the <blockquote> tag, or use the class attribute to apply a predefined style class to the tag. Section 8.1.1 / Section 8.3

You may assign a unique id to the <blockquote> tag, as well as a title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.8.1.4 Event attributes

Like with most other tagged segments of content, user-related events can happen in and around the <blockquote> tag, such as when a user clicks or double-clicks within its display space. Many of these events are recognized by the current browsers. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.8.2 The <q> Tag

Introduced in HTML 4.0, the <q> tag is virtually identical to its <blockquote> counterpart. The difference is in their display and application. Use <q> for short quotes that may be in line with surrounding plain text. Although not yet supported by the popular browsers, the HTML and XHTML standards dictate that the <q>-enclosed text begin and end with double-quote marks. Use the <blockquote> tag, on the other hand, for longer segments that the browser will set off - usually as an indented block - from the surrounding content, such as that shown in Figure 4-24.

<q>

Function:

Define a short quotation

Attributes:

CITE	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	

End tag:

</q>; never omitted

Contains:

body_content

Used in:

text

4.8.2.1 The cite attribute

The cite attribute works with the <q> tag just like it does for the <blockquote> tag: it lets you indicate the source of a quote. The attribute's value should be a quote-enclosed URL that points to the online document and, if possible, the exact location in the document where the quote came from.

4.8.2.2 The dir and lang attributes

The dir attribute lets you advise the browser as to which direction the text within the <q> segment ought to be displayed, and lang lets you specify the language used within that tag. Section 3.6.1.1 / Section 3.6.1.2

4.8.2.3 The class, id, style, and title attributes

Use the `style` attribute to specify an inline style for the `<q>` tag, or use the `class` attribute to apply a predefined style class to the tag. Section 8.1.1 / Section 8.3

You may assign a unique id to the `<q>` tag, as well as a title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.8.2.4 Event attributes

Like with most other tagged segments of content, user-related events can happen in and around the `<q>` tag, such as when a user clicks or double-clicks within its display space. Many of these events are recognized by the current browsers. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.9 Addresses

Addresses are a very common element in text documents, so there is a special tag that sets addresses apart from the rest of a document's text. While this may seem a bit extravagant - addresses have few formatting peculiarities that would require a special tag - it is an example of content, not format, which is the intent and purpose of HTML and XHTML markup.

By defining text that constitutes an address, the author lets the browser format that text in a different manner, as well as process that text in ways helpful to users. It also makes the content readily accessible to automated readers and extractors. For instance, an online directory might include addresses the browser collects into a separate document or table, or automated tools might extract addresses from a collection of documents to build a separate database of addresses.

4.9.1 The <address> Tag

The `<address>` and its required end (`</address>`) tag tell a browser that the enclosed text is an address. The browser may format the text in a different manner than the rest of the document text, or use the address in some special way. You also have control over the display properties through the style and class attributes for the tag (see Chapter 8).

The text within the `<address>` tag may contain any element normally found in the body of a document, excluding another `<address>` tag. Style changes are allowed, but may conflict with the style chosen by the browser to render the address element.

We think most, if not all, documents should have their authors' addresses included somewhere convenient to the user, usually at the end. At the very least, the address should be the author's or webmaster's email address, along with a link to their home page. Street addresses and phone numbers are optional; personal ones are usually not included for reasons of privacy.

For example, the address for the webmaster responsible for a collection of commercial web documents often appears in source documents as follows, including the special `mailto:` URL protocol that lets users activate the browser's email tool:

```
<address>
  <a href="mailto:webmaster@oreilly.com">Webmaster</a><br>
  O'Reilly & Associates, Inc.<br>
  Cambridge, Massachusetts<br>
</address>
```

Figure 4-25 displays the results.

Figure 4-25. The <address> tag in action

<address>

Function:

Define an address

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

</address>; never omitted

Contains:

body_content

Used in:

address_content

Whether it is short and sweet or long and complete, make sure every document you create has an address attached to it. If something is worth creating and putting on the Web, it is worth comment and query by your readership. Anonymous documents carry little credibility on the Web.

4.9.1.1 The dir and lang attributes

The `dir` attribute lets you advise the browser as to which direction the text within the `<address>` segment ought to be displayed, and `lang` lets you specify the language used within that tag. Section 3.6.1.1 / Section 3.6.1.2

4.9.1.2 The class, id, style, and title attributes

Use the `style` attribute to specify an inline style for the `<address>` tag, or use the `class` attribute to apply a predefined style class to the tag. Section 8.1.1 / Section 8.3

You may assign a unique id to the `<address>` tag, as well as a title, using the respective attribute and accompanying quote-enclosed string value. Section 4.1.1.4 / Section 4.1.1.5

4.9.1.3 Event attributes

Like with most other tagged segments of content, user-related events can happen in and around the `<address>` tag, such as when a user clicks or double-clicks within its display space. Many of these events are recognized by the current browsers. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

4.10 Special Character Encoding

For the most part, characters within documents that are not part of a tag are rendered as is by the browser. However, some characters have special meaning and are not directly rendered, while other characters can't be typed into the source document from a conventional keyboard. Special characters need either a special name or a numeric character encoding for inclusion in a document.

4.10.1 Special Characters

As has become obvious in the discussion and examples leading up to this section, three characters in source documents have very special meaning: the less-than sign (<), the greater-than sign (>), and the ampersand (&). These characters delimit tags and special character references. They'll confuse a browser if left dangling alone or with improper tag syntax. So you've got to go out of your way to include their actual, literal characters in your documents.[7]

> [7] The only exception is that these characters may appear literally within the `<listing>` and `<xmp>` tags, but this is a moot point, since the tags are obsolete.

Similarly, you've got to use a special encoding to include double quotation mark characters within a quoted string, or when you want to include a special character that doesn't appear on your keyboard but is part of the ISO Latin-1 character set implemented and supported by most browsers.

4.10.2 Inserting Special Characters

To include a special character in your document, enclose either its standard entity name or a pound sign (#) and its numeric position in the Latin-1 standard character set[8] inside a leading ampersand and an ending semicolon, without any spaces in-between.

> [8] The popular ASCII character set is a subset of the more comprehensive Latin-1 character set. Composed by the well-respected International Organization for Standardization (ISO), the Latin-1 set is a list of all letters, numbers, punctuation marks, and so on, commonly used by Western language writers, organized by number and encoded with special names. Appendix F contains the complete Latin-1 character set and encoding.

Whew. That's a long explanation for what is really a simple thing to do, as the following example illustrates. The example shows how to include a greater-than sign in a snippet of code by using the character's entity name. It also demonstrates how to include a greater-than sign in your text by referencing its Latin-1 numeric value:

```
if a &gt; b, then t = 0
if a &#62; b, then t = 0
```

Both examples cause the text to be rendered as:

```
if a > b, then t = 0
```

The complete set of character entity values and names are in Appendix F. You could write an entire document using character encoding, but that would be silly.

Chapter 5. Rules, Images, and Multimedia

While the body of most documents is text, an appropriate seasoning of horizontal rules, images, and other multimedia elements make for a much more inviting and attractive document. These features are not simply gratuitous geegaws that make your documents look pretty, mind you. Multimedia elements bring HTML and XHTML documents alive, providing a dimension of valuable information often unavailable in other media, such as print. In this chapter, we describe in detail how you can insert special multimedia elements into your documents, when their use is appropriate, and how to avoid overdoing it.

You also might want to jump ahead and skim Chapter 12. There we describe some catch-all tags, the HTML 4 and XHTML standard `<object>` and Netscape's `<embed>`, which let you insert all kinds of content and data file types, including multimedia, into your documents.

5.1 Horizontal Rules

Horizontal rules give you a way to separate sections of your document visually. That way, you give readers a clean, consistent, visual indication that one portion of your document has ended and another portion is beginning. Horizontal rules effectively set off small sections of text, delimit document headers and footers, and provide extra visual punch to headings within your document.

5.1.1 The <hr> Tag

The `<hr>` tag tells the browser to insert a horizontal rule across the display window. Like the `<br>` tag, `<hr>` forces a simple line break, although unlike `<br>`, `<hr>` causes the paragraph alignment to revert to the default (left-justified). The browser places the rule immediately below the current line, and content flow resumes below the rule. Section 4.7.1

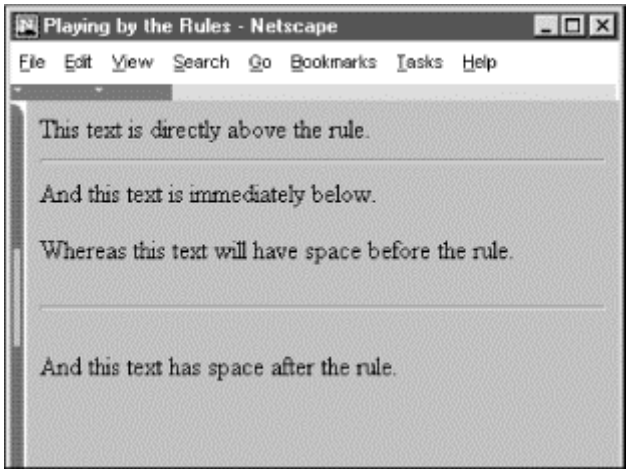
The rendering of a horizontal rule is at the discretion of the browser. Typically, it extends across the entire document. Graphical browsers may render the rule with a chiseled or embossed effect; character-based browsers most likely use dashes or underscores to create the rule.

There is no additional space above or below a horizontal rule. If you wish to set it off from the surrounding text, you must explicitly place the rule in a new paragraph, followed by another paragraph containing the subsequent text. For example, note the spacing around the horizontal rules in the following source and in Figure 5-1:

```
This text is directly above the rule.
<hr>
And this text is immediately below.
<p>
Whereas this text will have space before the rule.
<p>
<hr>
<p>
And this text has space after the rule.
```

A paragraph tag following the rule tag is necessary if you want the content beneath the rule line aligned in any style other than the default left.

Figure 5-1. Paragraph tags give your text extra elbow room



<hr>

Function:

Break a text flow and insert a horizontal rule

Attributes:

ALIGN	ONMOUSEDOWN
CLASS	ONMOUSEMOVE
COLOR	ONMOUSEOUT
DIR	ONMOUSEOVER
ID	ONMOUSEUP
LANG	SIZE
NOSHADE	STYLE
ONCLICK	TITLE
ONDBLCLICK	WIDTH
ONKEYDOWN	
ONKEYPRESS	
ONKEYUP	

End tag:

None in HTML; </hr> or <hr ... /> with XHTML

Contains:

Nothing

Used in:

body_content

5.1.1.1 The size attribute

Normally, browsers render horizontal rules two to three pixels[1] thick with a chiseled, 3D appearance, making the rule look incised into the page.

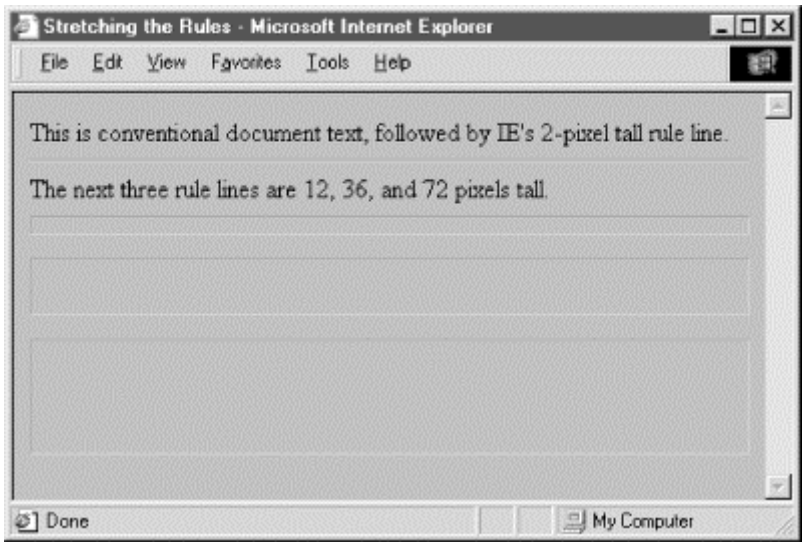
[1] A pixel is one of the many tiny dots that make up the display on your computer. While display sizes vary, a good rule of thumb is that one pixel equals one point on a 75 dot-per-inch display monitor. A point is a unit of measure used in printing and is roughly equal to $^1/_{72}$ of an inch (there are 72.27 points in an inch, to be exact). Typical typefaces used by various browsers are usually 12 points tall, yielding six lines of text per inch.

You may thicken the rules with the `size` attribute. The required value is the thickness, in pixels. You can see the effects of this attribute in Figure 5-2 as constructed from the following source:

```
<p>
This is conventional document text,
followed by a IE's 2-pixel tall rule line.
<hr>
The next three rule lines are 12, 36, and 72 pixels tall.
<hr size=12><hr size=36><hr size=72>
```

The size attribute is deprecated in HTML 4 and XHTML, since its effects can be achieved with appropriate use of style sheets.

Figure 5-2. Internet Explorer and Netscape let you vary the horizontal rule size



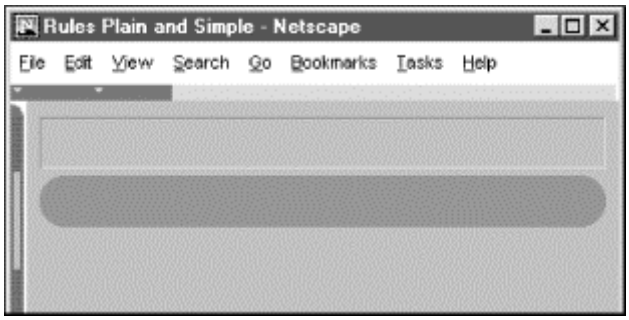
5.1.1.2 The noshade attribute

You may not want a 3D rule line, preferring a flat, 2D rule. Just add the noshade attribute to the <hr> tag to eliminate the effect. No value is required with HTML. Use noshade="noshade" with XHTML. Note the difference in appearance of a "normal" 3D rule versus the noshade 2D one in Figure 5-3. (We've also exaggerated the rule's thickness for obvious effect, as evident in the source HTML fragment.)

```
<hr size=32>
<p>
<hr size=32 noshade>
```

The noshade attribute is deprecated in HTML 4 and XHTML, since its effects can be achieved with appropriate use of style sheets.

Figure 5-3. Netscape's 3D rule versus the noshade 2D option



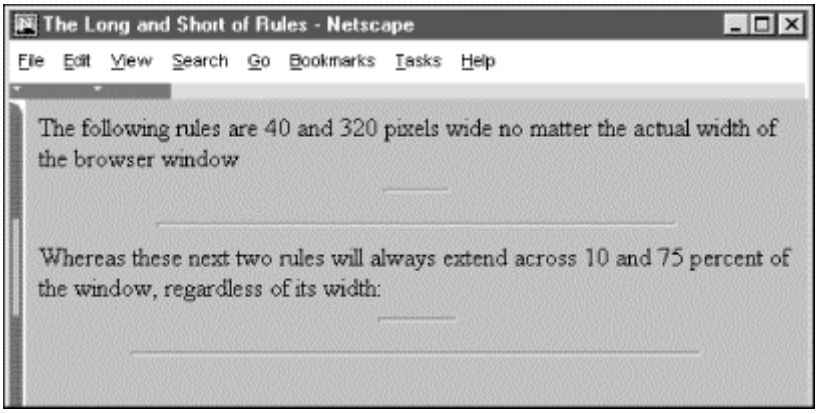
5.1.1.3 The width attribute

The default rule is drawn across the full width of the view window. You can shorten or lengthen rules with the width attribute, creating rule lines that are either an absolute number of pixels wide or extend across a certain percentage of the current text flow. Most browsers automatically center partial-width rules; see the align attribute (Section 5.1.1.4) to left- or right-justify horizontal rules.

Here are some examples of width-specified horizontal rules (see Figure 5-4):

```
The following rules are 40 and 320 pixels wide
no matter the actual width of the browser window
<hr width=40>
<hr width=320>
Whereas these next two rules will always extend across
10 and 75 percent of the window, regardless of its width:
<hr width="10%">
<hr width="75%">
```

Figure 5-4. The long and short of absolute and relative rule widths



Notice, too, that the relative (percentage) value for the `width` attribute is enclosed in quotation marks; the absolute (integer) pixel value is not. In fact, the quotation marks aren't absolutely necessary with standard HTML,[2] but since the percent symbol normally means that an encoded character follows, failure to enclose the percent width value in quotation marks may confuse other browsers and trash a portion of your document.

[2] With XHTML, double quotes are required around all attribute values.

In general, it isn't a good idea to specify the width of a rule as an exact number of pixels. Browser windows vary greatly in their width, and what might be a small rule on one browser might be annoyingly large on another. For this reason, we recommend specifying rule width as a percentage of the window width. That way, when the width of the browser window changes, the rules retain their same relative size.

The `width` attribute is deprecated in HTML 4 and XHTML, since its effects can be achieved with appropriate use of style sheets.

5.1.1.4 The align attribute

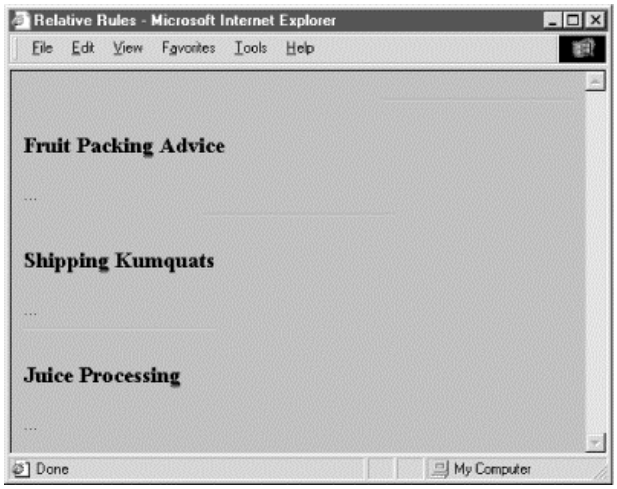
The `align` attribute for a horizontal rule can have one of three values: `left`, `center`, or `right`. For those rules whose width is less than the current text flow, the rule will be positioned relative to the window margins accordingly. The default alignment is `center`.

A varied rule alignment makes for nice section dividers. For example, the source shown below alternates a 35 percent-wide rule from right to center to the left margin (see Figure 5-5):

```
<hr width="35%" align=right>   <h3>Fruit Packing Advice</h3>
...
<hr width="35%" align=center>   <h3>Shipping Kumquats</h3>
...
<hr width="35%" align=left>   <h3>Juice Processing</h3>
...
```

The `align` attribute is deprecated in HTML 4 and XHTML, since its effects can be achieved with appropriate use of style sheets.

Figure 5-5. Varying horizontal rule alignment makes for subtle section dividers



5.1.1.5 The color attribute

Supported only by Internet Explorer, the `color` attribute lets you set the color of the rule line. The value of this attribute is either the name of a color or a hexadecimal triplet that defines a specific color. For a complete list of color names and values, see Appendix G.

By default, a rule is set to the same color as the document background, with the chiseled edges slightly darker and lighter than the background color. You lose the 3D effect when you specify another color, either in a style sheet or with the `color` attribute.

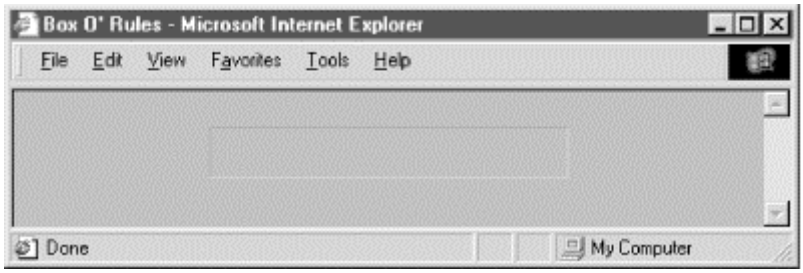
5.1.1.6 Combining rule attributes

You may combine the various rule attributes; their order isn't important. To create big rectangles, for example, combine the `size` and `width` attributes (see Figure 5-6):

```
<hr size=32 width="50%" align=center>
```

In fact, some combinations of rule attributes are necessary - `align` and `width`, for example. `Align` alone appears to do nothing because the default rule width stretches all the way across the display window.

Figure 5-6. Combining rule attributes for special effects



5.1.1.7 The class, dir, event, id, lang, style, and title attributes

There are several nearly universal attributes for the many content tags. These attributes give you a common way to identify (`title`) and label (`id`) a tag's contents for later reference or automated treatment, to change the contents' display characteristics (`class`, `style`), and to reference the language (`lang`) used and related direction the text should flow (`dir`). Of course, how language and the direction of text affect a horizontal rule is unclear. Nonetheless, they are standard attributes for the tag. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.4 / Section 4.1.1.5 / Section 8.1.1 / Section 8.3

In addition, there are all the user events that may happen in and around the horizontal rule that the browser senses and that you may react to via an on-event attribute and some programming. Section 12.3.3

5.1.2 Using Rules to Divide Your Document

Horizontal rules provide a handy visual navigation device for your readers. To use `<hr>` effectively as a section divider, first determine how many levels of headings your document has and how long you expect each section of the document to be. Then decide which of your headings warrant being set apart by a rule.

A horizontal rule can also delimit the front matter of a document, separating the table of contents from the document body, for example. Use a horizontal rule also to separate the document body from a trailing index, bibliography, or list of figures.

Experienced authors also use horizontal rules to mark the beginning and end of a form. This is especially handy for long forms that make users scroll up and down the page to view all the fields. By consistently marking the beginning and end of a form with a rule, you help users stay within the form, better ensuring they won't inadvertently miss a portion when filling out its contents.

5.1.3 Using Rules in Headers and Footers

A fundamental style approach to creating document families is to have a consistent look and feel, including a standard header and footer for each document. Typically, the header contains navigational tools that help users easily jump to internal sections as well as related documents in the family, while the footer contains author and document information as well as feedback mechanisms like an email link to the webmaster.

To ensure that these headers and footers don't infringe on the main document contents, consider using rules directly below the header and above the footer. For example (see also Figure 5-7):

```
<body>
Kumquat Growers Handbook - Growing Season Guidelines
<hr>
<h1>Growing Season Guidelines</h1>
Growing season for the noble fruit varies throughout the
United States, as shown in the following map:
<p>
<img src="pics/growing-season.gif">
<p>
<hr>
<i>Provided as a public service by the
<a href="feedback.html">Kumquat Lovers of America</a></i>
```

Figure 5-7. Clearly delineate headers and footers with horizontal rules

By consistently setting apart your headers and footers using rules, you help users locate and focus upon the main body of your document.

5.2 Inserting Images in Your Documents

One of the most compelling features of HTML and XHTML is their ability to include images with your document text, either as an intrinsic component of the document (inline images), as separate documents specially selected for download via hyperlinks, or as a background for your document. When judiciously added to the body content, images - static and animated icons, pictures, illustrations, drawings, and so on - can make your documents more attractive, inviting, and professional looking, as well as informative and easy to browse. You may also specially enable an image so that it becomes a visual map of hyperlinks. When used to excess, however, images make your document cluttered, confusing, and inaccessible, as well as unnecessarily lengthening the time it takes for users to download and view your pages.

5.2.1 Understanding Image Formats

Neither HTML nor XHTML prescribe an official format for images. However, the popular browsers specifically accommodate certain image formats: GIF and JPEG, in particular (see following sections for explanations). Most other multimedia formats require special accessory applications that each browser owner must obtain, install, and successfully operate to view the special files. So it's not too surprising that GIF and JPEG are the *de facto* image standards on the Web.

Both image formats were already in widespread use before the Web came into being, so there's lots of supporting software out there to help you prepare your graphics for either format. However, each has its own advantages and drawbacks, including features that some browsers exploit for special display effects.

5.2.1.1 GIF

The Graphics Interchange Format (GIF) was first developed for image transfer among users of the CompuServe online service. The format has several features that make it popular for use in HTML and XHTML documents. Its encoding is cross-platform, so that with appropriate GIF decoding software (included with most browsers), the graphics you create and make into a GIF file on a Macintosh, for example, can be loaded into a Windows-based PC, decoded, and viewed without a lot of fuss. The second main feature is that GIF uses special compression technology that can significantly reduce the size of the image file for faster transfer over a network. GIF compression is "lossless," too; none of an image's original data is altered or deleted, so the uncompressed and decoded image exactly matches its original. And GIF images can be easily animated.

Even though GIF image files invariably have the *.gif* (or *.GIF*) filename suffix, there actually are two GIF versions: the original GIF87 and an expanded GIF89a, which supports several new features, including transparent backgrounds, interlaced storage, and animation, that are popular with web authors (see section 5.2.1.2). The currently popular browsers support both GIF versions, which use the same encoding scheme that maps 8-bit pixel values to a color table, for a maximum of 256 colors per image. Most GIF images have even fewer colors; there are special tools to simplify the colors in more elaborate graphics. By simplifying the GIF images, you create a smaller color map and enhance pixel redundancy for better file compression and consequently faster downloading.

However, because of the limited number of colors, a GIF-encoded image is not always appropriate, particularly for photorealistic pictures (see JPEG discussion in Section 5.2.1.3). GIFs make excellent icons, reduced color images, and drawings.

Because most graphical browsers explicitly support the GIF format, it is currently the most widely accepted image-encoding format on the Web. It is acceptable for both inline images and externally linked ones. When in doubt as to which image format to use, choose GIF.[3] It will work in almost any situation.

[3] We cannot resist the temptation to point out that choosy authors choose GIF.

5.2.1.2 Interlacing, transparency, and animation

GIF images can be made to perform three special tricks: interlacing, transparency, and animation. With interlacing, a GIF image seemingly materializes on the display, rather than progressively flowing onto it from top to bottom. Normally, a GIF encoded image is a sequence of pixel data, in order row-by-row, from top to bottom of the image. While the common GIF image renders onscreen like pulling down a window shade, interlaced GIFs open like a venetian blind. That's because interlacing sequences every fourth row of the image. Users get to see a full image - top to bottom, albeit fuzzy - in a quarter of the time it takes to download and display the remainder of the image. The resulting quarter-done image usually is clear enough so that users with slow network connections can evaluate whether to take the time to download the remainder of the image file.

Not all graphical browsers, although able to display an interlaced GIF, are actually able to display the materializing effects of interlacing. With those that do, users still can defeat the effect by choosing to delay image display until after download and decoding. Older browsers, on the other hand, always download and decode images before display and don't support the effect at all.

Another popular effect available with GIF images - GIF89a-formatted images, actually - is the ability to make a portion of them transparent so that what's underneath - usually the browser window's background - shows through. The transparent GIF image has one color in its color map designated as the background color. The browser simply ignores any pixel in the image that uses that background color, thereby letting the display window's background show through. By carefully cropping its dimensions and by using a solid, contiguous background color, a transparent image can be made to seamlessly meld into a page's surrounding content or float above it.

Transparent GIF images are great for any graphic you want to meld into the document and not stand out as a rectangular block. Transparent GIF logos are very popular, as are transparent icons and dingbats - any graphic that should appear to have an arbitrary, natural shape. You may also insert a transparent image inline with conventional text to act as a special character glyph within conventional text.

The downside to transparency is that the GIF image will look lousy if you don't remove its border when it is included in a hyperlink anchor (<a> tag), or is otherwise specially framed. And content flow happens around the image's rectangular dimensions, not adjacent to its apparent shape. That can lead to unnecessarily isolated images or odd-looking sections in your web pages.

The third unique trick available with GIF89a-formatted images is the ability to do simple frame-by-frame animation. Using special GIF animation software utilities, you may prepare a single GIF89a file to contain a series of GIF images. The browser displays each image in the file, one after the other, something like the page-flipping animation booklets we had (even drew!) as kids. Special control segments between each image in the GIF file let you set the number of times the browser runs through the complete sequence (looping), how long to pause between each image, whether the image space gets wiped to background before the browser displays the next image, and so on. By combining these control features with those normally available for GIF images, including individual color tables, transparency, and interlacing, you can create some very appealing and elaborate animations.[4]

[4] Songline Studios has published an entire book dedicated to GIF animation: *GIF Animation Studio*, by Richard Koman.

Simple GIF animation is powerful for one other important reason: you don't need to specially program your HTML documents to achieve animation. But there is one major downside that limits their use except for small, icon-sized, or thin bands of space in the browser window: GIF animation files get large fast, even if you are careful not to repeat static portions of the image in successive animation cells. And if you have several animations in one document, download delays may - and usually will - annoy the user. If there is any feature that deserves close scrutiny for excess, it's GIF animation.

Any and all GIF tricks - interlacing, transparency, and animation - don't just happen; you need special software to prepare the GIF file. Many image tools now save your creations or acquired images in GIF format, and most now let you enable transparency, as well as let you make interlaced GIF files. There also are a slew of shareware and freeware programs specialized for these tasks, as well as for creating GIF animation. Look into your favorite Internet software archives for GIF graphics and conversion tools and also see Chapter 17 for details on creating transparent images.

5.2.1.3 JPEG

The Joint Photographic Experts Group (JPEG) is a standards body that developed what is now known as the JPEG image-encoding format. Like GIFs, JPEG images are platform-independent and specially compressed for high-speed transfer via digital communication technologies. Unlike GIF, JPEG supports tens of thousands of colors for more detailed, photorealistic digital images. And JPEG uses special algorithms that yield much higher data-compression ratios. It is not uncommon, for example, for a 200-kilobyte GIF image to be reduced to a 30-kilobyte JPEG image. To achieve that amazing compression, JPEG does lose some image data. However, you can adjust the degree of "lossiness" with special JPEG tools, so that although the uncompressed image may not exactly match the original, it will be close enough that most people cannot tell the difference.

Although JPEG is an excellent choice for photographs, it's not a particularly good choice for illustrations. The algorithms used for compressing and uncompressing the image leave noticeable artifacts when dealing with large areas of one color. Therefore, if you're trying to display a drawing, the GIF format may be preferable.

The JPEG format, usually designated by the *.jpg* (or *.JPG*) filename suffix, is nearly universally understood by today's graphical browsers. On rare occasions, you'll come across an older browser that cannot directly display JPEG images.

5.2.2 When to Use Images

Most pictures are worth a thousand words. But don't forget that no one pays attention to a blabbermouth. First and foremost, think of your document images as visual tools, not gratuitous trappings. They should support your text content and help readers navigate your documents. Use images to clarify, illustrate, or exemplify the contents. Content-supporting photographs, charts, graphs, maps, and drawings are all natural and appropriate candidates. Product photographs are essential components in online catalogs and shopping guides, for example. And link-enabled icons and dingbats, including animated images, can be effective visual guides to internal and external resources. If an image doesn't do any of these valuable services for your document, throw it out already!

One of the most important considerations when adding images to a document is the additional delay they add to the retrieval time for a document over the network, particularly for modem connections. While a common text document might run, at most, 10 or 15 thousand bytes, images can easily extend to hundreds of thousands of bytes each. And the total retrieval time for a document is not only equal to the sum of all its component parts, but also to compounded networking overhead delays.

Depending on the speed of the connection (*bandwidth*, usually expressed as bits or bytes per second) as well as network congestion that can delay connections, a single document containing one 100-kilobyte image may take anywhere from around 15 seconds through a 57.6 kilobit-per-second modem connection in the wee hours of the morning when most everyone else is asleep, to well over *ten minutes* with a 9600 bit-per-second modem at noon. You get the picture?

That said, of course, pictures and other multimedia are driving Internet providers to come up with faster, better, more robust ways to deliver Web content. Soon, 57.6 kilobit-per-second modem connections will go the way of the horse and carriage (as 9600 bit-per-second modems already have), to be replaced by technologies like cable modems and ADSL. Indeed, soon most connections will attain data rates approaching or exceeding what used to be available only to the biggest users (besides costing an arm and a leg), over a megabit per second.

Still, as the price lowers, use goes up, so there is the issue of congestion. If you are competing for access to an overburdened server, it doesn't matter how fast your connection may be.

5.2.3 When to Use Text

Text hasn't gone out of style. For some users, it is the only portion of your document they can access. We argue that, in most circumstances, your documents should be usable by readers who cannot view images or have disabled their automatic download in their browser to improve their connection. While the urge to add images to all of your documents may be strong, there are times when pure text documents make more sense.

Documents being converted to the Web from other formats rarely have embedded images. Reference materials and other serious content often is completely usable in a text-only form.

You should create text-only documents when access speed is critical. If you know that many users will be vying for your pages, you should accommodate them by avoiding the use of images within your documents. In some extreme cases, you might provide a home (leading) page that lets readers decide between duplicate collections of your work: one containing the images, and another stripped of them. (The popular browsers include special picture icons as place holders for yet-to-be downloaded images, which can trash and muddle your document's layout into an unreadable mess.)

Text is most appropriate - supporting images only, without frills or nonessential graphics - if your documents are to be readily searchable by any of the many web indexing services. Images are almost always ignored by these search engines. If the major content of your pages is provided with images, very little information about your documents will find its way into the online web directories.

5.2.4 Speeding Image Downloads

There are several ways to ameliorate the overhead and delays inherent with images, besides being very choosy about which to include in your documents:

Keep it simple

> A full-screen, 24-bit color graphic, even when reduced in size by digital compression with one of the standard formats like GIF or JPEG, is still going to be a network bandwidth hog. Acquire and use the various image management tools to optimize image dimensions and number of colors into the fewest number of pixels. Simplify your drawings. Stay away from panoramic photographs. Avoid large empty backgrounds in your images, as well as gratuitous borders and other space-consuming elements. Also avoid dithering (blending two colors among adjacent pixels to achieve a third color); this technique can significantly reduce the compressibility of your images. Strive for large areas of uniform colors, which compress readily in both GIF and JPEG format.

Reuse images

> This is particularly true for icons and GIF animations. Most browsers cache incoming document components in local storage for the very purpose of quick, network connection-less retrieval of data. For smaller GIF animation files, try to prepare each successive image to update only portions that change in the animation, rather than redraw the entire image (this speeds up the animation, too).

Divide up large documents

This is a general rule that includes images. Many small document segments, organized through hyperlinks (of course!) and effective tables of contents tend to be better accepted by users than a few large documents. In general, people would rather "flip" several pages than dawdle waiting for a large one to download. (It's related to the TV channel-surfing syndrome.) One accepted rule of thumb is to keep your documents under 50 kilobytes each, so even the slowest connections won't overly frustrate your readers.

Isolate necessarily large graphics

Provide a special link to large images, perhaps one that includes a thumbnail of the graphic, thereby letting readers decide if and when they want to spend the time downloading the full image. And since the downloaded image isn't mixed with other document components like inline images, it's much easier for the reader to identify and save the image on their system's local storage for later study. (For details on non-inline image downloads, see Section 5.6.2.)

Specify image dimensions

Finally, another way to improve performance is by including the image's rectangular height and width information in its tag. By supplying those dimensions, you eliminate the extra steps the extended browsers must take to download, examine, and calculate an image's space in the document. There is a downside to this approach, however, that we explore in Section 5.2.6.12.

5.2.5 JPEG or GIF?

You may choose to use only JPEG or GIF images in your HTML documents if your sources for images or your software toolset prefers one over the other format. Both are nearly universally supported by today's browsers, so there shouldn't be any user-viewing problems.

Nevertheless, we recommend that you acquire the facilities to create and convert to both formats to take advantage of their unique capabilities. For instance, use GIF's transparency feature for icons and dingbats. Alternatively, use JPEG for large and colorful images for faster downloading.

5.2.6 The <img> Tag

The `<img>` tag lets you reference and insert a graphic image into the current text flow of your document. There is no implied line or paragraph break before or after the `<img>` tag, so images can be truly "inline" with text and other content.

The format of the image itself is not defined by the HTML or XHTML standard, although the popular graphical browsers support GIF and JPEG images. The standards don't specify or restrict the size or dimensions of the image, either. Images may have any number of colors, but how those colors are rendered is highly browser-dependent.

Image presentation in general is very browser-specific. Images may be ignored by nongraphical browsers. Browsers operating in a constrained environment may modify the image size or complexity. And users, particularly those with slow network connections, may choose to defer image loading altogether. Accordingly, you should make sure your documents make sense and are useful, even if the images are completely removed.

5.2.6.1 The src attribute

The `src` attribute for the `<img>` tag is required (unless you use `dynsrc` with Internet Explorer-based movies; see section 5.2.7.1). Its value is the image file's URL, either absolute or relative to the document referencing the image. To unclutter their document storage, authors typically collect image files into a separate folder they often name something like "pics" or "images." Section 6.2

For example, this HTML fragment places an image of a famous kumquat packing plant into the narrative text (see Figure 5-8):

```
Here we are, on day 17 of the tour, in front of the kumquat
packing plant:
<p>
<img src="pics/packing_plant.gif">
<p>
What an exciting moment, to see the boxes of fruit moving
```

In the example, the paragraph (`<p>`) tags surrounding the `<img>` tag cause the browser to render the image by itself with some vertical space after the preceding text and before the trailing text. Text may also abut the image, as we describe in Section 5.2.6.4.

Function:

Inserts an image into a document

Attributes:

ALIGN	ONDBLCLICK
ALT	ONERROR
BORDER	ONKEYDOWN
CLASS	ONKEYPRESS
CONTROLS I	ONKEYUP
DIR	ONLOAD
DYNSRC I	ONMOUSEDOWN
HEIGHT	ONMOUSEMOVE
HSPACE	ONMOUSEOUT
ID	ONMOUSEOVER
ISMAP	ONMOUSEUP
LANG	SRC
LONGDESC	START I
LOOP I	STYLE
LOWSRC N	TITLE
NAME N	USEMAP
ONABORT	VSPACE
ONCLICK	WIDTH

End tag:

None in HTML; </img> or with XHTML

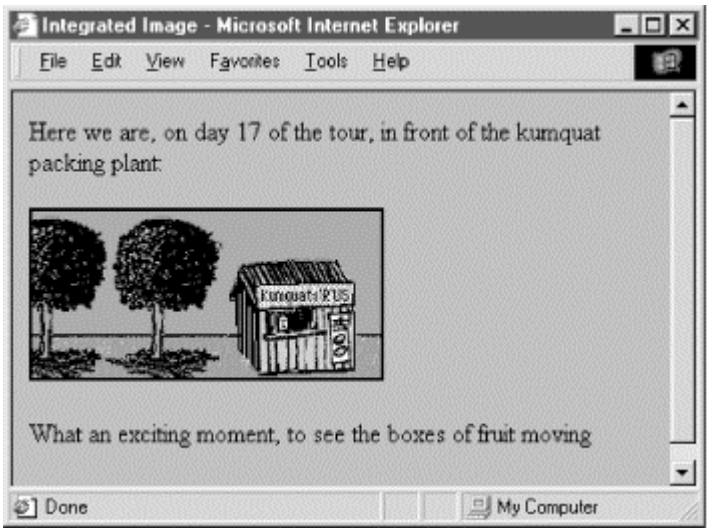
Contains:

Nothing

Used in:

text

Figure 5-8. Image integrated with text



5.2.6.2 The lowsrc attribute

To the benefit of users, particularly those with slow Internet connections, Netscape provides the `lowsrc` companion to the `src` attribute in the `<img>` tag as a way to speed up document rendering. The `lowsrc` attribute's value, like `src`, is the URL of an image file that the browser loads and displays when it first encounters the `<img>` tag. When the document has been completely loaded and can be read by the user, Netscape retrieves the image specified by the `src` attribute.

The `lowsrc` image is a low-resolution, abbreviated version of the final `src` image that loads faster by comparison to quickly give the reader an idea of its content until the final, higher-resolution image eventually replaces it onscreen. But the `lowsrc` attribute can also be used for some very special effects.

Netscape uses the `lowsrc` image's dimensions to reserve space in the document for both the `lowsrc` and `src` images, unless you explicitly allocate that space with the `height` and `width` attributes described later in this chapter. Hence, if the dimensions of the image specified in the `src` attribute are different than those for the `lowsrc` image or your explicitly included height and width values, the `src` image will be reduced, enlarged, stretched, or compressed to fit in the allotted space. Moreover, the `lowsrc` and `src` images needn't be identical, so you might take advantage of the delayed rendering of the `src` image for simple animation.

The `lowsrc` attribute is for Netscape only. Other browsers ignore it and only load the image specified by the `src` attribute. Netscape won't load either image if the user chooses not to auto-load images. In that case, both images will load in order when the user clicks the images button or clicks the image icon placeholder. No browser loads the `lowsrc` image only; you must include a `src` image, otherwise nothing will appear except the missing image icon.

5.2.6.3 The alt and longdesc attributes

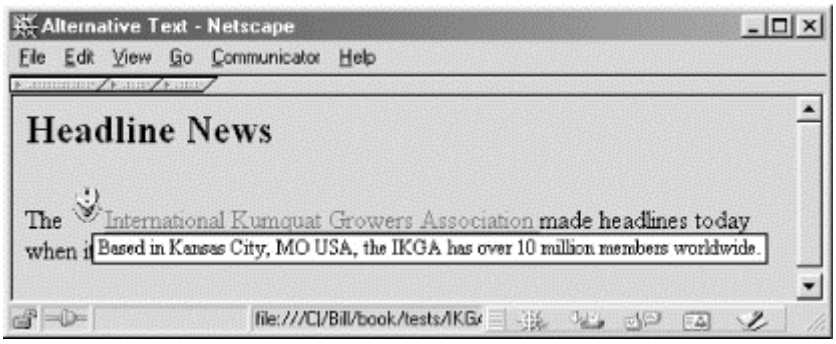
The `alt` attribute specifies alternative text the browser may show if image display is not possible or disabled by the user. It's an option, but one we highly recommend you exercise for most images in your document. This way, if the image is not available, the user still has some indication of what it is that's missing.

In addition, the latest browsers display the alternative description in a text box when users pass their mouse over the image. Accordingly, you might embed short, parenthetical information that pops up when users pass over a small, inline icon, such as shown in Figure 5-9.

The value for the `alt` attribute is a text string of up to 1024 characters if you include spaces or other punctuation. The string must be enclosed in quotation marks. The alternative text may contain entity references to special characters, but it may not contain any other sort of markup; in particular, no style tags are allowed.

Graphical browsers don't normally display the `alt` attribute if the image is available and the user has enabled picture downloading. Otherwise, they insert the `alt` attribute's text as a label next to an image placeholder icon. Well-chosen `alt` labels thereby additionally support those users with a graphical browser who have disabled their automatic image download because of a slow connection to the Web.

Figure 5-9. Contemporary graphical browers display `alt` in a temporary pop-up window



Nongraphical, text-only browsers like Lynx put the `alt` text directly into the content flow just like any other text element. So, when used effectively, the `alt` tag sometimes can transparently substitute for missing images. (Your text-only browser users will appreciate not being constantly reminded of their second-class web citizenship.) For example, consider using an asterisk as the `alt` attribute alternative to a special bullet icon:

```
<h3><img src="pics/fancy_bullet.gif" alt="*">Introduction</h3>
```

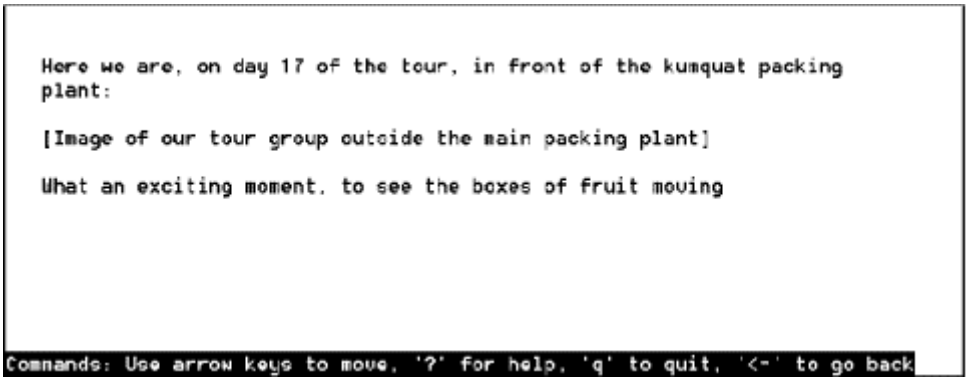
A graphical browser displays the bullet image, while in a nongraphical browser the `alt` asterisk takes the place of the missing bullet. Similarly, use `alt` text to replace special image bullets for list items. For example, the following code:

```
<ul>
  <li> Kumquat recipes <img src="pics/new.gif" alt="(New!)">
  <li> Annual harvest dates
</ul>
```

displays the `new.gif` image with graphical browsers, and the text "(New!)" with text-only browsers. The `alt` attribute uses even more complex text (see Figure 5-10):

```
Here we are, on day 17 of the tour, in front of the kumquat
packing plant:
<p>
<img src="pics/packing_plant.gif"
  alt="[Image of our tour group outside the main packing plant]">
<p>
What an exciting moment, to see the boxes of fruit moving
```

Figure 5-10. Text-only browsers like Lynx display an image's alt attribute text



The `longdesc` attribute is similar to the `alt` attribute, but allows for larger descriptions. The value of `longdesc` is the URL of a document containing a description of the image. If you have a description longer than 1024 characters, use the `longdesc` attribute to link to it. Neither HTML 4 nor XHTML specify what the content of the description must be, nor do any browsers currently implement `longdesc`; all bets are off when deciding how to create those long descriptions.

5.2.6.4 The align attribute

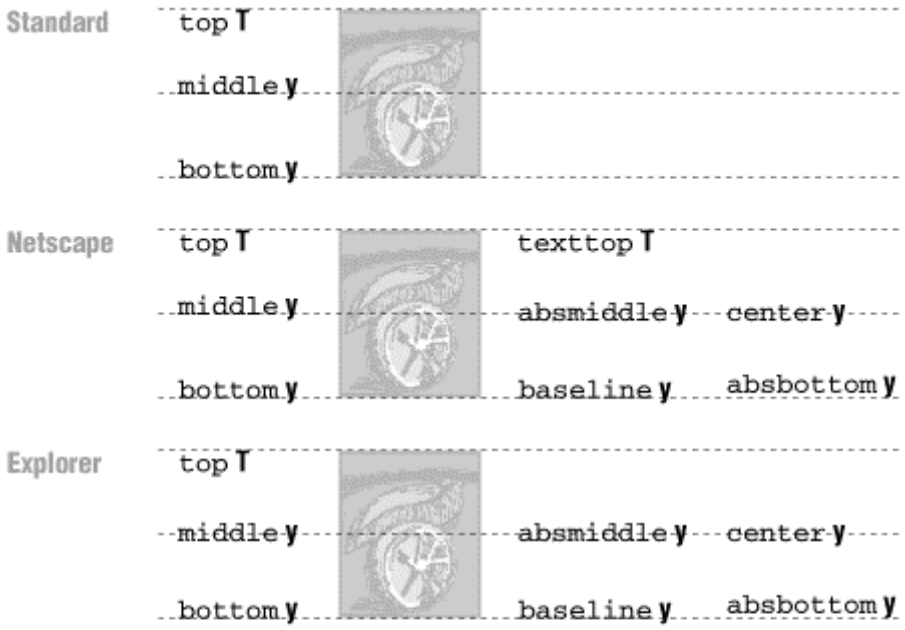
The standards don't define a default alignment for images with respect to other text and images in the same line of text: you can't always predict how the text and images will look.[5] HTML images normally appear in line with a single line of text. Common print media like magazines wrap text around images, with several lines next to and abutting the image, not just a single line.

[5] Most of the popular graphical browsers insert an image so its base aligns with the baseline of the text - the same alignment specified by the attribute value of `bottom`. But document designers should assume that alignment varies between browsers and always include the desired type of image alignment.

Fortunately, document designers can exert some control over the alignment of images with the surrounding text through the `align` attribute for the `<img>` tag. The HTML and XHTML standards specify five image-alignment attribute values: `left`, `right`, `top`, `middle`, and `bottom`. The `left` and `right` values flow any subsequent text around the image, which is moved to the corresponding margin; the remaining three align the image vertically with respect to the surrounding text. Netscape adds four more vertical alignment attributes to that list: `texttop`, `absmiddle`, `baseline`, and `absbottom`, while Internet Explorer adds `center`.

The following list contains descriptions for the inline image alignments; see Figure 5-11 for examples.

Figure 5-11. Standard and browser-extended inline image alignments with text



Alignment	Standard	Netscape	Explorer
top	●	●	●
texttop		●	
middle	●	●	●
absmiddle		●	●
center		●	●
bottom	●	●	●
baseline		●	●
absbottom		●	●

`top`

The top of the image is aligned with the top edge of the tallest item in the current line of text. If there are no other images in the current line, the top of the image is aligned with the top of the text.

`texttop`

The `align=texttop` attribute and value tells Netscape to align the top of the image with the top of the tallest text item in the current line. It is different from the `top` option, which aligns the top of the image with the top of the tallest item, image or text, in the current line. If the line contains no other images that extend above the top of the text, `texttop` and `top` have the same effect.

`absmiddle`

If you set the `align` attribute of the `<img>` tag to `absmiddle`, the browser will fit the absolute middle of the image to the absolute middle of the current line. For Netscape and early versions of Internet Explorer, this is different from the common `middle` option, which aligns the middle of the image with the baseline of the current line of text (the bottom of the characters). Version 3 and later of Internet Explorer, on the other hand, treat `absmiddle` the same as `middle` and `center`.

`center`

The `center` image alignment value gets treated the same as `absmiddle` by both Internet Explorer and Netscape, but note that the browsers treat `absmiddle` and `middle` differently.

`middle`

Netscape and Internet Explorer treat the `middle` image alignment value differently: Netscape aligns the middle of the image to the baseline of the text, regardless of other inline elements, such as another inline image (Figure 5-12). Internet Explorer aligns the middle of the image to the middle of the tallest item in the current line, text or image (Figure 5-13). Notice the alignments and differences in Figure 5-12 and Figure 5-13, particularly when only one image contains the `align` attribute. Both figures display the HTML fragment:

```
Line of text
<img src="pics/horiz.gif" align=middle>
<img src="pics/vert.gif">
goes on ...
<br clear=left>
<p>
Line of text
<img src="pics/horiz.gif" align=middle>
<img src="pics/vert.gif" align=middle>
goes on ...
```

Also note that Internet Explorer Version 3 and later treats `middle`, `absmiddle`, and `center` the same, whereas earlier Internet Explorer versions and Netscape distinguish between `middle` and `absmiddle` alignments. (If you are confused as to exactly what each alignment value means, please raise your hand.)

Figure 5-12. Netscape aligns middle of image to baseline of text

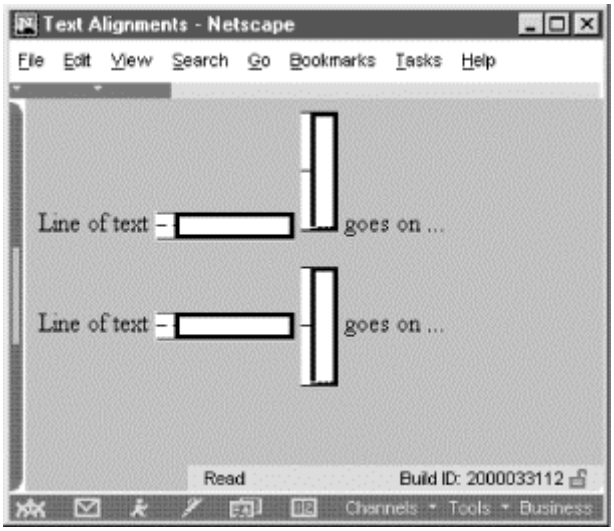
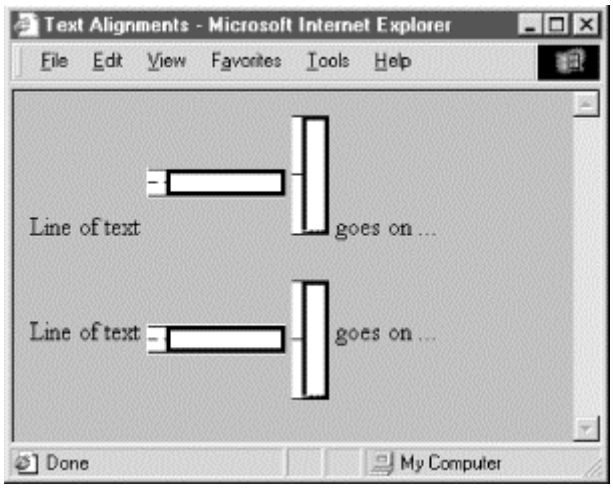


Figure 5-13. Internet Explorer aligns middle of image to middle of tallest line element



`bottom` and `baseline` (default)

With Netscape and early versions of Internet Explorer, the `bottom` and `baseline` image alignment values have the same effect as if you didn't include any alignment attribute at all: the browsers align the bottom of the image in the same horizontal plane as the baseline of the text. This is not to be confused with the `absbottom`, which takes into account letter "descenders" like the tail on the lowercase "y." Internet Explorer Version 3 and later, on the other hand, treat `bottom` the same as `absbottom`. (Did we see a hand up in the audience?)

`absbottom`

The `align=absbottom` attribute tells the browsers to align the bottom of the image with the true bottom of the current line of text. The true bottom is the lowest point in the text taking into account descenders, even if there are no descenders in the line. A descender is the tail on a "y," for example; the baseline of the text is the bottom of the "v" in the "y" character.

Use the `top` or `middle` alignment values for best integration of icons, dingbats, or other special inline effects with the text content. Otherwise, `align=bottom` (the default) usually gives the best appearance. When aligning one or more images on a single line, select the alignment that gives the best overall appearance to your document.

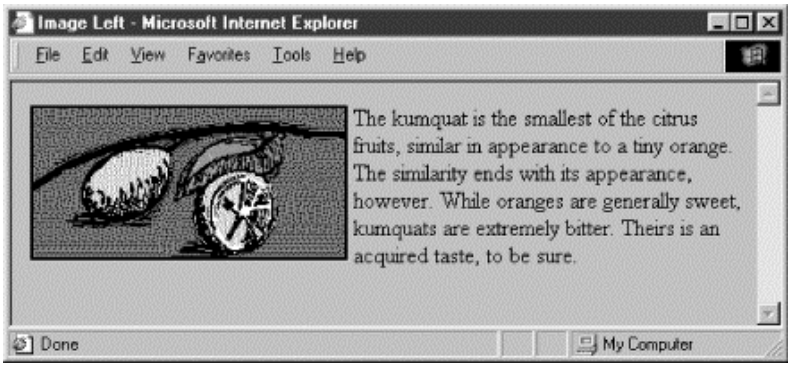
5.2.6.5 Wrapping text around images

The `left` and `right` image alignment values tell the browser to place an image against the left or right margin, respectively, of the current text flow. The browser then renders subsequent document content in the remaining portion of the flow adjacent to the image. The net result is that the document content following the image gets wrapped around the image.

```
<img src="pics/kumquat.gif" align=left>
The kumquat is the smallest of the citrus fruits, similar in appearance to a
tiny orange. The similarity ends with its appearance, however. While oranges
are generally sweet, kumquats are extremely bitter. Theirs is an acquired taste, to be sure.
```

Figure 5-14 shows text flow around a left-aligned image.

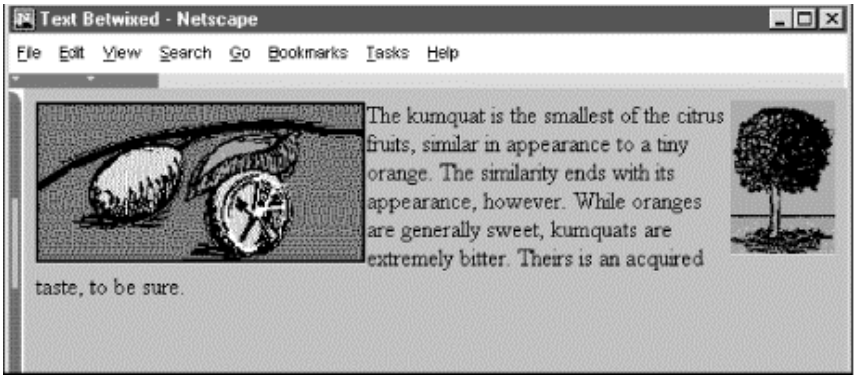
Figure 5-14. Text flow around a left-aligned image



You can place images against both margins simultaneously (Figure 5-15) and the text will run down the middle of the page between them:

```
<img src="pics/kumquat.gif" align=left>
<img src="pics/tree.gif" align=right>
The kumquat is the smallest of the citrus fruits, similar in appearance to a
tiny orange. The similarity ends with its appearance, however. While oranges
are generally sweet, kumquats are extremely bitter. Theirs is an acquired taste, to be sure.
```

Figure 5-15. Running text between left- and right-aligned images

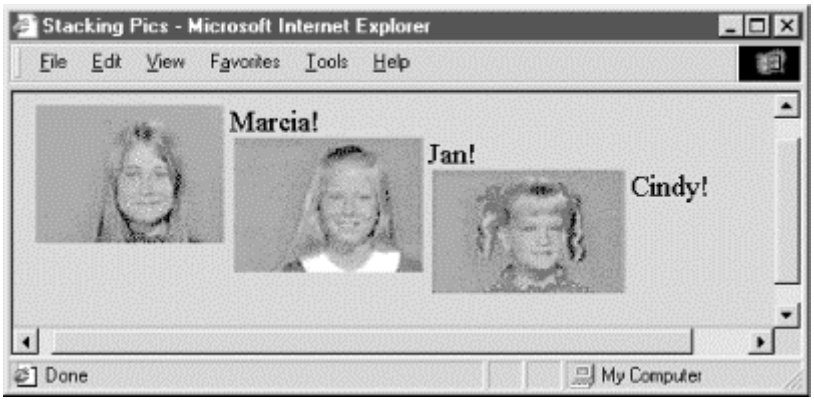


While text is flowing around an image, the left (or right) margin of the page is temporarily redefined to be adjacent to the image as opposed to the edge of the page. Subsequent images with the same alignment will stack up against each other. The following source fragment achieves that staggered image effect:

```
<img src="pics/marcia.gif" align=left>
Marcia!
<br>
<img src="pics/jan.gif" align=left>
Jan!
<br>
<img src="pics/cindy.gif" align=left>
Cindy!
```

The results of this example are shown in Figure 5-16.

Figure 5-16. Three very lovely girls



When the text flows beyond the bottom of the image, the margin returns to its former position, typically at the edge of the browser window.

5.2.6.6 Centering an image

Have you noticed that you can't horizontally center an image in the browser window with the `align` attribute? The `middle` and `absmiddle` values center the image vertically with the current line, but the image is horizontally justified depending on what content comes before it in the current flow and the dimensions of the browser window.

You can horizontally center an inline image in the browser window, but only if it's isolated from surrounding content, such as by paragraph, division, or line break tags. Then, either use the `<center>` tag, or use the `align=center` attribute or center-justified style in the paragraph or division tag to center the image. For example:

```
Kumquats are tasty treats
<br>
<center>
<img src="pics/kumquat.gif">
</center>
that everyone should strive to eat!
```

Use the paragraph tag with its `align=center` attribute if you want some extra space above and below the centered image:

```
Kumquats are tasty treats
<p align=center>
<img src="pics/kumquat.gif">
</p>
that everyone should strive to eat!
```

5.2.6.7 Align is deprecated

The HTML 4 and XHTML standards have deprecated the `align` attribute for all tags, including `<img>`, in deference to style sheets. Nonetheless, the attribute is very popular among HTML authors and remains well supported by the popular browsers. So, while we do expect that someday `align` will disappear, it won't be anytime soon. Just don't say we didn't warn you.

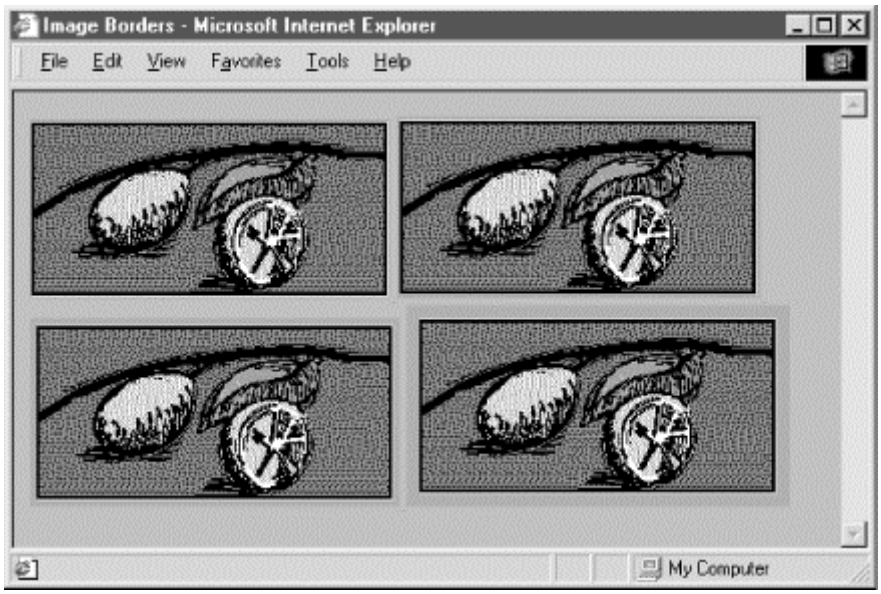
5.2.6.8 The border attribute

Browsers normally render images that also are hyperlinks (an image included in an `<a>` tag) with a two-pixel-wide colored border, indicating to the reader that the image can be selected to visit the associated document. Use the `border` attribute and a pixel-width thickness value to remove (`border=0`) or widen that image border. Be aware that this attribute, too, is deprecated in HTML 4 and XHTML in lieu of style sheets, but continues to be well supported by the popular browsers.

Figure 5-17 shows you the thick and thin of image borders, as rendered by Internet Explorer from the following XHTML source:

```
<a href="test.html">
 <img src="pics/kumquat.gif" border="1" />
</a>
<a href="test.html">
  <img src="pics/kumquat.gif" border="2" />
</a>
<a href="test.html">
  <img src="pics/kumquat.gif" border="4" />
</a>
<a href="test.html">
  <img src="pics/kumquat.gif" border="8" />
</a>
```

Figure 5-17. The thick and thin of image borders



5.2.6.9 Removing the image border

You can eliminate the border around an image hyperlink altogether with the `border=0` attribute within the `<img>` tag. For some images, particularly image maps, the absence of a border can improve the appearance of your pages. Images that are clearly link buttons to other pages may also look best without a border.

Be careful, though, that by removing the border, you don't diminish your page's usability. No border means you've removed a common visual indicator of a link, making it less easy for your readers to find the links on the page. Browsers will change the mouse cursor as readers pass it over an image that is a hyperlink, but you should not assume they will, nor should you make readers test your borderless images to find hidden links.

We strongly recommend that you use some additional way with borderless images to let your readers know to click the images. Even including simple text instructions will go a long way to making your pages more accessible to readers.

5.2.6.10 The height and width attributes

Ever watch the display of a page's contents shift around erratically while the document is loading? That happens because the browser readjusts the page layout to accommodate each loaded image. The browser determines the size of an image and, hence, the rectangular space to reserve for it in the display window, by retrieving the image file and extracting its embedded height and width specifications. The browser then adjusts the page's display layout to insert that picture in the display.[6] This is not the most efficient way to render a document, since the browser must sequentially examine each image file and calculate its screen space before rendering adjacent and subsequent document content. That can significantly increase the amount of time it takes to render the document and disrupt reading by the user.

[6] Another reminder that images are separate files, which are loaded individually and in addition to the source document.

A more efficient way for authors to specify an image's dimensions is with the `height` and `width` `<img>` attributes. That way, the browser can reserve space before actually downloading an image, speeding document rendering and eliminating the content shifting. Both attributes require an integer value that indicates the image size in pixels; and the order in which they appear in the `<img>` tag is not important.

5.2.6.11 Resizing and flood-filling images

A hidden feature of the `height` and `width` attributes is that you don't need to specify the actual image dimensions; the attribute values can be larger or smaller than the actual size of the image. The browser automatically scales the image to fit the predefined space. This gives you a down-and-dirty way of creating thumbnail versions of large images and a way to enlarge very small pictures. Be careful, though: the browser still must download the entire file, no matter what its final rendered size is, and you will distort an image if you don't retain its original height versus width proportions.

Another trick with `height` and `width` provides an easy way to flood-fill areas of your page and can also improve document performance. Suppose you want to insert a colored bar across your document.[7]

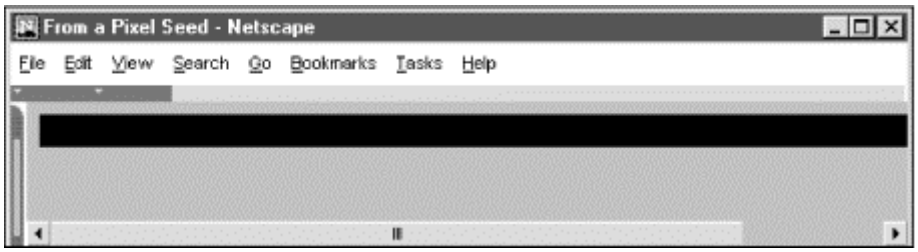
[7] This is one way to create colored horizontal rules in Netscape 3 or earlier versions, which don't support the `color` attribute of the `<hr>` tag.

Rather than create an image to the full dimensions, create one that is just one pixel high and wide and set it to the desired color. Then use the `height` and `width` attributes to scale it to the larger size:

```
<img src="pics/one-pixel.gif" width=640 height=20>
```

The smaller image downloads much faster than a full-scale image, and the `width` and `height` attributes create the desired bar after the tiny image arrives at the browser (see Figure 5-18).

Figure 5-18. This bar was made from a one-pixel image



One last trick with the `width` attribute is to use a percentage value instead of an absolute pixel value. This causes the browser to scale the image to a percentage of the document window width. Thus, to create a colored bar 20 pixels high and the width of the window, you could use:

```
<img src="pics/one-pixel.gif" width="100%" height=20>
```

As the document window changes size, the image will change size as well.

If you provide a percentage `width` and omit the `height`, the browser will retain the image's aspect ratio as it grows and shrinks. This means that the height will always be in the correct proportion to the width and the image will display without distortion.

5.2.6.12 Problems with height and width

Although the `height` and `width` attributes for the `<img>` tag can improve performance and let you perform neat tricks, there is a knotty downside to using them. The browser sets aside the specified rectangle of space to the prescribed dimensions in the display window even if the user has turned off automatic download of images. What the user often is left with is a page full of semi-empty frames with meaningless picture placeholder icons inside. The page looks terribly unfinished and is mostly useless. Without accompanying dimensions, on the other hand, the browser simply inserts a placeholder icon inline with the surrounding text, so at least there's something there to read in the display.

We don't have an answer to this dilemma, other than to insist that you use the `alt` attribute with some descriptive text so that users at least know what they are missing (see Section 5.2.6.3). We do recommend that you include these size attributes because we encourage any practice that improves network performance.

5.2.6.13 The hspace and vspace attributes

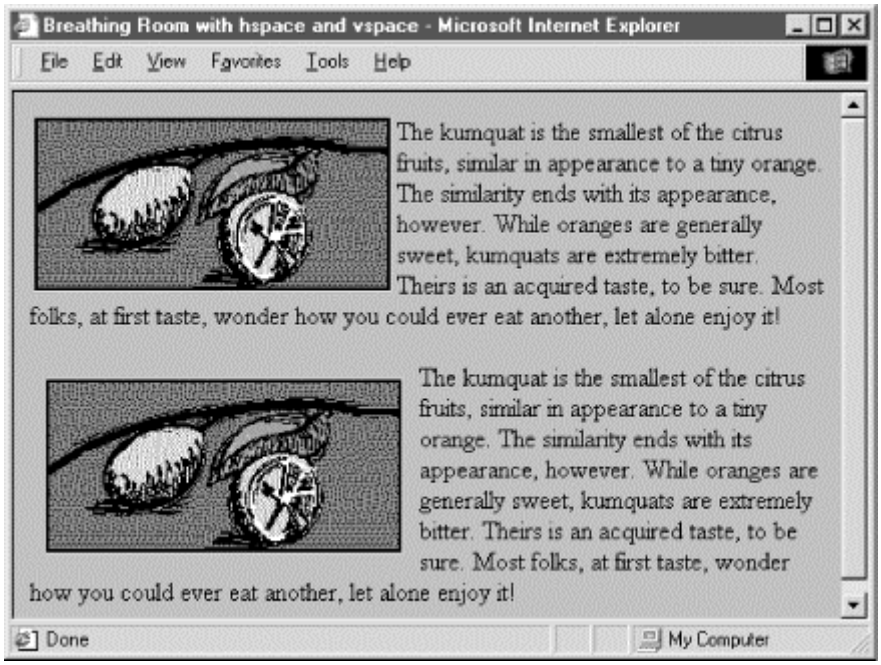
Graphical browsers usually don't give you much space between an image and the text around it. And unless you create a transparent image border that expands the space between them, the typical two-pixel buffer between an image and adjacent text is just too close for most designers' comfort. Add the image into a hyperlink, and the special colored border will negate any transparent buffer space you labored to create, as well as draw even more attention to how close the adjacent text butts up against the image.

The hspace and vspace attributes can give your images breathing room. With hspace, you specify the number of pixels of extra space to leave between the image and text on the left and right sides of the image; the vspace value is the number of pixels on the top and bottom:

```
<img src="pics/kumquat.gif" align=left>
The kumquat is the smallest of the citrus fruits, similar
in appearance to a tiny orange. The similarity ends with its
appearance, however. While oranges are generally sweet,
kumquats are extremely bitter. Theirs is an acquired taste,
to be sure. Most folks, at first taste, wonder how you could
ever eat another, let alone enjoy it!
<p>
<img src="pics/kumquat.gif" align=left hspace=10 vspace=10>
The kumquat is the smallest of the citrus fruits, similar
in appearance to a tiny orange. The similarity ends with its
appearance, however. While oranges are generally sweet,
kumquats are extremely bitter. Theirs is an acquired taste,
to be sure. Most folks, at first taste, wonder how you could
ever eat another, let alone enjoy it!
```

Figure 5-19 shows the difference between two wrapped images.

Figure 5-19. Improve image/text interfaces with vspace and hspace extensions



We're sure you'll agree that the additional space around the image makes the text easier to read and the overall page more attractive.

5.2.6.14 The ismap and usemap attributes

The ismap and usemap attributes for the <img> tag tell the browser that the image is a special mouse-selectable visual map of one or more hyperlinks, commonly known as an *image map*. The ismap style of image maps, known as a *server-side* image map, may be specified only within an <a> tag hyperlink. Section 6.3.1

For example:

```
<a href="/cgi-bin/images/map2">
  <img src="pics/map2.gif" ismap>
</a>
```

The browser automatically sends the x,y position of the mouse (relative to the upper-left corner of the image) to the server when the user clicks somewhere on the ismap image. Special server software (the */cgi-bin/images/map2* program in the example) may then use those coordinates to determine a response.

The usemap attribute provides a *client-side* image map mechanism that effectively eliminates server-side processing of the mouse coordinates and its incumbent network delays and problems. Using special <map> and <area> tags, HTML authors provide a map of coordinates for the hyperlink-sensitive regions in the usemap image along with related hyperlink URLs. The value of the usemap attribute is a URL that points to that special <map> section. The browser on the user's computer translates the coordinates of a click of the mouse on the image into some action, including loading and displaying another document. Section 6.5.3 / Section 6.5.4

For example, the following source specially encodes the 100 x 100-pixel map2.gif image into four segments, each of which, if clicked by the user, links to a different document. Notice we've included, validly, the ismap image map processing capability in the example <img> tag so that users of other, usemap-incapable browsers have access to the alternative, server-side mechanism to process the image map:

```
<a href="/cgi-bin/images/map2">
  <img src="pics/map2.gif" ismap usemap="#map2">
</a>
...
<map name="map2">
  <area coords="0,0,49,49" href="link1.html">
  <area coords="50,0,99,49" href="link2.html">
  <area coords="0,50,49,99" href="link3.html">
  <area coords="50,50,99,99" href="link4.html">
</map>
```

Geographical maps make excellent ismap and usemap examples: browsing a nationwide company's pages, for instance, the users might click on their home towns on a map to get the addresses and phone numbers for nearby retail outlets. The advantage of the usemap client-side image map processing is that it does not require a server or special server software and so, unlike the ismap mechanism, can be used in non-web (networkless) environments, such as local files or CD-ROM.

Please read our more complete discussion of anchors and links, including image maps within links, in Section 6.5.

5.2.6.15 The class, dir, event, id, lang, style, and title attributes

Several nearly universal attributes give you a common way to identify (title) and label (id) the image tag's contents for later reference or automated treatment, to change the contents' display characteristics (class, style), and to reference the language (lang) used and related direction the text should flow (dir). And, of course, there are all the user events that may happen in and around the tagged contents that the browser senses and that you may react to via an on-event attribute and some programming. Section 8.1.1 / Section 8.3

Of these many HTML 4 and XHTML attributes, id is the most important. It lets you label the image for later access by a program or browser operation (see Chapter 12). Section 4.1.1.4

The remaining attributes have questionable meaning in context with <img>. Granted, there are a few style sheet options available that may influence an image's display, and a title is good to include, although alt is better. And it's hard to imagine what the influence of language (lang) or its presentation direction (dir) might have on an image. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.5

5.2.6.16 The name, onAbort, onError, onLoad and other event attributes

There are four <img> attributes currently supported by Netscape that enable you to use JavaScript to manipulate the image. The first is the name attribute. Now redundant with the standard id attribute,[8] name lets you label the image so that it can be referenced by a JavaScript applet.

> [8] HTML Version 4.01 adopts the name attribute into the standard, even though only Netscape currently supports it with <img>.

For example:

```
<img src="pics/kumquat.gif" name="kumquat">
```

lets you later refer to that picture of a kumquat as simply "kumquat" in a JavaScript applet perhaps to erase or otherwise modify it. You cannot individually manipulate an image with JavaScript if it is not named or doesn't have an associated id.

The other three attributes let you provide some special JavaScript event handlers. The value of the attribute is a chunk of JavaScript code, enclosed in quotation marks; it may consist of one or more JavaScript expressions, separated by semicolons.

Netscape invokes the onAbort event handler if the user stops loading an image, usually by clicking the browser's "stop" button. You might, for instance, use an onAbort message to warn users if they stop loading some essential image, such as an image map (see Section 6.5):

```
<img src="pics/kumquat.gif" usemap="#map1"
onAbort="window.alert('Caution: This image contains important hyperlinks.  Please load the
entire image.')">
```

The onError attribute is invoked if some error occurs during the loading of the image, but not for a missing image or one that the user chose to stop loading. Presumably, the applet could attempt to recover from the error or load a different image in its place. Netscape executes the JavaScript code associated with the <img> tag's onLoad attribute right after the browser successfully loads and displays the image.

See Section 13.3.3 for more information about JavaScript and event handlers.

5.2.6.17 Combining <img> attributes

You may combine any of the various standard and extension attributes for images where and when they make sense. The order for inclusion of multiple attributes in the `<img>` tag is not important, either. Just be careful not to use redundant attributes or you won't be able to predict the outcome.

5.2.7 Video Extensions

The special `controls`, `dynsrc`, `loop`, and `start` attribute extensions for the `<img>` tag are unique to Internet Explorer and are not HTML 4 or XHTML standard attributes. They let you embed an inline movie into the body content, just like an image.

Equivalent behavior is available in Netscape via an extension program known as a plug-in. Plug-ins place an additional burden on the user, in that each user must find and install the appropriate plug-in before being able to view the inline video. The Internet Explorer `<img>` tag extensions, on the other hand, make video display an intrinsic part of the browser. Section 12.2

However, the Internet Explorer movie extensions currently are very limited. They are not supported by any other browser and can be used only with Audio Video Interleave (AVI) formatted movie files, since that's the player format built into Internet Explorer and enabled through Microsoft's Windows operating system software. Moreover, recent innovations in browser technology, objects, and applets in particular may make Internet Explorer's approach of extending the already overloaded `<img>` tag obsolete.

5.2.7.1 The dynsrc attribute

Use the `dynsrc` attribute extension in the `<img>` tag to reference an AVI movie for inline display by Internet Explorer. Its required value is the URL of the movie file enclosed in quotation marks. For example, this text displays the tag and attribute for an AVI movie file entitled *intro.avi*:

```
<img dynsrc="movies/intro.avi">
```

The browser sets aside a video viewport in the HTML display window and plays the movie, with audio if it's included in the clip and if your computer is able to play audio. Internet Explorer treats `dynsrc` movies similar to inline images: in line with current body content and according to the dimension of the video frame. And, like common images, the `dynsrc` referenced movie file gets displayed immediately after download from the server. You may change those defaults and add some user controls with other attributes, as described later.

Because all other browsers currently ignore the special Internet Explorer attributes for movies, they may become confused by an `<img>` tag that does not contain the otherwise required `src` attribute and an image URL. We recommend that you include the `src` attribute and a valid image file URL in all `<img>` tags, including those that reference a movie for Internet Explorer users. The other browsers display the still image in place of the movie; Internet Explorer does the reverse and plays the movie, but does not display the image. Note that the order of attributes does not matter. For example:

```
<img dynsrc="movies/intro.avi" src="pics/mvstill.gif">
```

Internet Explorer loads and plays the AVI movie *intro.avi*; other graphical browsers will load and display the *mvstill.gif* image instead.

5.2.7.2 The controls attribute

Normally, Internet Explorer plays a movie inside a framed viewport once, without any visible user controls. Although no longer supported with Internet Explorer version 5 or later, with older versions of the browser the user may restart, stop, and continue the movie by clicking inside that viewport with the mouse. Use the `controls` attribute (no value) to add visible controls to the movie viewport so that the user may, with the mouse, play, fast-forward, reverse, stop, and pause the movie, like on a VCR. If the movie clip includes a soundtrack, Internet Explorer provides an audio volume control as well. For example:

```
<img dynsrc="movies/intro.avi" controls src="pics/mvstill.gif">
```

adds the various playback controls to the video window of the *intro.avi* movie clip, as shown in Figure 5-20.

With Internet Explorer version 5 and later, users stop movie playback with the browser's stop button, and may restart it by right-clicking on the movie image and selecting "Play" from the dialog menu options.

Figure 5-20. The controls attribute added video playback controls to inline movies

5.2.7.3 The loop attribute

Internet Explorer normally plays a movie clip from beginning to end once after download. The `loop` attribute for the movie `<img>` tag lets you have the clip play repeatedly for an integer number of times set by the attribute's value, or forever if the value is `infinite`. The user may still cut the loop short by clicking on the movie image, by pressing the stop button, if given controls (see Section 5.2.7.2), or by moving on to another document.

The following *intro.avi* movie clip will play from beginning to end, then restart at the beginning and play through to the end nine more times:

```
<img dynsrc="movies/intro.avi" loop=10 src="pics/mvstill.gif">
```

Whereas the following movie will play over and over again, incessantly:

```
<img dynsrc="movies/intro.avi" loop=infinite src="pics/mvstill.gif">
```

Looping movies aren't necessarily meant to annoy. Some special effects animations, for instance, are a sequence of repeated frames or segments. Rather than string the redundant segments into one, long movie that extends its download time, simply loop the single, compact segment.

5.2.7.4 The start attribute

Normally, an Internet Explorer movie clip starts playing as soon as it's downloaded. You can modify that behavior with the `start` attribute in the movie's `<img>` tag. By setting its value to `mouseover`, you delay playback until the user passes the mouse pointer over the movie viewport. The other valid `start` attribute value, `fileopen`, is the default: start playback just after download. It is included because both values may be combined in the `start` attribute to cause the movie to play back automatically once after download, and then whenever the user passes the mouse over its viewport. Add a value-separating comma, with no intervening spaces, or else enclose them in quotes, when combining the `start` attribute values.

For example, our by-now-infamous *intro.avi* movie will play once when its host HTML document is loaded by the user, and whenever he or she passes the mouse over the movie's viewport:

```
<img dynsrc="movies/intro.avi" start="fileopen,mouseover" src="pics/mvstill.gif">
```

5.2.7.5 Combining movie <img> attributes

Treat Internet Explorer inline movies as you would any image, mixing and matching the various movie-specific as well as the standard and extended `<img>` tag attributes and values supported by the browser. For example, you might align the movie (or its image alternative, if displayed by another browser) to the right of the browser window:

```
<img dynsrc="movies/intro.avi" src="pics/mvstill.gif" align=right>
```

Combining attributes to achieve a special effect is good. We also recommend you combine attributes to give control to the user, when appropriate. For instance, if you set up a movie to loop incessantly, you should also include the `controls` attribute so the user can stop the movie without having to leave the HTML document.

As we stated in Section 5.2.7.4, by combining attributes you can also delay playback until the user passes the mouse over its viewport. Magically, the movie comes alive and plays continuously:

```
<img dynsrc="movies/magic.avi" start=mouseover
     loop=infinite src="pics/magic.gif">
```

5.3 Document Colors and Background Images

The HTML 4 and XHTML standards provide a number of attributes for the <body> tag that let you define text, link, and document background colors, in addition to defining an image to be used as the document background. Internet Explorer extends these attributes to include document margins and better background image control. And, of course, the latest style sheet technologies integrated into the current browsers let you manipulate all these various display parameters.

5.3.1 Additions and Extensions to the <body> Tag

The attributes that control the document background, text color, and document margins are used with the <body> tag. Section 3.8.1

5.3.1.1 The bgcolor attribute

One standard, although deprecated, way you can change the default background color in the browser window to another hue is with the bgcolor attribute for the <body> tag. Like the color attribute for the <font> tag, the required value of the bgcolor attribute may be expressed in either of two ways: as the red, green, and blue (RGB) components of the desired color or as a standard color name. Appendix G provides a complete discussion of RGB color encoding along with a table of acceptable color names you can use with the bgcolor attribute.

Setting the background color is easy. To get a pure red background using RGB encoding, try:

```
<body bgcolor="#FF0000">
```

For a more subtle background, try:

```
<body bgcolor="peach">
```

5.3.1.2 The background attribute

If a splash of color isn't enough, you may also place an image into the background of a document with the background attribute in its <body> tag.

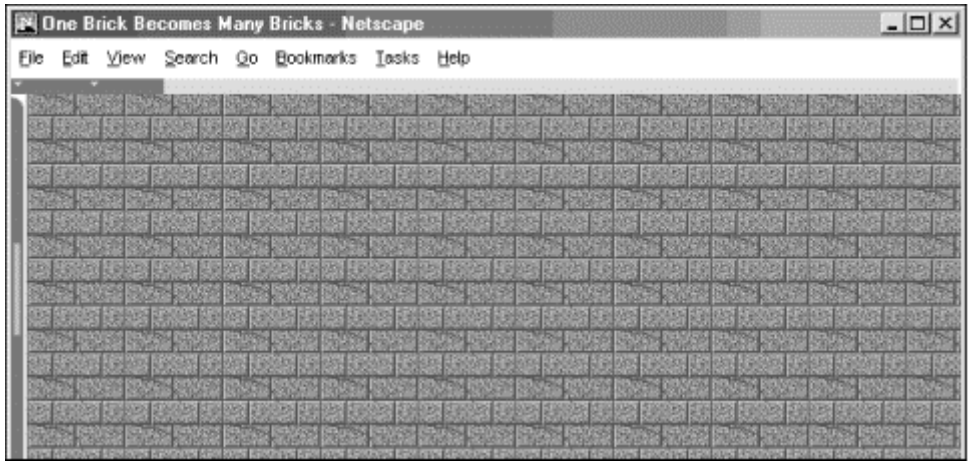
The required value of the background attribute is the URL of an image. The browser automatically repeats (tiles) the image both horizontally and vertically to fill the entire window.

You normally should choose a small, somewhat dim image to create an interesting but unobtrusive background pattern. Besides, a small, simple image traverses the network much faster than an intricate, full-screen image.

Figure 5-21 shows you how the extended browsers render a single brick to create a wall of bricks for the document background:

```
<body background="pics/onebrick.gif">
```

Figure 5-21. One brick becomes many in a Netscape background

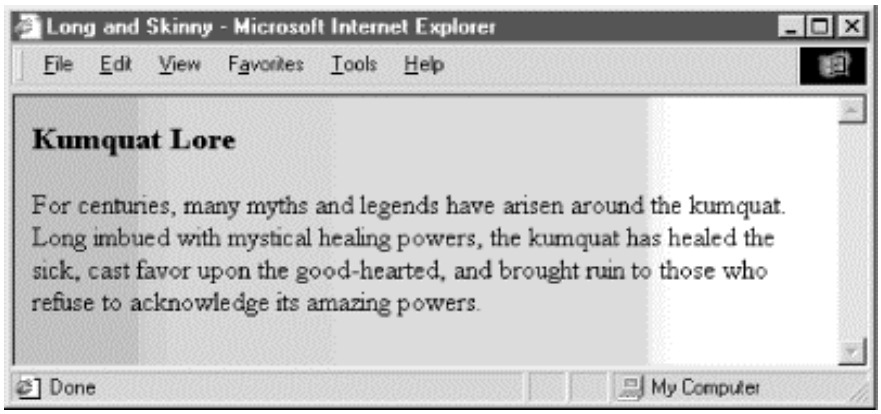


Background images of various dimensions and sizes create interesting vertical and horizontal effects on the page. For instance, a tall skinny image might set off your document heading:

```
<body background="pics/vertical_fountain.gif">
<h3>Kumquat Lore</h3>
For centuries, many myths and legends have arisen around the kumquat.
...
```

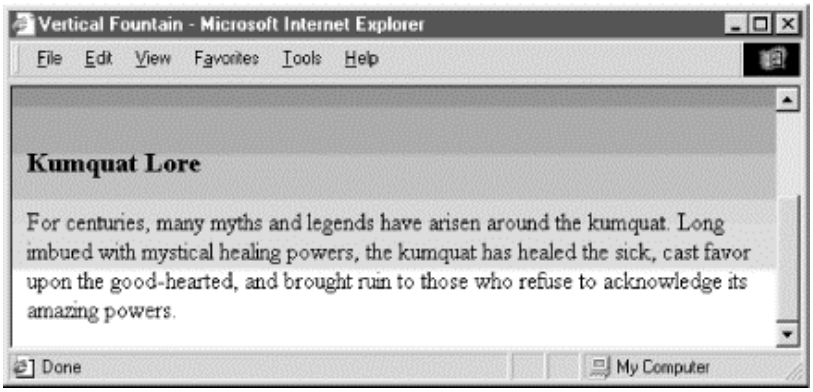
If the *vertical_fountain.gif* is a narrow, tall image whose color grows lighter towards its base and whose length exceeds the length of the document body, the resulting document might look like the one shown in Figure 5-22.

Figure 5-22. A tall and skinny background



You can achieve a similar effect horizontally with an image that is much wider than it is long (see Figure 5-23).

Figure 5-23. A long and skinny background



The `background` attribute is deprecated in HTML 4 and XHTML, since you can achieve similar effects using style sheets.

5.3.1.3 The bgproperties attribute

The `bgproperties` attribute extension for the `<body>` tag is exclusive to Internet Explorer and works only in conjunction with the `background` attribute extension. The `bgproperties` attribute has a single value, `fixed`. It freezes the background image to the browser window, so it does not scroll with the other window contents. Hence, the example *H2Omark.gif* background image serves as a watermark for the document:

```
<body background="pics/H2Omark.gif" bgproperties="fixed">
```

5.3.1.4 The text attribute

Once you alter a document's background color or add a background image, you also might need to adjust the text color to ensure that users can read the text. The HTML 4/XHTML `text` standard attribute for the `<body>` tag does just that: it sets the color of all nonanchor text in the entire document.

Give the `text` attribute a color value in the same format as you use to specify a background color (see `bgcolor` in Section 6.3.1.1) - an RGB triplet or color name, as described in Appendix G. For example, to produce a document with blue text on a pale yellow background, use:

```
<body bgcolor="#777700" text="blue">
```

Of course, it's best to select a text color that contrasts well with your background color or image.

The `text` attribute is deprecated in HTML 4 and XHTML, since you can achieve similar effects using style sheets.

5.3.1.5 The link, vlink, and alink attributes

The `link`, `vlink`, and `alink` attributes of the `<body>` tag control the color of hypertext (`<a>` tag) in your documents. All three accept values that specify a color as an RGB triplet or color name, just like the `text` and `bgcolor` attributes.

The `link` attribute determines the color of all hyperlinks the user has not yet followed. The `vlink` attribute sets the color of all links the user had followed at one time or another. The `alink` attribute defines a color for active link text - one that is currently selected by the user and is under the mouse cursor with the mouse button depressed.

Like text color, you should be careful to select link colors that can be read against the document background. Moreover, the link colors should be different from the regular text as well as from each other.

These attributes are deprecated in HTML 4 and XHTML, since you can achieve similar effects using style sheets.

5.3.1.6 The leftmargin attribute

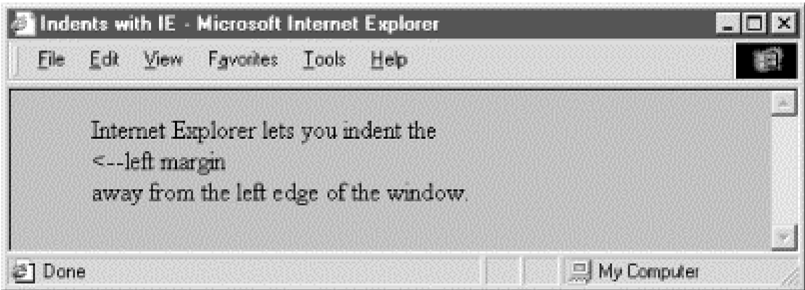
Peculiar to Internet Explorer, the `leftmargin` attribute extension for the `<body>` tag lets you indent the left margin relative to the left edge of the browser's window, much like a margin on a sheet of paper. Other browsers ignore this attribute and normally left-justified body content abuts the left edge of the document window.

The value of the `leftmargin` attribute is the integer number of pixels for that left-margin indent; a value of is the default. The margin is filled with the background color or image.

For example, Internet Explorer renders the following text justified against a margin 50 pixels away from the left edge of the browser window (see Figure 5-24):

```
<body leftmargin=50>
Internet Explorer lets you indent the<br>
&lt;--left margin<br>
away from the left edge of the window.
</body>
```

Figure 5-24. Internet Explorer's leftmargin attribute for indenting body content



5.3.1.7 The topmargin attribute

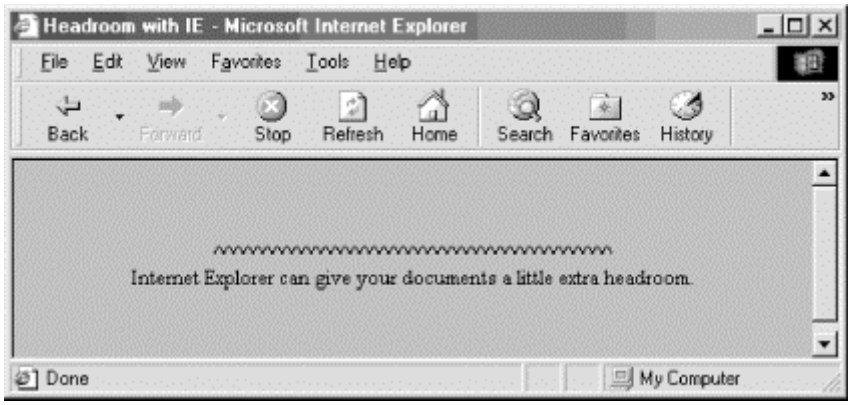
Like `leftmargin`, the `topmargin` attribute extension currently is exclusive to Internet Explorer. It may be included in the `<body>` tag to set a margin of space at the top of the document. The margin space is filled with the document's background color or image.

Body content begins flowing below the integer number of pixels you specify as the value for `topmargin`; a value of is the default.

For example, Internet Explorer renders the following text at least 50 pixels down from the top edge of the browser window (see Figure 5-25):

```
<body topmargin=50>
<p align=center>
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Internet Explorer can give your documents
a little extra headroom.
</p>
</body>
```

Figure 5-25. Internet Explorer's topmargin attribute for lowering body content



5.3.1.8 The style and class attributes

You also can set all the various style-related `<body>` features and then some with Cascading Style Sheets. But although you may include the `style` attribute with the `<body>` tag to create an inline style for the entire body content, we recommend that you set the styles for the entire document body at the document level (`<style>` tag inside the document head) or via a collection-level (imported) style sheet. Use the `class` attribute and name value to apply the appropriate style of a predefined class of the `<body>` tag to the contents. (Since there can only be one body per document, what is the point of setting a class name otherwise?) We cover the use of `style` and class definitions in Chapter 8.

5.3.1.9 Mixing and matching body attributes

Although `background` and `bgcolor` attributes can appear in the same `<body>` tag, a background image will effectively hide the selected background color unless the image contains substantial portions of transparent areas, as we described earlier in this chapter. But even if the image does hide the background color, go ahead and include the `bgcolor` attribute and some appropriate color value. That's because users can turn off image downloading, which includes background images, and so they may find your page otherwise left naked and unappealing. Moreover, without a `bgcolor` attribute or a downloaded (for whatever reason) background image, the browsers merrily ignore your text and link color attributes, too, reverting instead to its own default values, or the ones chosen by the user.

5.3.2 Extending a Warning

The various color and image extensions work wonderfully, particularly the colorful ones, assuming that all users have a 256-color display, lots of available memory, unlimited network bandwidth, and good visual acuity. In reality, many users have monochrome or limited color displays, limited memory for caching images, extremely restricted network bandwidth, and poor vision.

Because of these limitations, you should seriously consider not using any of these extensions in your documents. Much like early users of the Macintosh felt compelled to create documents using ransom-note typography ("I've got 40 fonts on this thing, and I'm going to use them all!"), many authors cannot avoid adding some sort of textured background to every document they create ("I've got 13 wood grains and 22 kinds of marbling, and I'm going to use them all!").

In reality, except for the very clever ones, texture-mapped backgrounds add no information to your documents. The value of your document ultimately lies in its text and imagery, not the cheesy blue swirly pattern in the background. No matter how cool it looks, your readers are not benefiting and could be losing readability.

We advise you not to use the color extensions except for comparatively frivolous endeavors or unless the extension really adds to the document's value, such as for business advertising and marketing pages.

5.3.2.1 Problems with background images

Here are some of the things that can go wrong with background images:

- The time to load the document is increased by the amount of time needed to load the image. Until the background image is completely downloaded, no further document rendering is possible.
- The background image takes up room in the browser's local cache, displacing other images that might actually contain useful information. This makes other documents, which might not even have backgrounds, take that much longer to load.
- The colors in the image may not be available on the user's display, forcing the browser to dither the image. This replaces large areas of a single color with repeating patterns of several other closer, but not cleaner, colors and can make the text more difficult to read.
- Because the browser must actually display an image in the background, as opposed to filling an area with a single color, scrolling through the document can take much longer.
- Even if it's clear onscreen, text printed on top of an image invariably is more difficult, if not impossible, to read.
- Fonts vary widely between machines; the ones you use with your browser that work fine with a background pattern often end up jagged and difficult to read on another machine.

5.3.2.2 Problems with background, text, and link colors

There also are a slew of problems you will encounter if you play with background colors, including:

- The color you choose, while just lovely in your eyes, may look terrible to the user. Why annoy them by changing what users most likely have already set as their own default background color?
- While you may be a member of the "light text on a dark background" school of document design, many people also favor the "dark text on a light background" style that has been consistently popular for over three thousand years. Instead of bucking the trend, assume that users have already set their browser to a comfortable color scheme.
- Some users are color-blind. What may be a nifty-looking combination of colors to you may be completely unreadable to others. One combination in particular to avoid is green for unvisited links and red for visited links. Millions of men are afflicted with red/green color blindness.
- Your brilliant hue may not be available on the user's display, and the browser may be forced to choose one that's close instead. For displays with very few colors (like those of several million 16-color VGA Windows-based machines currently in use) the close colors for the text and the background might be the same color!
- For the same reasons above, active, unvisited, and visited links may all wind up as the same color on limited-color displays.
- By changing text colors, particularly those for visited and unvisited links, you may completely confuse the user. By changing those colors, you effectively force them to experiment with your page, clicking a few links here and there to learn your color scheme.
- Most page designers have no formal training in cognitive psychology, fine arts, graphic arts, or industrial design, yet feel fully capable of selecting appropriate colors for their documents. If you must fiddle with the colors, ask a professional to pick them for you.

5.3.2.3 And then again

There is no denying the fact that these extensions result in some very stunning HTML documents. And they are fun to explore and play with. So, rather than leave this section on a sour note of caution, we encourage you to go ahead and play - just play carefully.

5.4 Background Audio

There is one other form of inline multimedia generally available to web surfers - audio. Most browsers treat audio multimedia as separate documents, downloaded and displayed by special helper applications, applets, or plug-ins. Internet Explorer, on the other hand, contains a built-in sound decoder and supports a special tag that lets you integrate an audio file with your document that plays in the background as a soundtrack for your page. Section 12.1 / Section 12.2

We applaud the developers of Internet Explorer for providing a mechanism that more cleanly integrates audio into HTML and XHTML documents. And the possibilities with audio are very enticing. But at the same time, we caution authors that the special tags and attributes for audio don't work with other browsers, and whether this is the method that the majority of browsers will eventually support is not at all assured. So, beware.

5.4.1 The <bgsound> Tag

Use the `<bgsound>` tag to play a soundtrack in the background. This tag is for Internet Explorer documents only. All other browsers ignore the tag. It downloads and plays an audio file when the host document is first downloaded by the user and displayed. The background sound file also will replay whenever the user refreshes the browser display.

<bgsound> 0

Function:

Plays a soundtrack in the document background

Attributes:

LOOP
SRC

End tag:

None in HTML

Contains:

Nothing

Used in:

body_content

5.4.1.1 The src attribute

The `src` attribute is required for the `<bgsound>` tag. Its value references the URL for the related sound file. For example, when the Internet Explorer user first downloads a document containing the tag:

```
<bgsound src="audio/welcome.wav">
```

they will hear the *welcome.wav* audio file - perhaps an inviting message - play once through their computer's sound system.

Currently, Internet Explorer can handle three different sound format files: `wav`, the native format for PCs; `au`, the native format for most Unix workstations; and MIDI, a universal music-encoding scheme (see also Table 5-1).

Table 5-1, Common Multimedia Formats and Respective Filename Extensions

Format	Type	Extension	Platform of Origin
GIF	Image	*gif*	Any
JPEG	Image	*jpg, jpeg, jpe*	Any
XBM	Image	*xbm*	Unix
TIFF	Image	*tif, tiff*	Any
PICT	Image	*pic, pict*	Any
Rasterfile	Image	*ras*	Sun
MPEG	Movie	*mpg, mpeg*	Any
AVI	Movie	*avi*	Microsoft
QuickTime	Movie	*qt, mov*	Apple
AU	Audio	*au, snd*	Sun
WAV	Audio	*wav*	Microsoft
AIFF	Audio	*aif, aiff*	Apple
MIDI	Audio	*midi, mid*	Any
PostScript	Document	*ps, eps, ai*	Any
Acrobat	Document	*pdf*	Any
PNG	Image	*png*	Any

5.4.1.2 The loop attribute

Like Internet inline movies, the `loop` attribute for the browser's `<bgsound>` tag lets you replay a background soundtrack for a certain number of times (or over and over again forever), at least until the user moves on to another page or quits the browser.

The value of the `loop` attribute is the integer number of times to replay the audio file, or `infinite`, which makes the soundtrack repeat endlessly.

For example:

```
<bgsound src="audio/tadum.wav" loop=10>
```

repeats the ta-dum soundtrack ten times, whereas:

```
<bgsound src="audio/noise.wav" loop=infinite>
```

continuously plays the noise soundtrack.

5.4.2 Alternative Audio Support

There are other ways to include audio in your documents, using more general mechanisms that support other embedded media as well. The most common alternative to the <bgsound> tag is the <embed> tag, originally implemented by Netscape and supplanted by the <object> tag in the HTML 4 and XHTML standards. Take a look in Chapter 12 for details.

5.5 Animated Text

In what appears to be an effort to woo advertisers, Internet Explorer has added a form of animated text to HTML. The animation is simple - text scrolling horizontally across the display - but effective for moving banners and other elements that readily and easily animate an otherwise static document. On the other hand, like the <blink> tag, animated text can easily become intrusive and abusive for the reader. Use with caution, please, if at all.

5.5.1 The <marquee> Tag

The <marquee> tag defines the text that scrolls across the Internet Explorer user's display.

<marquee> O

Function:

Create a scrolling text marquee

Attributes:

ALIGN	LOOP
BEHAVIOR	SCROLLAMOUNT
BGCOLOR	SCROLLDELAY
CLASS	STYLE
DIRECTION	VSPACE
HEIGHT	WIDTH
HSPACE	

End tag:

</marquee>; never omitted

Contains:

plain_text

Used in:

body_content

The <marquee> tag is for Internet Explorer only and is not a standard tag. The text between the <marquee> tag and its required </marquee> end tag scrolls horizontally across the display. The various tag attributes control the size of the display area, its appearance, its alignment with the surrounding text, and the scrolling speed.

The <marquee> tag and attributes are ignored by other browsers, but its contents are not. They are displayed as static text, sans any alignment or special treatments afforded by the <marquee> tag attributes.

5.5.1.1 The align attribute

Internet Explorer places `<marquee>` text into the surrounding body content just as if it were an embedded image. As a result, you can align the marquee within the surrounding text.

The `align` attribute accepts a value of `top`, `middle`, or `bottom`, meaning that the specified point of the marquee will be aligned with the corresponding point in the surrounding text. Thus:

```
<marquee align=top>
```

aligns the top of the marquee area with the top of the surrounding text. Also see the `height` and `width`, `hspace`, and `vspace` attributes (later in this chapter), which control the dimensions of the marquee.

5.5.1.2 The behavior, direction, and loop attributes

Together, these three attributes control the style, direction, and duration of the scrolling in your marquee.

The `behavior` attribute accepts three values:

`scroll` (default)

> The value of `scroll` causes the marquee to act like the grand marquee in Times Square: the marquee area is empty initially; the text then scrolls in from one side (controlled by the `direction` attribute), continues across until it reaches the other side of the marquee, and then scrolls off until the marquee is once again empty.

`slide`

> This value causes the marquee to start empty. Text then scrolls in from one side (controlled by the `direction` attribute), stops when it reaches the other side, and remains onscreen.

`alternate`

> Specifying `alternate` as the value for the `behavior` attribute causes the marquee to start with the text fully visible at one end of the marquee area. The text then scrolls until it reaches the other end, whereupon it reverses direction and scrolls back to its starting point.

If you do not specify a marquee `behavior`, the default `behavior` is `scroll`.

The `direction` attribute sets the direction for marquee text scrolling. Acceptable values are either `left` (the default) or `right`. Note that the starting end for the scrolling is opposite to the direction: `left` means that the text starts at the right of the marquee and scrolls to the left. Remember also that rightward-scrolling text is counter-intuitive to anyone who reads left to right.

The `loop` attribute determines how many times the marquee text scrolls. If an integer value is provided, the scrolling action is repeated that many times. If the value is `infinite`, the scrolling repeats until the user moves on to another document within the browser.

Putting some of these attributes together:

```
<marquee align=center loop=infinite>
  Kumquats aren't filling
  ..........      Taste great, too!
</marquee>
```

The example message starts at the right side of the display window (default), scrolls leftward all the way across and off the Internet Explorer display, and then starts over again until the user moves on to another page. Notice the intervening periods and spaces for the "trailer"; you can't append one marquee to another.

Also, the `slide`-style of scrolling looks jerky when repeated and should only be scrolled once. Other scrolling behaviors work well with repeated scrolling.

5.5.1.3 The bgcolor attribute

The `bgcolor` attribute lets you change the background color of the marquee area. It accepts either an RGB color value or one of the standard color names. See Appendix G for a full discussion of both color-specification methods.

To create a marquee area whose color is yellow, you would write:

```
<marquee bgcolor=yellow>
```

5.5.1.4 The height and width attributes

The `height` and `width` attributes determine the size of the marquee area. If not specified, the marquee area extends all the way across the Internet Explorer display and will be just high enough to enclose the marquee text.

Both attributes accept either a numeric value, indicating an absolute size in pixels, or a percentage, indicating the size as a percentage of the browser window height and width.

For example, to create a marquee that is 50 pixels tall and occupies one-third of the display window width, use:

```
<marquee height=50 width="33%">
```

While it is generally a good idea to ensure the `height` attribute is large enough to contain the enclosed text, it is not uncommon to specify a width that is smaller than the enclosed text. In this case, the text scrolls the smaller marquee area, resulting in a kind of "viewport" marquee familiar to most people.

5.5.1.5 The hspace and vspace attributes

The `hspace` and `vspace` attributes let you create some space between the marquee and the surrounding text. This usually makes the marquee stand out from the text around it.

Both attributes require an integer value specifying the space needed in pixels. The `hspace` attribute creates space to the left and right of the marquee; the `vspace` attribute creates space above and below the marquee. To create 10 pixels of space all the way around your marquee, for example, use:

```
<marquee vspace=10 hspace=10>
```

5.5.1.6 The scrollamount and scrolldelay attributes

These attributes control the speed and smoothness of the scrolling marquee.

The `scrollamount` attribute value is the number of pixels needed to move text each successive movement during the scrolling process. Lower values mean smoother, but slower scrolling; higher numbers create faster, jerkier text motion.

The `scrolldelay` attribute lets you set the number of milliseconds to wait between successive movements during the scrolling process. The smaller this value, the faster the scrolling.

You can use a low `scrolldelay` to mitigate the slowness of a small, smooth `scrollamount`. For example:

```
<marquee scrollamount=1 scrolldelay=1>
```

scrolls the text one pixel for each movement, but does so as fast as possible. In this case, the scrolling speed is limited by the capabilities of the user's computer.

5.6 Other Multimedia Content

The Web is completely open minded about the types of content that can be exchanged by servers and browsers. In this section, we look at a different way to reference images, along with audio, video, and other document formats.

5.6.1 Embedded Versus Referenced Content

Images currently enjoy a special status among the various media that can be included within an HTML or XHTML document and displayed inline with other content by all but a few browsers. Sometimes, however, as we discussed earlier in this chapter, you may also reference images externally, particularly large ones in which details are important, but not immediately necessary to the document content. Other multimedia elements, including digital audio and video, can be referenced as separate documents external to the current one.

You normally use the anchor tag (`<a>`) to link external multimedia elements to the current document. Just like other link elements selected by the user, the browser downloads the multimedia object and presents it to the user, possibly with the assistance of an external application or plug-in. Referenced content is always a two-step process: present the document that links to the desired multimedia object, then present the object if the user selects the link. Section 6.3.1

In the case of images, you can choose how to present images to the user: inline and immediately available via the `<img>` tag, or referenced and subsequently available via the `<a>` tag. If your images are small and critical to the current document, you should provide them inline. If they are large or are only a secondary element of the current document, make them available as referenced content via the `<a>` tag.

If you choose to provide images via the `<a>` tag, it is sometimes a courtesy to your readers to indicate the size of the referenced image in the referencing document and perhaps provide a thumbnail sketch. Users can then determine whether it is worth their time and expense to retrieve it.

5.6.2 Referencing Audio, Video, and Images

You reference any external document, regardless of type or format document via a conventional anchor (<a>) link:

```
The <a href="sounds/anthem.au">Kumquat Grower's Anthem</a> is a rousing tribute to
the thousands of 'quat growers around the world.
```

Just like any referenced document, the server delivers the desired multimedia object to the browser when the user selects the link. If the browser finds the document is not HTML or XHTML, but some other format, it automatically invokes an appropriate rendering tool to display or otherwise convey the contents of the object to the user.

You can configure your browser with special helper applications that handle different document formats in different ways. Audio files, for example, might be passed to an audio-processing tool, while video files are given to a video-playing tool. If a browser has not been configured to handle a particular document format, the browser will inform you and offer to simply save the document to disk. You can later use an appropriate viewing tool to examine the document.

Browsers identify and specially handle multimedia files from one of two different hints: either from the file's Multipurpose Internet Mail Extension (MIME) type provided by the server or from a special suffix in the file's name. The browser prefers MIME because of its richer description of the file and its contents, but will infer the file's contents (type and format) of the object by the file suffix: *.gif* or *.jpg*, for GIF and JPEG encoded images, for example, or *.au* for a special sound file.

Since not all browsers look for a MIME type, nor will they all be correctly configured with helper applications by their users, you should always use the correct file suffix in the names of multimedia objects. See Table 5-1 earlier in this chapter.

5.6.3 Appropriate Linking Styles

Creating effective links to external multimedia documents is critical. The user needs some indication of what the object is and perhaps the kind of application the linked object needs to execute. Moreover, most multimedia objects are quite large, so common courtesy tells us to provide users with some indication of the time and expense involved in downloading it.

In lieu of, or in addition to, the anchor and surrounding text, a small thumbnail of large images or a familiar icon that indicates the referenced object's format is useful.

5.6.4 Embedding Other Document Types

The Web can deliver nearly any type of electronic document, not just graphics, sound, and video files. To display them, however, the client browser needs a helper application installed and referenced. Recent browsers also support plug-in accessory software and, as described in Chapter 12, may extend the browser for some special function, including inline display of multimedia objects.

For example, consider a company whose extensive product documentation was prepared and stored in some popular layout application such as FrameMaker, Quark XPress, or PageMaker. The Web offers an excellent way for distributing that documentation over a worldwide network, but converting to HTML or XHTML would be too costly at this time.

The solution is to prepare a few HTML or XHTML documents that catalog and link the alternative files and invoke the appropriate display applet. Or, make sure that the users' browsers have the plug-in software or are configured to invoke the appropriate helper application. Adobe's Acrobat Reader is a very popular plugin, for example. If the document is in Acrobat (*.pdf*) format, then if a link to an Acrobat document is chosen, the tool is started and accordingly displays the document, often right in the browser's window.

Chapter 6. Links and Webs

Up to this point, we've dealt with HTML and XHTML documents as standalone entities, concentrating on the language elements you use for structure and to format your work. The true power of these markup languages, however, lies in their ability to join collections of documents together into a full library of information and to link your library of documents with other collections around the world. Just as readers have considerable control over how the document looks onscreen, with hyperlinks they also have control over the order of presentation as they navigate through your information. It's the "HT" in HTML and XHTML - hypertext - and it's the twist that spins the Web.

6.1 Hypertext Basics

A fundamental feature of hypertext is that you can hyperlink documents; you can point to another place inside the current document, inside another document in the local collection, or inside a document anywhere on the Internet. The documents become an intricately woven web of information. (Get the name analogy now?) The target document is usually somehow related to and enriches the source; the linking element in the source should convey that relationship to the reader.

Hyperlinks can be used for all kinds of effects. They can be used inside tables of contents and lists of topics. With a click of the mouse on their browser screen or a press of a key on their keyboard, readers select and automatically jump to a topic of interest in the same document or to another document located in an entirely different collection somewhere around the world.

Hyperlinks also point readers to more information about a mentioned topic. "For more information, see Kumquats on Parade," for example. Authors use hyperlinks to reduce repetitive information. For instance, we recommend you sign your name to each of your documents. Rather than include full contact information in each document, a hyperlink connects your name to a single place that contains your address, phone number, and so forth.

A hyperlink, or *anchor* in standard parlance, is marked by the `<a>` tag and comes in two flavors. As we describe in detail later, one type of anchor creates a hot spot in the document that, when activated and selected (usually with a mouse) by the user, causes the browser to link. It automatically loads and displays another portion of the same or another document altogether, or triggers some Internet service-related action, such as sending email or downloading a special file. The other type of anchor creates a label, a place in a document that can be referenced as a hyperlink.[1]

> [1] Both types of anchors use the same tag; perhaps that's why they have the same name. We find it's easier if you differentiate them and think of the one type that provides the hotspot and address of a hyperlink as the "link," and the other type that marks the target portion of a document as the "anchor."

There also are some mouse-related events associated with hyperlinks, which, through JavaScript, let you incorporate some exciting effects.

6.2 Referencing Documents: The URL

As we discussed earlier, every document on the World Wide Web has a unique address. (Imagine the chaos if they didn't.) The document's address is known as its *uniform resource locator* (URL).[2]

> [2] "URL" usually is pronounced "you are ell," not "earl."

Several tags include a URL attribute value, including hyperlinks, inline images, and forms. All use the same URL syntax to specify the location of a web resource, regardless of the type or content of that resource. That's why it's known as a *uniform* resource locator.

Since they can be used to represent almost any resource on the Internet, URLs come in a variety of flavors. All URLs, however, have the same top-level syntax:

scheme:*scheme_specific_part*

The *scheme* describes the kind of object the URL references; the *scheme_specific_part* is, well, the part that is peculiar to the specific scheme. The important thing to note is that the *scheme* is always separated from the *scheme_specific_part* by a colon with no intervening spaces.

6.2.1 Writing a URL

Write URLs using the displayable characters in the US-ASCII character set. For example, surely you have heard what has become annoyingly common on the radio for an announced business website, "h, t, t, p, colon, slash, slash, w, w, w, dot, blah-blah, dot, com." That's a simple URL, written:

```
http://www.blah-blah.com
```

If you need to use a character in a URL that is not part of this character set, you must encode the character using a special notation. The encoding notation replaces the desired character with three characters: a percent sign and two hexadecimal digits whose value corresponds to the position of the character in the ASCII character set.

This is easier than it sounds. One of the most common special characters is the space (Macintosh owners, take special notice), whose position in the character set is 20 hexadecimal. You can't type a space in a URL (well, you can, but it won't work). Rather, replace spaces in the URL with `%20`:

```
http://www.kumquat.com/new%20pricing.html
```

This URL actually retrieves a document named *new pricing.html* from the *www.kumquat.com* server.

6.2.1.1 Handling reserved and unsafe characters

In addition to the nonprinting characters, you'll need to encode reserved and unsafe characters in your URLs as well.

Reserved characters are those that have a specific meaning within the URL itself. For example, the slash character separates elements of a pathname within a URL. If you need to include a slash in a URL that is not intended to be an element separator, you'll need to encode it as `%2F`:[3]

> [3] Hexadecimal numbering is based on 16 characters: through 9 followed by A through F, which in decimal are equivalent to values through 15. Also, letter case for these extended values is not significant; "a" (10 decimal) is the same as "A", for example.

```
http://www.calculator.com/compute?3%2f4
```

This URL actually references the resource named *compute* on the *www.calculator.com* server and passes the string `3/4` to it, as delineated by the question mark (`?`). Presumably, the resource is a server-side program that performs some arithmetic function on the passed value and returns a result.

Unsafe characters are those that have no special meaning within the URL, but may have a special meaning in the context in which the URL is written. For example, double quotes (`""`) delimit URL attribute values in tags. If you were to include a double quotation mark directly in a URL, you would probably confuse the browser. Instead, you should encode the double quotation mark as `%22` to avoid any possible conflict.

Other reserved and unsafe characters that should always be encoded are shown in Table 6-1.

Table 6-1, Reserved and Unsafe Characters and Their URL Encodings

Character	Description	Usage	Encoding
;	Semicolon	Reserved	%3B
/	Slash	Reserved	%2F
?	Question mark	Reserved	%3F
:	Colon	Reserved	%3A
@	At sign	Reserved	%40
=	Equal sign	Reserved	%3D
&	Ampersand	Reserved	%26
<	Less than sign	Unsafe	%3C
>	Greater than sign	Unsafe	%3E
"	Double quotation mark	Unsafe	%22
#	Hash symbol	Unsafe	%23

Character	Description	Usage	Encoding
%	Percent	Unsafe	%25
{	Left curly brace	Unsafe	%7B
}	Right curly brace	Unsafe	%7D
|	Vertical bar	Unsafe	%7C
\	Backslash	Unsafe	%5C
^	Caret	Unsafe	%5E
~	Tilde	Unsafe	%7E
[	Left square bracket	Unsafe	%5B
]	Right square bracket	Unsafe	%5D
`	Back single quotation mark	Unsafe	%60

In general, you should always encode a character if there is some doubt as to whether it can be placed as-is in a URL. As a rule of thumb, any character other than a letter, number, or any of the characters `$-_.+!*'( )` should be encoded.

It is never an error to encode a character, unless that character has a specific meaning in the URL. For example, encoding the slashes in an http URL causes them to be used as regular characters, not as pathname delimiters, breaking the URL.

6.2.2 The http URL

The http URL is by far the most common within the World Wide Web. It is used to access documents from a web server, and it has two formats:

```
http://server:port/path#fragment
http://server:port/path?search
```

Some of the parts are optional. In fact, the most common form of the http URL is simply like this:

```
http://server/path
```

which designates the unique server and the directory path and name of a document.

6.2.2.1 The http server

The *server* is the unique Internet name or Internet Protocol (IP) numerical address of the computer system that stores the web resource. We suspect you'll mostly use more easily remembered Internet names for the servers in your URLs.[4]

[4] Each Internet-connected computer has a unique address, a numeric (IP) address, of course, because computers deal only in numbers. Humans prefer names, so the Internet folks provide us with a collection of special servers and software (Domain Name System or DNS) that automatically resolve Internet names into IP addresses. InterNIC, a nonprofit agency, registers domain names mostly on a first-come, first-serve basis, and distributes new names to DNS servers worldwide.

The name consists of several parts, including the server's actual name and the successive names of its network domain, each part separated by a period. Typical Internet names look like *www.oreilly.com* or *hoohoo.ncsa.uiuc.edu*.[5]

> [5] The three-letter suffix of the domain name identifies the type of organization or business that operates that portion of the Internet. For instance, "com" is a commercial enterprise; "edu" is an academic institution; and "gov" identifies a government-based domain. Outside the United States, a less-descriptive suffix is often assigned, typically a two-letter abbreviation of the country name such as "jp" for Japan and "de" for Deutschland. Many organizations around the world now use the generic three-letter suffixes in place of the more conventional two-letter national suffixes.

It has become something of a convention that webmasters name their servers *www* for quick and easy identification on the Web. For instance, O'Reilly & Associates' web server's name is *www*, which, along with the publisher's domain name, becomes the very easily remembered web site *www.oreilly.com*. Similarly, Sun Microsystems' web server is named *www.sun.com*; Apple Computer's is *www.apple.com*, and even Microsoft makes their web server easily memorable as *www.microsoft.com*. The naming convention has very obvious benefits, which you, too, should take advantage of if you are called upon to create a web server for your organization.

You may also specify the address of a server using its numerical IP address. The address is a sequence of four numbers, zero to 255, separated by periods. Valid IP addresses look like 137.237.1.87 or 192.249.1.33.

It'd be a dull diversion to tell you now what the numbers mean or how to derive an IP address from a domain name, particularly since you'll rarely if ever use one in a URL. Rather, this is a good place to hyperlink: pick up any good Internet networking treatise for rigorous detail on IP addressing, such as Ed Krol's *The Whole Internet User's Guide and Catalog* (O'Reilly & Associates).

6.2.2.2 The http port

The *port* is the number of the communication port to which the client browser connects to the server. It's a networking thing: servers perform many functions besides serve up web documents and resources to client browsers: electronic mail, FTP document fetches, filesystem sharing, and so on. Although all that network activity may come into the server on a single wire, it's typically divided into software-managed "ports" for service-specific communications - something analogous to boxes at your local post office.

The default URL port for web servers is 80. Special secure web servers (Secure HTTP, SHTTP or Secure Socket Layer, SSL) run on port 443. Most web servers today use port 80; you need to include a port number along with an immediately preceding colon in your URL if the target server does *not* use port 80 for web communication.

When the Web was in its infancy, pioneer webmasters ran their Wild Wild Web connections on all sorts of port numbers. For technical and security reasons, system-administrator privileges are required to install a server on port 80. Lacking such privileges, these webmasters chose other, more easily accessible, port numbers.

Now that web servers have become acceptable and are under the care and feeding of responsible administrators, documents being served on some port other than 80 or 443 should make you wonder if that server is really on the up and up. Most likely, the maverick server is being run by a clever user unbeknownst to the server's bona fide system administrators.

6.2.2.3 The http path

The document *path* is the Unix-style hierarchical location of the file in the server's storage system. The pathname consists of one or more names separated by slashes. All but the last name represent directories leading down to the document; the last name is usually that of the document itself.

It has become a convention that for easy identification, HTML document names end with the suffix *.html* (otherwise they're plain ASCII text files, remember?). Although recent versions of Windows allow longer suffixes, their users often stick to the three-letter *.htm* name suffix for HTML documents.

Although the server name in a URL is not case-sensitive, the document pathname may be. Since most web servers are run on Unix-based systems and Unix file names are case-sensitive, the document pathname will be case-sensitive, too. Web servers running on Windows machines are not case-sensitive, so the document pathname is not, but since it is impossible to know the operating system of the server you are accessing, always assume that the server has case-sensitive pathnames and take care to get the case correct when typing your URLs.

Certain conventions regarding the document pathname have arisen. If the last element of the document path is a directory, not a single document, the server usually will send back either a listing of the directory contents or the HTML index document in that directory. You should end the document name for a directory with a trailing slash character, but in practice, most servers will honor the request even if the character is omitted.

If the directory name is just a slash alone or sometimes nothing at all, you will retrieve the first (top-level) document or so-called *home page* in the uppermost root directory of the server. Every well-designed http server should have an attractive, well-designed "home page"; it's a shorthand way for users to access your web collection since they don't need to remember the document's actual filename, just your server's name. That's why, for example, you can type *http://www.oreilly.com* into Netscape's "Open" dialog box and get O'Reilly's home page.

Another twist: if the first component of the document path starts with the tilde character (~), it means that the rest of the pathname begins from the personal directory in the home directory of the specified user on the server machine. For instance, the URL *http://www.kumquat.com/~chuck /* would retrieve the top-level page from Chuck's document collection.

Different servers have different ways of locating documents within a user's home directory. Many search for the documents in a directory named *public_html*. Unix-based servers are fond of the name *index.html* for home pages. When all else fails, servers tend to cough up the first text document in the home page directory.

6.2.2.4 The http document fragment

The *fragment* is an identifier that points to a specific section of a document. In URL specifications, it follows the server and pathname and is separated by the pound sign (#). A fragment identifier indicates to the browser that it should begin displaying the target document at the indicated fragment name. As we describe in more detail later in this chapter, you insert fragment names into a document either with the universal `id` tag attribute or with the `name` attribute for `<a>` tag. Like pathnames, a fragment name may be any sequence of characters.

The fragment name and the preceding hash symbol are optional; omit them when referencing a document without defined fragments.

Formally, the fragment element only applies to HTML or XHTML documents. If the target of the URL is some other document type, the fragment name may be misinterpreted by the browser.

Fragments are useful for long documents. By identifying key sections of your document with a fragment name, you make it easy for readers to link directly to that portion of the document, avoiding the tedium of scrolling or searching through the document to get to the section that interests them.

As a rule of thumb, we recommend that every section header in your documents be accompanied by an equivalent fragment name. By consistently following this rule, you'll make it possible for readers to jump to any section in any of your documents. Fragments also make it easier to build tables of contents for your document families.

6.2.2.5 The http search parameter

The *search* component of the http URL, along with its preceding question mark, is optional. It indicates that the path is a searchable or executable resource on the server. The content of the search component is passed to the server as parameters that control the search or execution function.

The actual encoding of parameters in the search component is dependent upon the server and the resource being referenced. The parameters for searchable resources are covered later in this chapter, when we discuss searchable documents. Parameters for executable resources are discussed in Chapter 9.

Although our initial presentation of http URLs indicated that a URL can have either a fragment identifier or a search component, some browsers let you use both in a single URL. If you so desire, you can follow the search parameter with a fragment identifier, telling the browser to begin displaying the results of the search at the indicated fragment. Netscape, for example, supports this usage.

We don't recommend this kind of URL, though. First and foremost, it doesn't work on a lot of browsers. Just as important, using a fragment implies that you are sure that the results of the search will have a fragment of that name defined within the document. For large document collections, this is hardly likely. You are better off omitting the fragment, showing the search results from the beginning of the document, and avoiding potential confusion among your readers.

6.2.2.6 Sample http URLs

Here are some sample http URLs:

```
http://www.oreilly.com/catalog.html
http://www.oreilly.com/
http://www.kumquat.com:8080/
http://www.kumquat.com/planting/guide.html#soil_prep
http://www.kumquat.com/find_a_quat?state=Florida
```

The first example is an explicit reference to a bona fide HTML document named *catalog.html* that is stored in the root directory of the *www.oreilly.com* server. The second references the top-level home page on that same server. That home page may or may not be *catalog.html*. Sample three, also, assumes that there is a home page in the root directory of the *www.kumquat.com* server, and that the web connection is to the nonstandard port 8080.

The fourth example is the URL for retrieving the web document named *guide.html* from the *planting* directory on the *www.kumquat.com* server. Once retrieved, the browser should display the document beginning at the fragment named *soil_ prep*.

The last example invokes an executable resource named *find_a_quat* with the parameter named *state* set to the value *Florida*. Presumably, this resource generates an HTML response that is subsequently displayed by the browser.

6.2.3 The javascript URL

The javascript URL actually is a pseudo-protocol, not usually included in discussions of URLs. Yet, with advanced browsers like Netscape and Internet Explorer, the javascript URL can be associated with a hyperlink and used to execute JavaScript commands when the user selects the link. Section 12.3.4

6.2.3.1 The javascript URL arguments

What follows the javascript pseudo-protocol is one or more semicolon-separated JavaScript expressions and methods, including references to multi-expression JavaScript functions that you embed within the `<script>` tag in your documents (see Chapter 12 for details). For example:

```
javascript:window.alert('Hello, world!')
javascript:doFlash('red', 'blue'); window.alert('Do not press me!')
```

are valid URLs that you may include as the value for a link reference (see Section 6.3.1.2 and Section 6.5.4.3). The first example contains a single JavaScript method that activates an alert dialog with the simple message.

The second javascript URL example contains two arguments: the first calls a JavaScript function, `doFlash`, which presumably you have located elsewhere in the document within the `<script>` tag and which perhaps flashes the background color of the document window between the red and blue. The second expression is the same alert method as in the first example, with a slightly different message.

The javascript URL may appear in a hyperlink sans arguments, too. In that case, the Netscape browser alone - not Internet Explorer - opens a special JavaScript editor wherein the user may type in and test the various expressions and methods.

6.2.4 The ftp URL

The ftp URL is used to retrieve documents from an FTP (File Transfer Protocol) server.[6]

> [6] FTP is an ancient Internet protocol that dates back to the Dark Ages, around 1975. It was designed as a simple way to move files between machines and is popular and useful to this day. Some people who are unable to run a true web server will place their documents on a server that speaks FTP instead.

It has the format:

`ftp://user:password@server:port/path;type=typecode`

6.2.4.1 The ftp user and password

FTP is an authenticated service, meaning that you must have a valid username and password in order to retrieve documents from a server. However, most FTP servers also support restricted, nonauthenticated access known as *anonymous FTP*. In this mode, anyone can supply the username "anonymous" and be granted access to a limited portion of the server's documents. Most FTP servers also assume (but may not grant) anonymous access if the username and password are omitted.

If you are using an ftp URL to access a site that requires a username and password, include the *user* and *password* components in the URL, along with the colon (`:`) and "at" sign (`@`). More commonly, you'll be accessing an anonymous FTP server, and the user and password components can be omitted.

If you keep the user component along with the "at" sign, but omit the password and the preceding colon, most browsers will prompt you for a password after connecting to the FTP server. This is the recommended way of accessing authenticated resources on an FTP server; it prevents others from seeing your password.

We recommend you *never* place an ftp URL with a username and password in any HTML document. The reasoning is simple: anyone can retrieve the document, extract the username and password from the URL, log into the FTP server, and tamper with its documents.

6.2.4.2 The ftp server and port

The ftp *server* and *port* are bound by the same rules as the server and port in an http URL, as described above. The server must be a valid Internet domain name or IP address of an FTP server. The port specifies the port on which the server is listening for requests.

If the port and its preceding colon are omitted, the default port of 21 is used. It is necessary to specify the port only if the FTP server is running on some port other than 21.

6.2.4.3 The ftp path and transfer type

The *path* component represents a series of directories, separated by slashes leading to the file to be retrieved. By default, the file is retrieved as a binary file; this can be changed by adding the *typecode* (and the preceding `;type=`) to the URL.

If the typecode is set to `d`, the path is assumed to be a directory. The browser will request a listing of the directory contents from the server and display this listing to the user. If the typecode is any other letter, it is used as a parameter to the FTP type command before retrieving the file referenced by the path. While some FTP servers may implement other codes, most servers accept `i` to initiate a binary transfer and `a` to treat the file as a stream of ASCII text.

6.2.4.4 Sample ftp URLs

Here are some sample ftp URLs:

```
ftp://www.kumquat.com/sales/pricing
ftp://bob@bobs-box.com/results;type=d
ftp://bob:secret@bobs-box.com/listing;type=a
```

The first example retrieves the file named *pricing* from the *sales* directory on the anonymous FTP server at *www.kumquat.com*. The second logs into the FTP server on *bobs-box.com* as user `bob`, prompting for a password before retrieving the contents of the directory named *results* and displaying them to the user. The last example logs into *bobs-box.com* as `bob` with the password `secret` and retrieves the file named *listing*, treating its contents as ASCII characters.

6.2.5 The file URL

The file URL specifies a file stored on a machine without indicating the protocol used to retrieve the file. As such, it has limited use in a networked environment. Its real benefit, however, is that it can reference a file on the user's machine, and is particularly useful for referencing personal HTML document collections, such as those "under construction" and not yet ready for general distribution, or HTML document collections on CD-ROM. It has the following format:

`file://`*`server/path`*

6.2.5.1 The file server

The file *server*, like the http server described earlier, must be the Internet domain name or IP address of the machine containing the file to be retrieved. No assumptions are made as to how the browser might contact the machine to obtain the file; presumably the browser can make some connection, perhaps via a Network File System or FTP, to obtain the file.

If the server is omitted, or the special name `localhost` is used, the file is assumed to reside on the same machine upon which the browser is running. In this case, the browser simply accesses the file using the normal facilities of the local operating system. In fact, this is the most common usage of the file URL. By creating document families on a diskette or CD-ROM and referencing your hyperlinks using the *file://localhost/* URL, you create a distributable, standalone document collection that does not require a network connection to use.

6.2.5.2 The file path

This is the path of the file to be retrieved on the desired server. The syntax of the path may differ based upon the operating system of the server; be sure to encode any potentially dangerous characters in the path.

6.2.5.3 Sample file URLs

The file URL is easy:

```
file://localhost/home/chuck/document.html
file:///home/chuck/document.html
file://marketing.kumquat.com/monthly_sales.html
```

The first URL retrieves */home/chuck/document.html* from the user's local machine. The second is identical to the first, except we've omitted the *localhost* reference to the server; the server name defaults to the local server.

The third example uses some protocol to retrieve *monthly_sales.html* from the *marketing.kumquat.com* server.

6.2.6 The news URL

The news URL accesses either a single message or an entire newsgroup within the Usenet news system. It has two forms:

```
news:newsgroup
news:message_id
```

An unfortunate limitation in news URLs is that they don't allow you to specify a server for the *newsgroup*. Rather, users specify their news-server resource in their browser preferences. At one time, not long ago, Internet newsgroups were nearly universally distributed; all news servers carried all the same newsgroups and their respective articles, so one news server was as good as any. Today, the sheer bulk of disk space needed to store the daily volume of newsgroup activity is often prohibitive for any single news server, and there's also local censorship of newsgroups. Hence you cannot expect that all newsgroups, and certainly not all articles for a particular newsgroup, will be available on the user's news server.

Many users' browsers may not be correctly configured to read news. We recommend you avoid placing news URLs in your documents except in rare cases.

6.2.6.1 Accessing entire newsgroups

There are several thousand newsgroups devoted to nearly every conceivable topic under the sun and beyond. Each group has a unique name, composed of hierarchical elements separated by periods. For example, the World Wide Web announcements newsgroup is:

```
comp.infosys.www.announce
```

To access this group, use the URL:

```
news:comp.infosys.www.announce
```

6.2.6.2 Accessing single messages

Every message on a news server has a unique message identifier (ID) associated with it. This ID has the form:

```
unique_string@server
```

The *unique_string* is a sequence of ASCII characters; the server is usually the name of the machine from which the message originated. The *unique_string* must be unique among all the messages that originated from the server. A sample URL to access a single message might be:

```
news:12A7789B@news.kumquat.com
```

In general, message IDs are cryptic sequences of characters not readily understood by humans. Moreover, the lifespan of a message on a server is usually measured in days, after which the message is deleted and the message ID is no longer valid. The bottom line: single message news URLs are difficult to create, become invalid quickly, and are generally not used.

6.2.7 The nntp URL

The nntp URL goes beyond the news URL to provide a complete mechanism for accessing articles in the Usenet news system. It has the form:

```
nntp://server:port/newsgroup/article
```

6.2.7.1 The nntp server and port

The nntp *server* and *port* are defined similarly to the http server and port, described earlier. The server must be the Internet domain name or IP address of a nntp server; the port is the port on which that server is listening for requests.

If the port and its preceding colon are omitted, the default port of 119 is used.

6.2.7.2 The nntp newsgroup and article

The *newsgroup* is the name of the group from which an article is to be retrieved, as defined in Section 6.2.6.

The *article* is the numeric id of the desired article within that newsgroup. Although the article number is easier to determine than a message id, it falls prey to the same limitations of single message references using the news URL, described in Section 6.2.6. Specifically, articles do not last long on most nntp servers, and nntp URLs quickly become invalid as a result.

6.2.7.3 Sample nntp URLs

A sample nntp URL might be:

```
nntp://news.kumquat.com/alt.fan.kumquats/417
```

This URL retrieves article 417 from the *alt.fan.kumquats* newsgroup on *news.kumquat.com*. Keep in mind that the article will be served only to machines that are allowed to retrieve articles from this server. In general, most nntp servers restrict access to those machines on that same local area network.

6.2.8 The mailto URL

The mailto URL causes an electronic mail message to be transmitted to a named recipient. It has the format:

```
mailto:address
```

The *address* is any valid email address, usually of the form:

```
user@server
```

Thus, a typical mailto URL might look like:

```
mailto:cmusciano@aol.com
```

Browsers like Netscape honor multiple recipients in the mailto URL, separated by a comma. For example:

```
mailto:cmusciano@aol.com,bkennedy@activmedia.com,booktech@ora.com
```

will address the message to all three recipients. There should be no spaces before or after the commas in the URL.

6.2.8.1 Defining mail header fields

Most browsers open an email composition window when the user selects a mailto URL. The recipient's address is filled in, taken from the URL, but the message subject and various other header fields are left blank. Many webmasters would like to fill in these fields as a courtesy to their readers, but the URL standard provides no way to do this.

The modern browsers extend the mailto URL to fill this gap. By adding CGI-like parameters to the mailto header, you can set the value of the subject with Netscape and Internet Explorer, and also cc (carbon copy) and bcc (blind carbon copy) fields for the mail message with Netscape. These URLs work with Netscape; only the first one works correctly with Internet Explorer. Section 9.2.4.2

```
mailto:cmusciano@aol.com?subject=Loved your book!
mailto:cmusciano@aol.com?cc=booktech@oreilly.com
mailto:cmusciano@aol.com?bcc=archive@myserver.com
```

As you can probably guess, the first URL sets the subject of the message. Note that spaces are allowed; you don't have to replace them with the hexadecimal equivalent `%20`. The second URL places the address `booktech@oreilly.com` in the cc field of a Netscape message. Similarly, the last example sets the bcc field of the message. You may also set several fields in one URL by separating the field definitions with ampersands. For example:

```
mailto:cmusciano@aol.com?subject=Loved your book!&cc=booktech@
oreilly.com&bcc=archive@myserver.com
```

sets the subject and carbon-copy address. (This line would normally appear as a single line but is broken here due to the width of the page.)

Internet Explorer Version 3 does not recognize the bcc and cc fields in the mailto URL and will either complain about them if they appear alone or append them to a preceding subject.

6.2.9 The telnet URL

The telnet URL opens an interactive session with a desired server, allowing the user to log in and use the machine. Often, the connection to the machine automatically starts a specific service for the user; in other cases, the user must know the commands to type to use the system. The telnet URL has the form:

```
telnet://user:password@server:port/
```

6.2.9.1 The telnet user and password

The telnet *user* and *password* are used exactly like the user and password components of the ftp URL, described previously. In particular, the same caveats apply regarding protecting your password and never placing it within a URL.

Just like the ftp URL, if you omit the password from the URL, the browser should prompt you for a password just before contacting the telnet server.

If you omit both the user and password, the telnet occurs without supplying a user name. For some servers, telnet automatically connects to a default service when no username is supplied. For others, the browser may prompt for a username and password when making the connection to the telnet server.

6.2.9.2 The telnet server and port

The telnet server and port are defined similarly to the http server and port, described above. The server must be the Internet domain name or IP address of a telnet server; the port is the port on which that server is listening for requests. If the port and its preceding colon are omitted, the default port of 23 is used.

6.2.10 The gopher URL

Gopher is a web-like document retrieval system that achieved some popularity on the Internet just before the World Wide Web took off, making Gopher obsolete. Some Gopher servers still exist, though, and the gopher URL lets you access Gopher documents. The gopher URL has the form:

```
gopher://server:port/path
```

6.2.10.1 The gopher server and port

The gopher server and port are defined similarly to the http server and port, described previously. The server must be the Internet domain name or IP address of a gopher server; the port is the port on which that server is listening for requests.

If the port and its preceding colon are omitted, the default port of 70 is used.

6.2.10.2 The gopher path

The path can take one of three forms:

```
type/selector
type/selector%09search
type/selector%09search%09gopherplus
```

The *type* is a single character value denoting the type of the gopher resource. If the entire path is omitted from the gopher URL, the type defaults to 1.

The *selector* corresponds to the path of a resource on the Gopher server. It may be omitted, in which case the top-level index of the Gopher server is retrieved.

If the Gopher resource is actually a Gopher search engine, the *search* component provides the string for which to search. The search string must be preceded by an encoded horizontal tab (`%09`).

If the Gopher server supports Gopher+ resources, the *gopherplus* component supplies the necessary information to locate that resource. The exact content of this component varies based upon the resources on the gopher server. This component is preceded by an encoded horizontal tab (`%09`). If you want to include the *gopherplus* component but omit the search component, you must still supply both encoded tabs within the URL.

6.2.11 Absolute and Relative URLs

You may write a URL in one of two ways: absolute or relative. An absolute URL is the complete address of a resource and has everything your system needs to find a document and its server on the Web. At the very least, an absolute URL contains the scheme and all required elements of the *scheme_specific_part* of the URL. It may also contain any of the optional portions of the *scheme_specific_part*.

With a relative URL, you provide an abbreviated document address that, when automatically combined with a "base address" by the system, becomes a complete address for the document. Within the relative URL, any component of the URL may be omitted. The browser automatically fills in the missing pieces of the relative URL using corresponding elements of a base URL. This base URL is usually the URL of the document containing the relative URL, but may be another document specified with the `<base>` tag. Section 6.7.1

6.2.11.1 Relative schemes and servers

A common form of a relative URL is missing the scheme and server name. Since many related documents are on the same server, it makes sense to omit the scheme and server name from the relative URL. For instance, assume the base document was last retrieved from the server *www.kumquat.com*. The relative URL, then:

```
another-doc.html
```

is equivalent to the absolute URL:

```
http://www.kumquat.com/another-doc.html
```

Table 6-2 shows how the base and relative URLs in the example are combined to form an absolute URL.

Table 6-2, Forming an Absolute URL				
	Protocol	**Server**	**Directory**	**File**
Base URL	http	*www.kumquat.com*	/	
Relative URL	↓	↓	↓	*another-doc.html*
↓	↓	↓	↓	↓
Absolute URL	http	*www.kumquat.com*	/	*another-doc.html*

6.2.11.2 Relative document directories

Another common form of a relative URL omits the leading slash and one or more directory names from the beginning of the document pathname. The directory of the base URL is automatically assumed to replace these missing components. It's the most common abbreviation, because most authors place their collection of documents and subdirectories of support resources in the same directory path as the home page. For example, you might have a *special/* subdirectory containing FTP files referenced in your document. Let's say that the absolute URL for that document is:

```
http://www.kumquat.com/planting/guide.html
```

A relative URL for the file *README.txt* in the *special/* subdirectory, looks like this:

```
ftp:special/README.txt
```

You'll actually be retrieving:

```
ftp://www.kumquat.com/planting/special/README.txt
```

Visually, the operation looks like that in Table 6-3.

Table 6-3, Forming an Absolute FTP URL				
	Protocol	**Server**	**Directory**	**File**
Base URL	http	*www.kumquat.com*	*/planting*	*guide.html*
Relative URL	ftp	↓	*special*	*README.txt*
↓	↓	↓	↓	↓
Absolute URL	ftp	*www.kumquat.com*	*/planting/special*	*README.txt*

Common "dot-slash" pathname notations also let you express the current directory ("./") and directory above the current directory (parent; "../") in a relative URL. The current directory notation is rarely used, since it is redundant. But the parent notation lets you set the target URL to directories in other branches of the filesystem hierarchy.

For example, if the directory portion of the current URL is */planting/special/*, and you want to reference an HTML document named *new_ground.html* in *planting/standard/*, you may simply form the relative URL as:

```
../standard/new_ground.html
```

You'll actually be retrieving:

```
http://www.kumquat.com/planting/standard/new_ground.html
```

Note that parent notation has limits. For instance, most web servers will not let you navigate above the base directory: *http://www.kumquat.com/../* probably won't deliver any document or directory listing to your browser.

6.2.11.3 Using relative URLs

Relative URLs are more than just a typing convenience. Because they are relative to the current server and directory, you can move the entire set of documents to another directory or even another server and never have to change a single relative link. Imagine the difficulties if you had to go into every source document and change the URL for every link every time you move it. We'd loathe using hyperlinks! Use relative URLs wherever possible.

6.3 Creating Hyperlinks

Use the HTML/XHTML <a> tag to create links to other documents and to name anchors for fragment indentifiers within documents.

6.3.1 The <a> Tag

You will use the <a> tag most commonly with its href attribute to create a hypertext link, or *hyperlink*, for short, to another place in the same document or to another document. In these cases, the current document is the source of the link; the value of the href attribute, a URL, is the target.[7]

> [7] You may run across the terms "head" and "tail," which reference the target and source of a hyperlink. This naming scheme assumes that the referenced document (the head) has many tails that are embedded in many referencing documents throughout the Web. We find this naming convention confusing and stick to the concept of source and target documents throughout this book.

The other way you can use the <a> tag is with the name attribute to mark a hyperlink target, or fragment identifier, in a document. This method, although part of the HTML 4 and XHTML standards, is slowly succumbing to the id attribute which lets you mark nearly any element, including paragraphs, divisions, forms, and so on, as a hyperlink target.

The standards let you use both the name and href attributes within a single <a> tag, defining a link to another document and a fragment identifier within the current document. We recommend against this, since it overloads a single tag with multiple functions, and some browsers may not be able to handle it.

Instead, use two <a> tags when such a need arises. Your source will be easier to understand and modify and will work better across a wider range of browsers.

6.3.1.1 Allowed content

Between the <a> tag and its required end tag, you may put only regular text, line breaks, images, and headings. The browser renders all of these elements normally, but with the addition of some special effects to indicate that it is a hyperlink to another document. For instance, the popular graphical browsers typically underline and color the text and draw a colored border around images that are enclosed by <a> tags.

While the allowed content may seem restricted (the inability to place style markup within an <a> tag is a bit onerous, for instance), most browsers let you put just about anything within an <a> tag that makes sense. To be compliant with the HTML 4 and XHTML standards, place the <a> tag inside other markup tags, not the opposite. For example, while most browsers make sense of either variation on this anchor theme:

```
To subscribe to
<cite><a href="ko.html">Kumquat Online</a></cite>,

To subscribe to
<a href="ko.html"><cite>Kumquat Online</cite></a>,
```

only the first example is technically correct, and the second is most certainly incorrect for XHTML.

6.3.1.2 The href attribute

Use the href attribute to specify the URL of the target of a hyperlink. Its value is any valid document URL, absolute or relative, including a fragment identifier or a JavaScript code fragment. If the user selects the contents of the <a> tag, the browser will attempt to retrieve and display the document indicated by the URL specified by the href attribute or execute the list of JavaScript expressions, methods, and functions. Section 6.2

A simple <a> tag that references another document might be:

```
The <a href="http:growing_season.html">growing
season</a> for kumquats in the Northeast.
```

which appears in the Netscape display as shown in Figure 6-1.

<a>

Function:

Define anchors within a text flow

Attributes:

ACCESSKEY	ONKEYPRESS
CHARSET	ONKEYUP
CLASS	ONMOUSEDOWN
COORDS	ONMOUSEMOVE
DIR	ONMOUSEOUT
HREF	ONMOUSEOVER
HREFLANG	ONMOUSEUP
ID	REL
LANG	REV
NAME	SHAPE
ONBLUR	STYLE
ONCLICK	TABINDEX
ONDBLCLICK	TARGET
ONFOCUS	TITLE
ONKEYDOWN	TYPE

End tag:

; always present

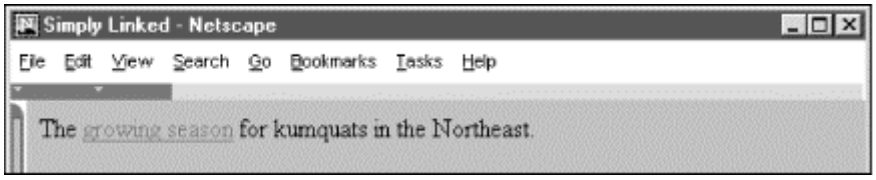
Contains:

a_content

Used in:

text

Figure 6-1. Hyperlink to another HTML document



Notice that the phrase "growing season" is specially rendered by the browser, letting the user know that it is a link to another document. Users also typically have the option to specially set the text color of the link and have the color change when a link is taken; blue initially and then red after it has been selected at least once, for instance.

More complex anchors might include images:

```
<ul>
  <li><a href="pruning_tips.html">
        <img src="pics/new.gif" align=center>
         New pruning tips!</a>
  <p>
  <li><a href="xhistory.html">
        <img src="pics/new2.gif" align=center>
         Kumquats throughout history</a>
</ul>
```

Most graphical browsers like Netscape and Internet Explorer place a special border around images that are part of an anchor, as shown in Figure 6-2. Remove that hyperlink border with the `border=0` attribute and value within the `<img>` tag reference for the image. Section 5.2.6.8

Figure 6-2. Internet Explorer puts a special border around an image that is inside an anchor

6.3.1.3 The name and id attributes

Use the `name` and `id` attributes with the `<a>` tag to create a fragment identifier within a document. Once created, the fragment identifier becomes a potential target of a link.

Prior to HTML 4.0, the only way to create a fragment identifier was to use the name attribute with the `<a>` tag. With the advent of the id attribute in HTML 4.0, and its ability to be used with almost any tag, any HTML or XHTML element can be a fragment identifier. The `<a>` tag retains the name attribute for historic purposes and honors the id attribute as well. These attributes can be used interchangeably, with id being the more "modern" version of the name attribute. Both name and id can be specified in conjunction with the href attribute, allowing a single `<a>` to be both a hyperlink and a fragment indentifier.

An easy way to think of a fragment identifier is as the HTML analog of the `goto` statement label common in many programming languages. The `name` attribute within the `<a>` tag or the `id` attribute within the `<a>` or other tags places a label within a document. When that label is used in a link to that document, it is the equivalent of telling the browser to `goto` that label.

The value of the `id` or `name` attribute is any character string, enclosed in quotation marks. The string must be a unique label, not reused in any other `name` or `id` attribute in the same document, although it can be reused in different documents. Here are `name` and `id` examples:

```
<h2><a name="Pruning">Pruning Your Kumquat Tree</a></h2>
<h2 id="Pruning">Pruning Your Kumquat Tree</h2>
```

Notice we set the anchor in a section header of a presumably large document. It's a practice we encourage you to use for all major sections of your work for easier reference and future smart processing, such as automated extraction of topics.

The following link, when taken by the user:

```
<a href="growing_guide.html#Pruning">
```

jumps directly to the section of the document we named in the previous examples.

The contents of the anchor <a> tag with the name or id attribute are not displayed in any special way.

Technically, you do not have to put any document content within the <a> tag with the name attribute, since it simply marks a location in the document. In practice, some browsers ignore the tag unless some document content - a word or phrase, even an image - is between the <a> and tags. For this reason, it's probably a good idea to have at least one displayable element in the body of any <a> tag.

6.3.1.4 The event attributes

There are a number of event handlers built into the modern browsers. These handlers watch for certain conditions and user actions, such as a click of the mouse or when an image finishes loading into the browser window. With client-side JavaScript, you may include selected event handlers as attributes of certain tags and execute one or more JavaScript commands and functions when the event occurs.

With the anchor (<a>) tag, you may associate JavaScript code with a number of mouse- and keyboard-related events. The value of the event handler is - enclosed in quotation marks - one or a sequence of semicolon-separated JavaScript expressions, methods, and function references that the browser executes when the event occurs. Section 12.3.3

A popular, albeit simple, use of the onMouseOver event with a hyperlink is to print an expanded description of the tag's destination in the JavaScript-aware browser's status box (Figure 6-3). Normally, the browser displays the frequently cryptic destination URL there whenever the user passes the mouse pointer over an <a> tag's contents:

```
<a href="http://www.ora.com/kumquats/homecooking/recipes.html#quat5"
onMouseOver="status='A yummy recipe for kumquat soup.'; return true;">
<img src="pics/bowl.gif" border=0>
</a>
```

We argue that the contents of the tag itself should explain the link, but there are times when window space is tight and an expanded explanation is helpful, such as when the link is in a table of contents.

See Chapter 12 for more about JavaScript.

Figure 6-3. Use JavaScript to display a message in the browser's status box

6.3.1.5 The rel and rev attributes

The rel and rev attributes express a formal relationship and direction between source and target documents. The rel attribute specifies the relationship from the source document to the target, and the rev attribute specifies the relationship from the target to the source. Both attributes can be placed in a single <a> tag, and the browser may use them to specially alter the appearance of the anchor content or to automatically construct document navigation menus. Other tools also may use these attributes to build special link collections, tables of contents, and indexes.

The value of either the rel or rev attribute is a space-separated list of relationships. The actual relationship names and their meanings are up to you: they are not formally addressed by the HTML or XHTML standards. For example, a document that is part of a sequence of documents might include its relationship in a link:

```
<a href="part-14.html" rel=next rev=prev>
```

The relationship from the source to the target is that of moving to the next document; the reverse relationship is that of moving to the previous document.

These document relationships are also used in the <link> tag in the document <head>. The <link> tag establishes the relationship without actually creating a link to the target document; the <a> tag creates the link and imbues it with the relationship attributes. Section 6.7.2

Commonly used document relationships include:

`next`
: Links to the next document in a collection

`prev`
: Links to the previous document in a collection

`head`
: Links to the top-level document in a collection

`toc`
: Links to a collection's table of contents

`parent`
: Links to the document above the source

`child`
: Links to a document below the source

`index`
: Links to the index for this document

`glossary`
: Links to the glossary for this document

Few browsers take advantage of these attributes to modify the link appearance. However, these attributes are a great way to document links you create, and we recommend that you take the time to insert them whenever possible.

6.3.1.6 The style and class attributes

Use the `style` and `class` attributes for the `<a>` tag to control the display style for the content enclosed by the tag and to format the content according to a predefined class of the `<a>` tag. Section 8.1.1 Section 8.3

6.3.1.7 The lang and dir attributes

Like almost all other tags, the `<a>` tag accepts the `lang` and `dir` attributes, denoting the language used for the content within the `<a>` tag and the direction in which that language is rendered. Section 3.6.1.1 Section 3.6.1.2

6.3.1.8 The target attribute

The `target` attribute lets you specify where to display the contents of a selected hyperlink. Commonly used in conjunction with frames or multiple browser windows, the value of this attribute is the name of the frame or window in which the referenced document should be loaded. If the named frame or window exists, the document is loaded in that frame or window. If not, a new window is created, given the specified name, and the document is loaded in that new window. For more information, including a list of special target names, see Section 11.7.

6.3.1.9 The title attribute

The `title` attribute lets you specify a title for the document to which you are linking. The value of the attribute is any string, enclosed in quotation marks. The browser might use it when displaying the link, perhaps flashing the title when the mouse passes over the link. The browser might also use the `title` attribute when adding this link to a user's hotlist.

The `title` attribute is especially useful for referencing an otherwise unlabeled resource, such as an image or a non-HTML document. For example, the browser might include the following title on this otherwise wordless image display page:

```
<a href="pics/kumquat.gif"
   title="A photograph of the Noble Fruit">
```

Ideally, the value specified should match the title of the referenced document, but it's not required.

6.3.1.10 The charset, hreflang, and type attributes

According to the HTML 4 and XHTML standards, the `charset` attribute specifies the character encoding used in the document that is the destination of the link. The value of this attribute must be the name of a standard character set: "euc-jp," for example. The default value is "ISO-8859-1".

The hreflang attribute may be specified only when the href attribute is used. Like the lang attribute, its value is an ISO standard two-character language code. Unlike the lang attribute, the hreflang attribute does not address the language used by the contents of the tag. Instead, it specifies the language used in the document referenced by the href attribute. Section 3.6.1.2

The type attribute specifies the content type of the resource referenced by the <a> tag. Its value is any MIME encoding type. For example, you might inform the browser that you are linking to a plain ASCII document with:

```
<a href="readme.txt" type="text/plain">
```

The browser might use this information when displaying the referenced document, or might even present the link differently based upon the content type.

6.3.1.11 The coords and shape attributes

These are two more attributes defined in the HTML and XHTML standards for the <a> tag that are not supported by the current, popular browsers. Like the attributes of the same names for <area> tag, the coords and shape attributes define a region of influence for the <a> tag. These attributes should only be used with the <a> tag when that tag is part of the content of a <map> tag, as described later in this chapter. Section 6.5.3 Section 6.5.4.2 Section 6.5.4.7

6.3.1.12 The accesskey and tabindex attributes

Traditionally, users of graphical browsers select and execute a hyperlink by pointing and clicking the mouse device on the region of the browser display defined by the anchor. What is less well known is that you may choose a hyperlink, among other objects in the browser window, by pressing the Tab key and then activate that link by pressing the Return or Enter key. With the tabindex attribute, you may reorder the sequence in which the browser steps through to each object when the user presses the Tab key. The value of this attribute is an integer greater than zero. The browser starts with the object whose tab index is 1 and moves through the other objects in increasing order.

With the accesskey attribute, you may select an alternative "hot-key" that, when pressed, activates the specific link. The value of this attribute is a single character that is pressed in conjunction with an "alt" or "meta" key, depending on the browser and computing platform. Ideally, this character should appear in the content of the <a> tag; if so, the browser may choose to display the character differently to indicate that it is a hot-key.

See an expanded description for both these attributes in Chapter 9.

6.3.2 Linking to Other Documents

You make a hyperlink to another document with the <a> tag and its href attribute, which defines the URL of the target document. The contents of the <a> tag are presented to the user in some distinctive manner in order to indicate that the link is available.

When creating a link to another document, you should consider adding the title, rel, and rev attributes to the <a> tag. They help document the link you are creating and allow the browser to embellish the display anchor contents.

6.3.3 Linking Within a Document

Creating a link within the same document or to a specific fragment of another document is a two-step process. The first step is to make the target fragment; the second is to create the link to the fragment.

linking within a document

Use the <a> tag with its name attribute to identify a fragment. Here's a sample fragment identifier:

```
<h3><a name="Section_7">Section 7</a></h3>
```

Alternatively, use the id attribute and embed the hyperlink target directly in a defining tag, such as with a header:[8]

[8] We prefer the id way, although not all browsers support it, yet.

```
<h3 id="Section_7">Section 7</h3>
```

A hyperlink to the fragment is an <a> tag with the href attribute, in which the attribute's value - the target URL - ends with the fragment's name, preceded by the pound sign (#). A reference to the previous example's fragment identifier, then, might look like:

```
See <a href="index.html#Section_7">Section 7</a>
for further details.
```

By far the most common use of fragment identifiers is in creating a table of contents for a lengthy document. Begin by dividing your document into several logical sections, using appropriate headers and consistent formatting. At the start of each section, add a fragment identifier for that section, typically as part of the section title. Finally, make a list of links to those fragment identifiers at the beginning of your document.

Our sample document extolling the life and wonders of the mighty kumquat, for example, is quite long and involved, including many sections and subsections of interest. It is a document to be read and read again. In order to make it easy for kumquat lovers everywhere to find their section of interest quickly, we've included fragment identifiers for each major section and placed an ordered list of links - a hotlinked table of contents, as it were - at the beginning of each of the Kumquat Lover's documents, a sample of which appears below along with sample fragment identifiers that appear in the same document. The ellipsis symbol (...) means that there are intervening segments of content, of course:

```
...
<h3>Table of Contents</h3>
<ol>
  <li><a href="#soil_prep">Soil Preparation</a>
  <li><a href="#dig_hole">Digging the Hole</a>
  <li><a href="#planting">Planting the Tree</a>
</ol>
...
<h3 id=soil_prep>Soil Preparation</h3>
...
<h3 id=dig_hole>Digging the Hole</h3>
...
<h3 id=planting>Planting the Tree</h3>
...
```

The kumquat lover can thereby click the desired link in the table of contents and jump directly to the section of interest, without lots of tedious scrolling.

Notice also that this example uses relative URLs - a good idea if you ever intend to move or rename the document without breaking all the hyperlinks.

6.4 Creating Effective Links

A document becomes hypertext by tossing in a few links in the same way that water becomes soup when you throw in a few vegetables. Technically, you've met the goal, but the outcome may not be very palatable.

Inserting anchors into your documents is something of an art, requiring good writing skills, HTML/XHTML prowess, and an architectural sense of your documents and their relationships to others on the Web. Effective links flow seamlessly into a document, quietly supplying additional browsing opportunities to the reader without disturbing the current document. Poorly designed links scream out, interrupt the flow of the source document, and generally annoy the reader.

While there are as many linking styles as there are authors, here are a few of the more popular ways to link your documents. All do two things: they give the reader quick access to related information, and they tell the reader how the link is related to the current contents.

6.4.1 Lists of Links

Perhaps the most common way to present hyperlinks is in ordered or unordered lists in the style of a table of contents or list of resources.

Two schools of style exist. One puts the entire list item into the source anchor; the other abbreviates the item and puts a shorthand phrase in the source anchor. In the former, make sure you keep the anchor content short and sweet; in the latter, use a direct writing style that makes it easy to embed the link.

If your list of links becomes overly long, consider organizing it into several sublists grouped by topic. Readers can then scan the topics (set off, perhaps, as <h3> headers) for the appropriate list and then scan that list for the desired document.

The alternative list style is much more descriptive, but also more wordy, so you have to be careful that it doesn't end up cluttered:

```
<p>
Kumquat-related documents include:
<ul>
  <li>A concise guide to <a href="kumquat_farming.html">
      profitable kumquat farming</a>,
      including a variety of business plans, lists of fruit
      packing companies, and farming supply companies.
  <li>101 different ways to <a href="kumquat_uses">
      use a kumquat</a>, including stewed kumquats and kumquat pie!
```

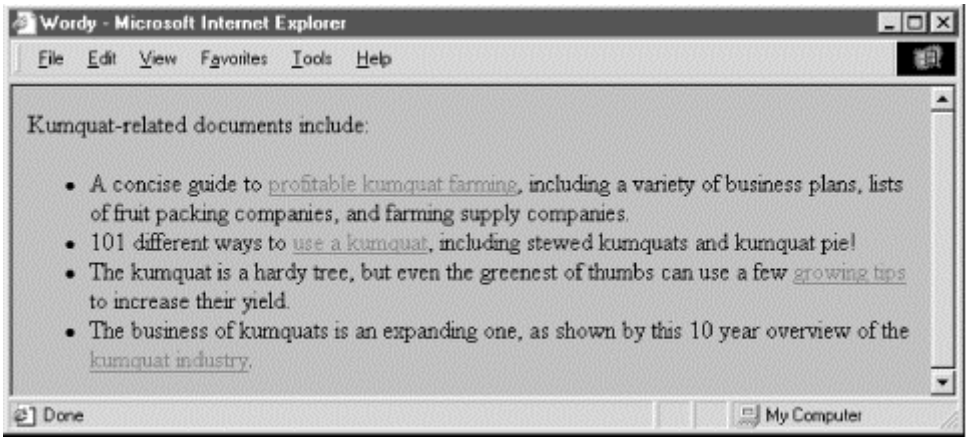
```
  <li>The kumquat is a hardy tree, but even the greenest of
      thumbs can use a few <a href="news:alt.kumquat_growers">
      growing tips</a> to increase
      their yield.
  <li>The business of kumquats is an expanding one, as
      shown by this 10 year overview of the
      <a href="http://www.oreilly.com/kumquat_report/">
      kumquat industry</a>.
</ul>
```

It sometimes gets hard to read a source HTML document, and it will become even more tedious with XHTML. Imagine the clutter if we'd used anchors with fragment identifiers for each of the subtopics in the list item explanations. Nonetheless, it all looks pristine and easily navigable when displayed by the browser, such as with Internet Explorer as shown in Figure 6-4.

This more descriptive style of presenting a link list tries hard to draw readers into the linked document by giving a fuller taste of what they can expect to find. Because each list element is longer and requires more scanning by the reader, you should use this style sparingly and dramatically limit the number of links.

Use the brief list style when presenting large numbers of links to a well-informed audience. The second, more descriptive style is better suited to a smaller number of links for which your readership is less well-versed in the topic at hand.

Figure 6-4. Wordy but effectively descriptive link list

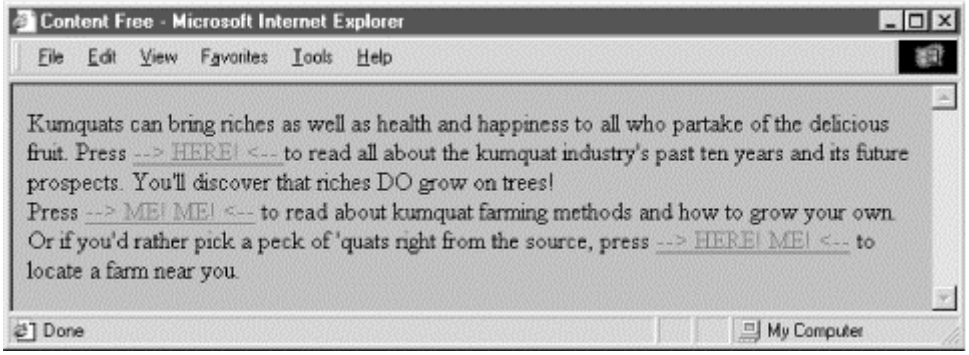


6.4.2 Inline References

If you aren't collecting links into lists, you're probably sprinkling them throughout your document. So-called inline links are more in keeping with the true spirit of hypertext, since they enable readers to mark their current place in the document, visit the related topic in more depth or find a better explanation, and then come back to the original and continue reading. That's very personalized information processing.

The biggest mistake made by novice authors, however, is to overload their documents with links and treat them as if they are panic buttons demanding to be pressed. You may have seen this style of linking; HTML pages with the word "here" all over the place, like the panic-ridden example in Figure 6-5 (we can't bring ourselves to show you the source for this travesty).

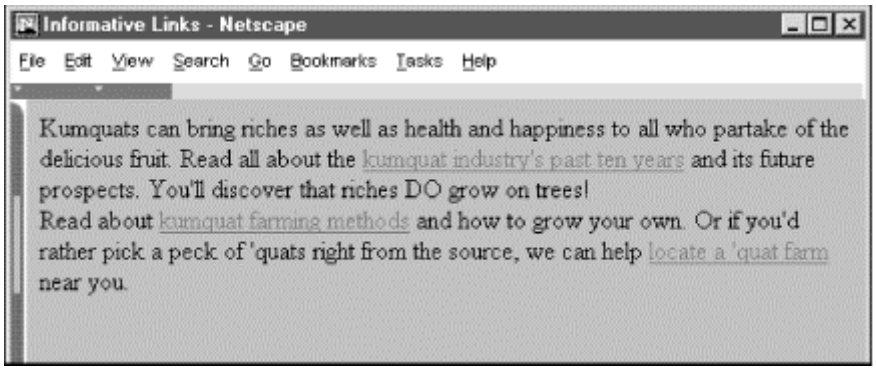
Figure 6-5. Links should not wave and yell like first-graders, "Here! Me! Me!"



As links, phrases like "click here" and "also available" are content-free and annoying. They make the person who is scanning the page for an important link read all the surrounding text to actually find the reference.

The better, more refined style for an inline link is to make every one contain a noun or noun/verb phrase relating to the topic at hand. Compare how kumquat farming and industry news references are treated in Figure 6-6 to the "Here! Me! Me!" example in Figure 6-5.

Figure 6-6. Kinder, gentler inline links work best



A quick scan of Figure 6-6 immediately yields useful links to "kumquat farming methods" and "kumquat industry's past ten years." There is no need to read the surrounding text to understand where the link will take you. Indeed, the immediately surrounding content in our example, as for most inline links, serves only as syntactic sugar in support of the embedded links.

Embedding links into the general discourse of a document takes more effort to create than link lists. You've got to actually understand the content of the current as well as the target documents, be able to express that relationship in just a few words, and then intelligently incorporate that link at some key place in the source document. Hopefully this key place is where you might expect the user is ready to interrupt their reading and ask a question or request more information. To make matters even more difficult, particularly for the traditional tech writer, this form of author-reader conversation is most effective when presented in active voice (he, she, or it does something to an object versus the object having something done to it). The effort expended is worthwhile, though, resulting in more informative, easily read documents. Remember, you'll write the document once, but it will be read thousands, if not millions, of times. Please your readers, please.

6.4.3 Linking Do's and Don'ts

Here are some hints for creating links:

Keep the link content as concise as possible

Long links or huge inline graphic icons for links are visually disruptive and potentially confusing.

Never place two links immediately adjacent to one another

Most browsers make it difficult to tell where one link stops and the next link starts. Separate them with regular text or line breaks.

Be consistent

If you are using inline references, make all of your links inline references. If you choose to use lists of links, stick to either the short or long form; don't mix styles in a single document.

Try reading your document with all the nonanchor text removed

If some links suddenly make no sense, rewrite them so that they stand on their own. (Many people scan documents looking only for links; the surrounding text becomes little more than a gray background to the visually more compelling links.)

6.4.4 Using Images and Links

It has become fashionable to use images and icons instead of words for link contents. For instance, instead of the word "next," you might use an icon of a little pointing hand. A link to the home page is not complete without a picture of a little house. Links to searching tools must now contain a picture of a magnifying glass, question mark, or binoculars. And all those flashing, GIF-animated little advertisements!

Resist falling prey to the "Mount Everest syndrome" of inserting images simply because you can. Again, it's a matter of context. If you or your document's readers can't tell at a glance what relationship a link has with the current document, you've failed. Use cute images for links sparingly, consistently, and only in ways that help readers scan your document for important information and leads. Also be ever mindful that your pages may be read by someone from nearly anywhere on Earth (perhaps beyond, even) and that images do not translate consistently across cultural boundaries. (Ever hear what the "okay" hand sign common in the United States means to a Japanese person?)

Creating consistent iconography for a collection of pages is a daunting task, one that really should be done with the assistance of someone formally schooled in visual design. Trust us, the kind of mind that produces nifty code and writes XHTML well is rarely suited to creating beautiful, compelling imagery. Find a good visual designer; your pages and readers will benefit immeasurably.

6.5 Mouse-Sensitive Images

Normally, an image placed within an anchor simply becomes part of the anchor content. The browser may alter the image in some special way (usually with a special border) to alert the reader that it is a hyperlink, but users click the image in the same way they click a textual hyperlink.

The HTML and XHTML standards provide a feature that lets you embed many different links inside the same image. Clicking different areas of the image causes the browser to link to different target documents. Such mouse-sensitive images, known as *image maps*, open up a variety of creative linking styles.

There are two ways to create image maps, known as *server-side* and *client-side* image maps. The former, enabled by the `ismap` attribute for the `<img>` tag, requires access to a server and related image map processing applications. The latter is created with the `usemap` attribute for the `<img>` tag, along with corresponding `<map>` and `<area>` tags. Since translation of the mouse position in the image to a link to another document happens on the user's machine, client-side image maps don't require a special server connection and can even be implemented in non-web environments, such as on a local hard drive or on a CD-ROM-based document collection. Section 6.5.3 Section 6.5.4 Section 5.2.6

6.5.1 Server-Side Image Maps

You add an image to an anchor simply by placing an `<img>` tag within the body of the `<a>` tag. Make that embedded image into a mouse-sensitive one by adding the `ismap` attribute to the `<img>` tag. This special `<img>` attribute tells the browser that the image is a special map containing more than one link. (The `ismap` attribute is ignored by the browser if the `<img>` tag is not within an `<a>` tag.) Section 5.2.6

When the user clicks some place within the image, the browser passes the coordinates of the mouse pointer along with the URL specified in the `<a>` tag to the document server. The server uses the mouse pointer coordinates to determine which document to deliver back to the browser.

When `ismap` is used, the `href` attribute of the containing `<a>` tag must contain the URL of a server application or, for some HTTP servers, a related map file that contains the coordinate and linking information. If the URL is simply that of a conventional document, errors may result and the desired document will most likely not be retrieved.

The coordinates of the mouse position are screen pixels counted from the upper-left corner of the image beginning with (0,0). The coordinates are added to the end of the URL, preceded by a question mark.

For example, if a user clicks 43 pixels over and 15 pixels down from the upper-left corner of the image displayed from the following link:

```
<a href="/cgi-bin/imagemap/toolbar.map">
<img ismap src="pics/toolbar.gif">
</a>
```

the browser sends the following search parameters to the HTTP server:

```
/cgi-bin/imagemap/toolbar.map?43,15
```

In the example, *toolbar.map* is a special image map file inside the *cgi-bin/imagemap* directory and containing coordinates and links. A special image map process uses that file to match the passed coordinates (43,15 in the example) and return the selected hyperlink document.

6.5.1.1 Server-side considerations

With mouse-sensitive `ismap`-enabled image maps, the browser is required to pass along only the URL and mouse coordinates to the server. Converting these coordinates into a specific document is handled by the document server. The conversion process differs between servers and is not defined by the HTML or XHTML standards.

You need to consult with your web server administrators and perhaps even read your server's documentation to determine how to create and program an image map. Most servers come with some software utility, typically located in a *cgi-bin/imagemap* directory, to handle image maps. And most of these use a text file containing the image map regions and related hyperlinks that is referenced by your image map URL to process the image map query.

Here's an example image map file that describes the sensitive regions in our example image:

```
# Imagemap file=toolbar.map

default                 dflt.html
circ 100,30,50          link1.html
rect 180,120,290,500    link2.html
poly 80,80,90,72,160,90 link3.html
```

Each sensitive region of the image map is described by a geometric shape and defining coordinates in pixels, such as the circle with its center point and radius, the rectangle's upper-left and lower-right edge coordinates, and the loci of a polygon. All coordinates are relative to the upper-left corner of the image (0,0). Each shape has a related URL.

An image map processing application typically tests each shape in the order it appears in the image file and returns the document specified by the corresponding URL to the browser if the user's mouse x,y coordinates fall within the boundaries of that shape. That means it's okay to overlap shapes; just be aware which takes precedence. Also, the entire image need not be covered with sensitive regions: if the passed coordinates don't fall within a specified shape, the default document gets sent back to the browser.

This is just one example for how an image map may be processed and the accessory files required for that process. Please huddle with your webmaster and server manuals to discover how to implement a server-side image map for your own documents and system.

6.5.2 Client-Side Image Maps

The obvious downside to server-side image maps is that they require a server. That means you need access to the required HTTP server or its */cgi-bin/* directory, which isn't always available. And server-side image maps limit portability, since not all image map processing applications are the same.

Server-side image maps also mean delays for the user while browsing, since the browser must get the server's attention to process the image coordinates. That's even if there's no action to take, such as a section of the image that isn't hyperlinked and doesn't lead anywhere.

Client-side image maps suffer from none of these difficulties. Enabled by the `usemap` attribute for the `<img>` tag, and defined by special `<map>` and `<area>` extension tags, client-side image maps let authors include in their documents a map of coordinates and links that describe the sensitive regions of an image. The browser on the client computer translates the coordinates of the mouse position within the image into an action, such as loading and displaying another document. And special JavaScript-enabled attributes provide a wealth of special effects for client-side image maps. Section 12.3.3

To create a client-side image map, include the `usemap` attribute as part of the `<img>` tag.[9] Its value is the URL of a `<map>` segment in an HTML document that contains the map coordinates and related link URLs. The document in the URL identifies the HTML document containing the map; the fragment identifier in the URL identifies the map to be used. Most often, the map is in the same document as the image itself, and the URL can be reduced to the fragment identifier: a pound sign (#) followed by the map name.

> [9] Alternatively, according to the HTML 4 standard, you may reference a client-side image map by including the `usemap` attribute with the `<object>` and form `<input>` tags, too. See Chapter 12 for details.

For example, the following source fragment tells the Netscape or Internet Explorer browser that the *map.gif* image is a client-side image map and that its mouse-sensitive coordinates and related link URLs are found in the `map` section of the document named *map* :

```
<img src="pics/map.gif" usemap="#map">
```

6.5.3 The <map> Tag

For client-side image maps to work, you must include somewhere in your document a set of coordinates and URLs that define the mouse-sensitive regions of a client-side image map and the hyperlink to take for each region that is clicked by the user. Include those coordinates and links as values of attributes in conventional <a> tags or special <area> tags; the collection of <area> specifications or <a> tags are enclosed within the <map> tag and its end tag </map>. The <map> segment may appear anywhere in the body of the document.

<map>

Function:

Encloses client-side image map (usemap) specifications

Attributes:

NAME

Contains:

map_content

Used in:

body_content

More specifically, the <map> tag may contain either a sequence of <area> tags or conventional HTML/XHTML content including <a> tags. You cannot mix and match <area> tags with conventional content. Conventional content within the <map> tag may be displayed by the browser; <area> tags will not. If you are concerned about compatability with older browsers, use only <map> tags containing <area> tags.

If you do place <a> tags within a <map> tag, they must include the shape and coords attributes that define a region within the objects that reference the <map> tag.

6.5.3.1 The name attribute

The value of the name attribute in the <map> tag is the name used by the usemap attribute in an <img> or <object> tag to locate the image map specification. The name must be unique and not used by another <map> in the document, but more than one image map may reference the same <map> specifications. Section 5.2.6.14

6.5.4 The <area> Tag

The guts of a client-side image map are the <area> tags within the map segment. These <area> tags define each mouse-sensitive region and the action the browser should take if it is selected by the user in an associated client-side image map.

The region defined by an <area> tag acts just like any other hyperlink: when the user moves the mouse pointer over the region of the image, the pointer icon will change and the browser may display the URL of the related hyperlink in the status box at the bottom of the browser window.[10] Regions of the client-side image map not defined in at least one <area> tag are not mouse-sensitive.

[10] That is, unless you activate a JavaScript event handler that writes the contents of the status box. See the onMouse event handlers in Section 6.5.4.6.

6.5.4.1 The alt attribute

Like its cousin for the <img> tag, the alt attribute for the <area> tag lets attaches a text label to the image, except in this case the label is associated with a particular area of the image. The popular browsers display this label to the user when the mouse passes over the area, and it may also be used by a nongraphical browser to present the client-side image map as a list of links identified by the alt labels.

<area>

Function:

Defines coordinates and links for a region on a client-side image map

Attributes:

ACCESSKEY	ONKEYPRESS
ALT	ONKEYUP
CLASS	ONMOUSEDOWN
COORDS	ONMOUSEMOVE
DIR	ONMOUSEOUT
HREF	ONMOUSEOVER
ID	ONMOUSEUP
LANG	SHAPE
NOHREF	STYLE
NOTAB I	TABINDEX
ONBLUR	TABORDER I
ONCLICK	TARGET N I
ONDBLCLICK	TITLE
ONFOCUS	
ONKEYDOWN	

End tag:

None in HTML; </area> or <area ... /> in XHTML

Contains:

Nothing

Used in:

map_content

6.5.4.2 The coords attribute

The required coords attribute of the <area> tag defines coordinates of a mouse-sensitive region in a client-side image map. The number of coordinates and their meaning depend upon the region's shape as determined by the shape attribute, discussed later in this chapter. You may define hyperlink regions as rectangles, circles, and polygons within a client-side image map. The appropriate values for each shape are listed.

circle or circ

coords="*x*,*y*,*r* ", where *x* and *y* define the position of the center of the circle (0,0 is the upper-left corner of the image) and *r* is its radius in pixels.

polygon or poly

coords="*x1*,*y1*,*x2*,*y2*,*x3*,*y3*,...", where each pair of *x,y* coordinates define a vertex of the polygon, with 0,0 being the upper-left corner of the image. At least three pairs of coordinates are required to define a triangle, higher order polygons require a larger number of vertices. The polygon is automatically closed, so it is not necessary to repeat the first coordinate at the end of the list to close the region.

rectangle or rect

coords="*x1*,*y1*,*x2*,*y2* ", where the first coordinate pair is one corner of the rectangle and the other pair is the corner diagonally opposite, with "0,0" being the upper-left corner of the image. Note that a rectangle is just a shortened way of specifying a polygon with four vertices.

For example, the following XHTML fragment defines a single mouse-sensitive region in the lower-right quarter of a 100 x 100 image and another circular region smack in the middle:

```
<map name="map1">
  <area shape="rect" coords="75,75,99,99" nohref="nohref" />
  <area shape="circ" coords="50,50,25" nohref="nohref" />
</map>
```

If the coordinates in one <area> tag overlap with another region, the first <area> tag takes precedence. The browsers ignore coordinates that extend beyond the boundaries of the image.

6.5.4.3 The href attribute

Like the href attribute for the anchor (<a>) tag, the href attribute for the <area> tag defines the URL of the desired link if its region in the associated image map is clicked. The value of the href attribute is any valid URL, relative or absolute, including JavaScript code.

For example, the browser will load and display the *link4.html* document if the user clicks in the lower-left quarter of a 100 x 100-pixel image, as defined by the first image map <area> tag in the following HTML example:

```
<map name="map">
  <area coords="75,75,99,99" href="link4.html">
  <area coords="0,0,25,25" href="javascript:window.alert
  ('Oooh, tickles!')"
  </map>
```

The second <area> tag in the example uses a JavaScript URL, which, when the user clicks in the upper-left quadrant of the image map, executes a JavaScript alert method that displays the silly message in a dialog box.

6.5.4.4 The nohref attribute

The nohref attribute for the <area> tag defines a mouse-sensitive region in a client-side image map for which no action is taken even though the user may select it. You must include either an href or a nohref attribute for each <area> tag.

6.5.4.5 The notab, taborder, and tabindex attributes

As an alternative to the mouse, a user may choose a document "hotspot," such as a hyperlink embedded in an image map, by pressing the Tab key. Once chosen, the user activates the hyperlink by pressing the Return or Enter key. By default, the browser steps to each hotspot in the order in which they appear in the document. Originally introduced by Internet Explorer with the taborder attribute, and now standardized as the tabindex attribute, you may change that default order. The value of the attribute is an integer indicating the position of this area in the overall tab sequence for the document.

Supported by Internet Explorer only and not part of either the HTML 4 or XHTML standards, notab areas get passed over as the user presses the Tab key to move the cursor around the document. Otherwise, this area will be part of the tabbing sequence. The attribute is useful, of course, in combination with the nohref attribute.

The notab and taborder attributes were supported by Internet Explorer version 4. The browser's version 5 supports tabindex as well.

6.5.4.6 The event attributes

The same mouse-related JavaScript event handlers that work for the anchor (<a>) tag also work with client-side image map hyperlinks. The value of the event handler is - enclosed in quotation marks - one or a sequence of semicolon-separated JavaScript expressions, methods, and function references that the browser executes when the event occurs. Section 12.3.3

For example, a popular, albeit simple, use of the onMouseOver event is to print a more descriptive explanation in the browser's status box whenever the user passes the mouse pointer over a region of the image map:

```
<area href="http://www.oreilly.com/kumquats/homecooking/recipes.html#quat5"
   onMouseOver="self.status='A recipe for kumquat soup.';return true">
```

We should point out that the current versions of the popular browsers automatically display the alt attribute's string value, ostensibly accomplishing the same task. So we recommend that you include the alt attribute and value in lieu of hacking JavaScript. And, in context with a text-based hyperlink, we argue that the contents of the tag itself should explain the link. But images can be deceptive, so we urge you to take advantage of both the alt attribute and event handlers to provide text descriptions with your image maps.

6.5.4.7 The shape attribute

Use the shape attribute to define the shape of an image map's mouse-sensitive region: a circle (circ or circle), polygon (poly or polygon), or rectangle (rect or rectangle).

The value of the shape attribute affects how the browser interprets the value of the coords attribute. If you don't include a shape attribute, the value default is assumed. According to the standard, default means that the area covers the entire image. In practice, the browsers default to a rectangular area and expect to find four coords values. And if you don't specify a shape and don't include four coordinates with the tag, the browser ignores the area altogether.

In fact, Netscape is the only browser that even recognizes the shape value default to provide a catch-all area for clicks that fall outside all the other defined hotspots. Since areas are in a "first-come, first-served" order in the <map> tag, you should place the default area last. Otherwise, it covers up any and all areas that follow in your image map.

The browsers are lax in their implementation of the shape names. Netscape 4, for example, doesn't recognize "rectangle" but does recognize "rect" for a rectangular shape. For this reason, we recommend that you use the abbreviated names.

6.5.4.8 The target attribute

The target attribute gives you a way to control where the contents of the selected hyperlink in the image map get displayed. Commonly used in conjunction with frames or multiple browser windows, the value of this attribute is the name of the frame or window in which the referenced document should be loaded. If the named frame or window exists, the document is loaded in that frame or window. If not, a new window is created, given the specified name, and the document is loaded in that new window. For more information, including a list of special target names, see Section 11.7.

6.5.4.9 The title attribute

The title attribute lets you specify a title for the document to which the image map's area links. The value of the attribute is any string, enclosed in quotes. The browser might use the title when displaying the link, perhaps flashing the title when the mouse passes over the area. The browser might also use the title attribute when adding this link to a user's hotlist.

The title attribute is especially useful for referencing an otherwise unlabeled resource, such as an image or a non-HTML document. Ideally, the value specified should match the title of the referenced document, but it's not required.

6.5.4.10 The class, dir, id, lang, and style attributes

The class and style attributes allow you to supply display properties and class names to control the appearance of the area, although their value seems limited for this tag. The id attribute allows you to create a name for the area that might referenced by a hyperlink. Section 4.1.1.4 Section 8.1.1 Section 8.3

The lang and dir attributes define the language used for this area and the direction in which text is rendered. Again, their use is not apparent with this tag. Section 3.6.1.1 Section 3.6.1.2

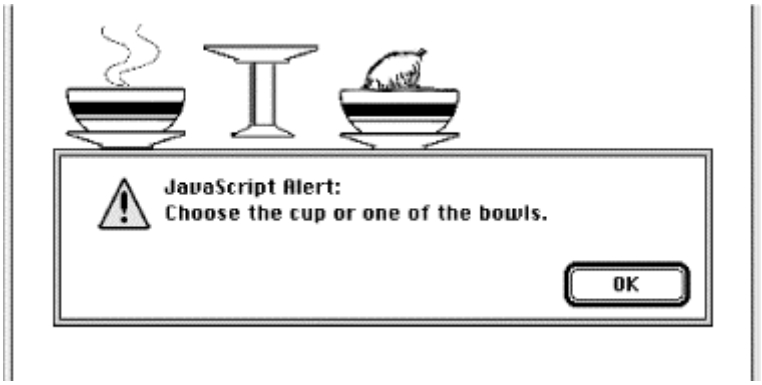
6.5.5 A Client-Side Image Map Example

The following example HTML fragment draws together the various components of a client-side image map discussed earlier in this section. It includes the <img> tag with the image reference and usemap attribute with a name that points to a <map> that defines four mouse-sensitive regions (three plus a default) and related links:

```
<body>
...
<img src="pics/map.gif" usemap="#map1" border=0>
...
<map name="map1">
  <area shape=rect coords="0,20,40,100"
      href="k_juice.html"
      onMouseOver="self.status='How to prepare kumquat juice.'
         ;return true">
  <area shape=rect coords="50,50,80,100"
      href="k_soup.html"
      onMouseOver="self.status='A recipe for hearty kumquat soup.'
      ;return true">
  <area shape=rect coords="90,50,140,100"
      href="k_fruit.html"
      onMouseOver="self.status='Care and handling of the native
                   kumquat.'
      ;return true">
  <area shape=default
      href="javascript:window.alert('Choose the cup or one of the
            bowls.')"
      onMouseOver="self.status='Select the cup or a bowl for more
                   information.'
      ;return true">
</map>
```

See Figure 6-7 for the results.

Figure 6-7. A simple client-side image map with JavaScript-enabled mouse event



6.5.6 Handling Other Browsers

Unlike its server-side ismap counterpart, the client-side image map tag (<img usemap>) doesn't need to be included in an <a> tag. But it may be, so that you can gracefully handle browsers that are unable to process client-side image maps.

For example, Mosaic or early versions of Netscape simply load a document named *main.html* if the user clicks the *map.gif* image referenced in the following source fragment. The extended browsers, on the other hand, will divide the image into mouse-sensitive regions, as defined in the associated <map> and link to a particular name anchor within the same *main.html* document if the image map region is selected by the user:

```
<a href="main.html">
  <img src="pics/map.gif" ismap usemap="#map1">
</a>
...
<map name="map1">
  <area coords="0,0,49,49" href="main.html#link1">
  <area coords="50,0,99,49" href="main.html#link2">
  <area coords="0,50,49,99" href="main.html#link3">
  <area coords="50,50,99,99" href="main.html#link4">
</map>
```

To make an image map backward-compatible with all image map-capable browsers, you may also include client-side and server-side processing for the same image map. Capable browsers will honor the faster client-side processing; all other browsers will ignore the usemap attribute in the <img> tag and rely upon the referenced server process to handle user selections in the traditional way.

For example:

```
<a href="/cgi-bin/images/map.proc">
  <img src="pics/map2.gif" usemap="#map2" ismap>
</a>
...
<map name="map2">
  <area coords="0,0,49,49" href="link1.html">
  <area coords="50,0,99,49" href="link2.html">
  <area coords="0,50,49,99" href="link3.html">
  <area coords="50,50,99,99" href="link4.html">
</map>
```

6.5.7 Effective Use of Mouse-Sensitive Images

Some of the most visually compelling pages on the Web have mouse-sensitive images: maps with regions that when clicked, for example, lead to more information about a country or town or result in more detail about the location and who to contact at a regional branch of a business. We've seen a mouse-sensitive image of a fashion model whose various clothing parts lead to their respective catalog entries, complete with detailed description and price tag for ordering.

The visual nature of mouse-sensitive images coupled with the need for an effective interface means you should strongly consider having an artist, a user-interface designer, and even a human-factors expert evaluate your mouse-sensitive imagery. At the very least, engage in a bit of user testing to make sure people know where to click to move to the desired document. Make sure the "mouseable" areas of the image indicate this to the user using a consistent visual mechanism. Consider using borders, drop shadows, or color changes to indicate those areas that can be selected by the user.

Finally, always remember that the decision to use mouse-sensitive images is an explicit decision to exclude text-based and image-restricted browsers from your pages. This includes browsers connecting to the Internet via slow modem connections. For these people, downloading your beautiful images is simply too expensive. To keep from disenfranchising a growing population, make sure any page that has a mouse-sensitive image has a text-only equivalent easily accessible from a link on the image-enabled version. Some thoughtful webmasters even provide separate pages for users preferring full graphics versus mostly text.

6.6 Creating Searchable Documents

Another extensible form of an HTML link that does not use the `<a>` tag is one that causes the server to search a database for a document that contains a user-specified keyword or words. An HTML document that contains such a link is known as a *searchable* document.

6.6.1 The <isindex> Tag (Deprecated)

No longer supported in the HTML 4 or XHTML standards, authors at one time used the `<isindex>` tag to pass keywords along with a search-engine's URL to the server. The server then matched the keywords against a database of terms to select the next document for display. Today's authors mostly use forms to pass information to the server and supporting programs. See Chapter 9 for details.

When a browser encounters the `<isindex>` tag, it adds a standard search interface to the document (rendered by Internet Explorer in Figure 6-8):

```
<html>
<head>
<title>Kumquat Advice Database</title>
<base href="cgi-bin/quat-query">
<isindex>
</head>
<body>
<h3>Kumquat Advice Database</h3>
<p>
Search this database to learn more about kumquats!
</body>
</html>
```

The user types a list of space-separated keywords into the field provided. When the user presses the Enter key, the browser automatically appends the query list to the end of a URL and passes the information to the server for further processing.

<isindex>

Function:

Indicates that a document can be searched

Attributes:

ACTION	LANG
CLASS	PROMPT
DIR	STYLE
ID	TITLE

End tag:

None in HTML; </isindex> or <isindex ... /> with XHTML

Contains:

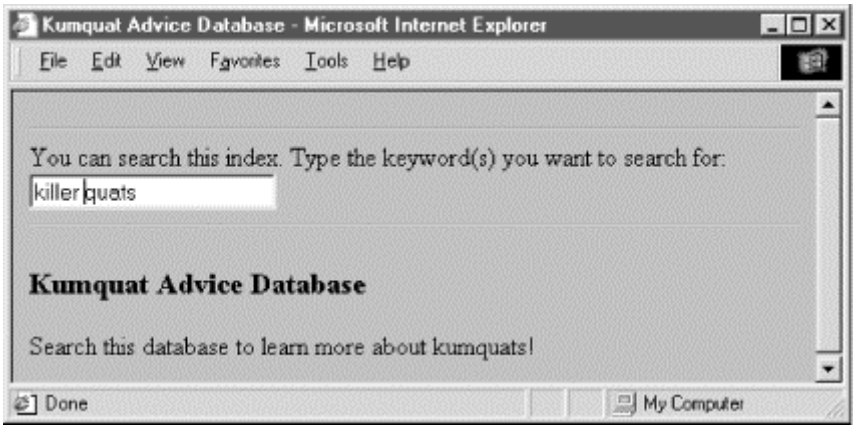
Nothing

Used in:

head_content

While the HTML and XHTML standards only allow the deprecated `<isindex>` tag to be placed in the document header, most browsers let the tag appear anywhere in the document and insert the search field in the content flow where the `<isindex>` tag appears. This convenient extension lets you add instructions and other useful elements before presenting the user with the actual search field.

Figure 6-8. A searchable document



6.6.1.1 The prompt attribute

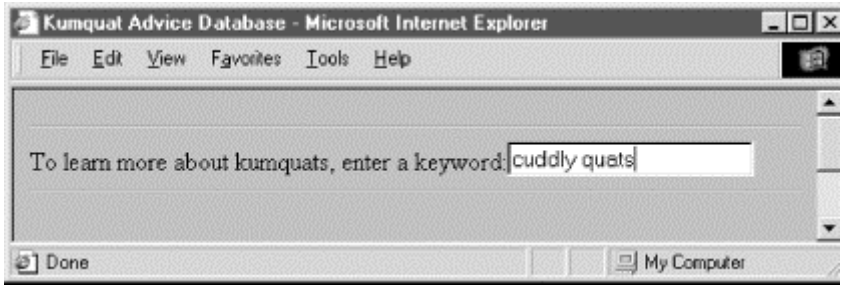
The browser provides a leading prompt just above or to the left of the user-entry field. Internet Explorer's default prompt, for example, is "You can search this index. Type the keyword(s) you want to search for:" (Figure 6-8). That default prompt is not the best for all occasions, so it is possible to change it with the `prompt` attribute.

When added to the `<isindex>` tag, the value of the `prompt` attribute is the string of text that precedes the keyword entry field placed in the document by the browser.

For example, compare Figure 6-8 with Figure 6-9, in which we added the following prompt to the previous source example:

```
<isindex prompt="To learn more about kumquats, enter a keyword:">
```

Figure 6-9. The prompt attribute creates custom prompts in searchable documents



Older browsers will ignore the `prompt` attribute, but there is little reason not to include a better prompt string for your more up-to-date readership.

6.6.1.2 The query URL

Besides the `<isindex>` tag in the header of a searchable document, the other important element of this special tag is the query URL. By default, it is the URL of the source document itself - not good if your document can't handle the query. Rather, most authors use the `<base>` attribute to point to a different URL for the search. Section 6.7.1

The browser appends a question mark to the query URL, followed by the specified search parameters. Nonprintable characters are appropriately encoded; multiple parameters are separated by a plus sign (+).

In the previous example, if a user types "insect control" in the search field, the browser would retrieve the URL:

```
cgi-bin/quat-query?insect+control
```

6.6.1.3 The action attribute

For Internet Explorer only, you can specify the query URL for the index with the `action` attribute. The effect is exactly as if you had used the `href` attribute with the `<base>` tag: the browser links to the specified URL with the search parameters appended to the URL.

While the `action` attribute provides the desirable feature of divorcing the document's base URL from the search index URL, it will cause your searches to fail if the user is not using Internet Explorer. For this reason, we do not recommend that you use the `action` attribute to specify the query URL for the search.

6.6.1.4 The class, dir, id, lang, style, and title attributes

The `class` and `style` attributes allow you to supply display properties and class names to control the appearance of the tag, although their value seems limited for `<isindex>`. The `id` and `title` attributes allow you to create a name and title for the tag; the name might referenced by a hyperlink. Section 4.1.1.4 Section 4.1.1.5 Section 8.1.1 Section 8.3

The `dir` and `lang` attributes define the language used for this tag and the direction in which text is rendered. Again, their use is not apparent with `<isindex>`. Section 3.6.1.1 Section 3.6.1.2

6.6.1.5 Server dependencies

Like image maps, searchable documents require support from the server to make things work. How the server interprets the query URL and its parameters is not defined by the HTML or XHTML standards.

You should consult your server's documentation to determine how you can receive and use the search parameters to locate the desired document. Typically, the server breaks the parameters out of the query URL and passes them to a program designated by the URL.

6.7 Relationships

Very few documents stand alone. Instead, a document is usually part of a collection of documents, each connected by one or several of the hypertext strands we describe in this chapter. One document may be a part of several collections, linking to some documents and being linked to by others. Readers move between the document families as they follow the links that interest them.

You establish an explicit relationship between two documents when you link them. Conscientious authors use the `rel` attribute of the `<a>` tag to indicate the nature of the link. In addition, two other tags may be used within a document to further clarify the location and relationship of a document within a document family. These tags, `<base>` and `<link>`, are placed within the body of the `<head>` tag. Section 3.7.1

6.7.1 The <base> Header Element

As we previously explained, URLs within a document can be either absolute (with every element of the URL explicitly provided by the author) or relative (with certain elements of the URL omitted and supplied by the browser). Normally, the browser fills in the blanks of a relative URL by drawing the missing pieces from the URL of the current document. You can change that with the `<base>` tag.

<base>

Function:

Define the base URL for other anchors in the document

Attributes:

HREF
TARGET

End tag:

None in HTML; </base> or <base ... /> with XHTML

Contains:

Nothing

Used in:

head_content

The `<base>` tag should appear only in the document header, not its body contents. The browser thereafter uses the specified base URL, not the current document's URL, to resolve all relative URLs, including those found in `<a>`, `<img>`, `<link>`, and `<form>` tags. It also defines the URL that will be used to resolve queries in searchable documents containing the `<isindex>` tag. Section 6.2

6.7.1.1 The href attribute

The `href` attribute must have a valid URL as its value, which the browser then uses to define the absolute URL against which relative URLs are based within the document. For example, the `<base>` tag in this XHTML document head:

```
<head>
<base href="http://www.kumquat.com/" />
</head>
...
```

tells the browser that any relative URLs within this document are relative to the top-level document directory on *www.kumquat.com*, regardless of the address and directory of the machine from which the user had retrieved the current document.

Contrary to what you may expect, you can make the base URL relative, not absolute. The browser should (but doesn't always) form an absolute base URL out of this relative URL by filling in the missing pieces with the URL of the document itself. This property can be used to good advantage. For instance, in this next HTML example:

```
<head>
<base href="/info/">
</head>
...
```

the browser makes the `<base>` URL into one relative to the server's */info/* directory, which probably is not the same directory of the current document. Imagine if you had to re-address every link in your document with that common directory. Not only does the `<base>` tag help you shorten those URLs in your document that have a common root, it also lets you constrain the directory from which relative references are retrieved without binding the document to a specific server.

6.7.1.2 The target attribute

When working with documents inside frames, the target attribute with the <a> tag ensures that a referenced URL gets loaded into the correct frame. Similarly, the target attribute for the <base> tag lets you establish the default name of one of the frames or windows in which the browser is to display redirected hyperlinked documents. Section 11.1

If you have no other default target for your hyperlinks within your frames, you may want to consider using <base target=_top>. This ensures that links that are not specifically targeted to a frame or window will thereby load in the top-level browser window. This eliminates the embarrassing and common error of having references to pages on other sites appear within a frame on your pages, instead of within their own pages. A minor bit of HTML, to be sure, but it makes life much easier for your readership.

6.7.1.3 Using <base>

The most important reason for using <base> is to ensure that any relative URLs within the document will resolve into a correct document address, even if the document itself is moved or renamed. This is particularly important when creating a document collection. By placing the correct <base> tag in each document, you can move the entire collection between directories and even servers without breaking all of the links within the documents.

You also need to use the <base> tag for a searchable document (<isindex>) if you want user queries posed to a URL different from the host document.

A document that contains both the <isindex> tag and other relative URLs may have problems if the relative URLs are not relative to the desired index-processing URL. Since this is usually the case, don't use relative URLs in searchable documents that use the <base> tag to specify the query URL for the document.

6.7.2 The <link> Header Element

Use the <link> tag to define the relationship between the current document and another in a Web collection.

The <link> tag belongs in the <head> content, nowhere else. The attributes of the <link> tag are used like those of the <a> tag, but their effects serve only to document the relationship between documents. The <link> tag has no content and only XHTML supports the closing </link> tag.

6.7.2.1 The href attribute

As with its other tag applications, the href attribute specifies the URL of the target <link> tag. It is a required attribute, and its value is any valid document URL. The specified document is assumed to have a relationship to the current document.

6.7.2.2 The rel and rev attributes

The rel and rev attributes express the relationship between the source and target documents. The rel attribute specifies the relationship from the source document to the target; the rev attribute specifies the relationship from the target document to the source document. Both attributes can be included in a single <link> tag.

The value of either attribute is a space-separated list of relationships. The actual relationship names are not specified by the HTML standard, although some have come into common usage as listed in Section 6.3.1.5. For example, a document that is part of a sequence of documents might use:

```
<link href="part-14.html" rel=next rev=prev>
```

when referencing the next document in the series. The relationship from the source to the target is that of moving to the next document; the reverse relationship is that of moving to the previous document.

6.7.2.3 The title attribute

The title attribute lets you specify the title of the document to which you are linking. This attribute is useful when referencing a resource that does not have a title, such as an image or a non-HTML document. In this case, the browser might use the <link> title when displaying the referenced document. For example:

```
<link href="pics/kumquat.gif"
 title="A photograph of the Noble Fruit">
```

tells the browser to use the indicated title when displaying the referenced image.

The value of the attribute is an arbitrary character string, enclosed in quotation marks.

<link>

Function:

Define a relationship between this document and another document

Attributes:

CHARSET	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
HREF	ONMOUSEOUT
HREFLANG	ONMOUSEOVER
ID	ONMOUSEUP
LANG	REL
MEDIA	REV
ONCLICK	STYLE
ONDBLCLICK	TARGET
ONKEYDOWN	TITLE
ONKEYPRESS	TYPE

End tag:

None in HTML; </link> or <link ... /> with XHTML

Contains:

Nothing

Used in:

head_content

6.7.2.4 The type attribute

The `type` attribute provides the MIME content type of the linked document. Supported by both Internet Explorer and Netscape, the HTML 4 and XHTML standard type attribute can be used with any linked document. It is often used to define the type of linked style sheets. In this context, the value of the `type` attribute is usually `text/css`. For example:

```
<link href="styles/classic.css" rel=stylesheet type="text/css">
```

creates a link to an external style sheet within the `<head>` of a document. See Chapter 8 for details.

6.7.2.5 How browsers might use <link>

Although the standards do not require browsers to do anything with the information provided by the `<link>` tag, it's not hard to envision how this information might be used to enhance the presentation of a document.

As a simple example, suppose you consistently provide <link> tags for each of your documents that define next, prev, and parent links. A browser could use this information to place a standard toolbar at the top or bottom of each document containing buttons that would jump to the appropriate related document. By relegating the task of providing simple navigational links to the browser, you are free to concentrate on the more important content of your document.

As a more complex example, suppose a browser expects to find a <link> tag defining a glossary for the current document, and that this glossary document is itself a searchable document. Whenever a reader clicked on a word or phrase in the document, the browser could automatically search the glossary for the definition of the selected phrase, presenting the result in a small pop-up window.

As the Web evolves, expect to see more and more uses of the <link> tag to define document relationships explicitly.

6.7.2.6 Other <link> attributes

The HTML 4 and XHTML standards also include the ubiquitous collection of attributes related to style sheets and user events, and language for the <link> tag. You can refer to the corresponding section describing these attributes for the <a> tag for a complete description of their usage. Section 6.3.1

Since you put the <link> tag in the <head> section, whose contents do not get displayed, it may seem that these attributes are useless. It is entirely possible that some future browser may find some way to display the <link> information to the user, possibly as a navigation bar or a set of hot-list selections. In those cases, the display and rendering information would prove useful. Currently, no browser provides these capabilities.

6.8 Supporting Document Automation

There are two additional header tags that have the primary function of supporting document automation, and interacting with the web server itself and document-generation tools.

6.8.1 The <meta> Header Element

Given the rich set of header tags for defining a document and its relationship with others that go unused by most authors, you'd think we'd all be satisfied.

<meta>

Function:

Supply additional information about a document

Attributes:

CHARSET	LANG
CONTENT	NAME
DIR	SCHEME
HTTP_EQUIV	

End tag:

None in HTML; </meta> or <meta ... /> with XHTML

Contains:

Nothing

Used in:

head_content

But no. There's always someone with special needs. They want to be able to give even more information about their precious document, information that might be used by browsers, readers of the source, or by document-indexing tools. The <meta> tag is for you who need to go beyond the beyond.

The <meta> tag belongs in the document header and has no content. Instead, attributes of the tag define name/value pairs that associate the document. In certain cases, these values are used by the web server serving the document to further define the document content type to the browser.

6.8.1.1 The name attribute

The name attribute supplies the name of the name/value pair defined by the <meta> tag. Neither the HTML nor the XHTML standard specify any predefined <meta> names. In general, you are free to use any name that makes sense to you and other readers of your source document.

One common name used is keywords, which defines a set of keywords for the document. When encountered by any of the popular search engines on the Web, these keywords will be used to categorize the document. If you want your documents to be indexed by a search engine, consider putting this kind of tag in the <head> of each document:

```
<meta name="keywords" content="kumquats, cooking, peeling, eating">
```

If the name attribute is not provided, the name of the name/value pair is taken from the http-equiv attribute.

6.8.1.2 The content attribute

The content attribute provides the value of the name/value pair. It can be any valid string, enclosed in quotes, if it contains spaces. It should always be specified in conjunction with either a name or http-equiv attribute.

As an example, you might place the author's name in a document with:

```
<meta name="Authors" content="Chuck Musciano & Bill Kennedy">
```

6.8.1.3 The http-equiv attribute

The http-equiv attribute supplies a name for the name/value pair and instructs the server to include the name/value pair in the MIME document header that is passed to the browser before sending the actual document.

When a server sends a document to a browser, it first sends a number of name/value pairs. While some servers might send a number of these pairs, all servers send at least one:

```
content-type: text/html
```

This tells the browser to expect to receive an HTML document.

When you use the <meta> tag with the http-equiv attribute, the server will add your name/value pairs to the content header it sends to the browser. For example, adding:

```
<meta http-equiv="charset" content="iso-8859-1">
<meta http-equiv="expires" content="31 Dec 99">
```

causes the header sent to the browser to contain:

```
content-type: text/html
charset: iso-8859-1
expires: 31 Dec 99
```

Of course, adding these additional header fields makes sense only if your browser accepts the fields and uses them in some appropriate manner.

6.8.1.4 The charset attribute

Internet Explorer provides explicit support for a charset attribute in the <meta> tag. Set the value of the attribute to the name of the character set to be used for the document. This is not the recommended way to define a document's character set. Rather, we recommend always using the http-equiv and content attributes to define the character set.

6.8.1.5 The scheme attribute

This attribute specifies the scheme to be used to interpret the property's value. This scheme should be defined within the profile specified by the profile attribute of the <head> tag. Section 3.7.1

6.8.2 The <nextid> Header Element

This tag is not defined in the HTML 4 or XHTML standards and should not be used. We describe it here for historical reasons.

<nextid>

Function:

Define the next valid document entity identifier

Attributes:

n

End tag:

None

Contains:

Nothing

Used in:

head_content

The idea behind the `<nextid>` tag is to provide some way of automatically indexing fragment identifiers.

6.8.2.1 The n attribute

The `n` attribute specifies the name of the next generated fragment identifier. It is typically an alphabetic string followed by a two-digit number. A typical `<nextid>` tag might look like this:

```
<html>
<head>
<nextid n=DOC54>
</head>
...
```

An automatic-document generator might use the `nextid` information to successively name fragment identifiers `DOC54`, `DOC55`, and so forth within this document.

Chapter 7. Formatted Lists

Making information more accessible is the single most important quality of HTML and its progeny XHTML. The languages' excellent collection of text style and formatting tools help you organize your information into documents readers quickly understand, scan, and extract, possibly with automated browser agents.

Beyond embellishing your text with specialized text tags, HTML and XHTML also provide a rich set of tools that help you organize content into formatted lists. There's nothing magical or mysterious about lists. In fact, the beauty of lists is their simplicity. They're based on common list paradigms we encounter every day, such as an unordered laundry list, ordered instruction lists, and dictionary-like definition lists. All are familiar, comfortable ways of organizing content. All provide powerful means for quickly understanding, scanning, and extracting pertinent information from your web documents.

7.1 Unordered Lists

Like a laundry or shopping list, an unordered list is a collection of related items that have no special order or sequence. The most common unordered list you'll find on the Web is a collection of hyperlinks to other documents. Some common topic, like "Related Kumquat Lovers' Sites," allies the items in an unordered list, but they have no order among themselves.

7.1.1 The <ul> Tag

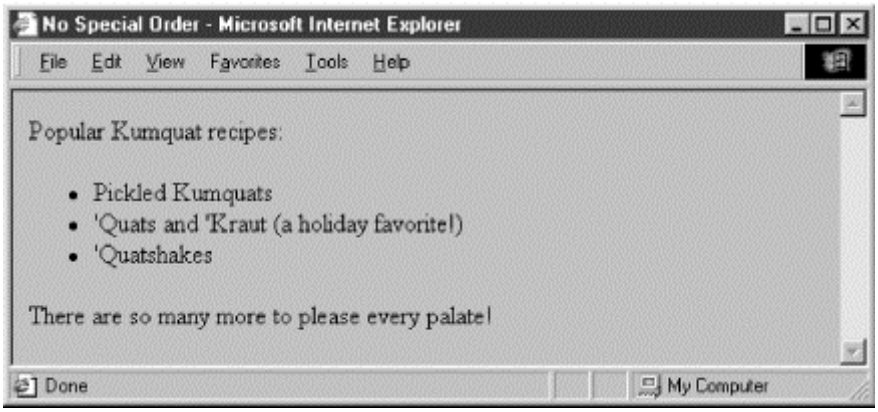
The `<ul>` tag signals to the browser that the following content, ending with the `</ul>` tag, is an unordered list of items. Inside, each item in the unordered list is identified by a leading `<li>` tag. Otherwise, nearly anything HTML/XHTML-wise goes, including other lists, text, and multimedia elements. Section 7.3

Typically, the browser adds a leading bullet character and formats each item on a new line, indented somewhat from the left margin of the document. The actual rendering of unordered lists, although similar for the popular browsers (see Figure 7-1), is not dictated by the standards, so you shouldn't get bent out of shape trying to attain exact positioning of the elements.

Here is an example XHTML unordered list, which Internet Explorer renders with bullets, as shown in Figure 7-1:

```
Popular Kumquat recipes:
<ul>
  <li>Pickled Kumquats</li>
  <li>'Quats and 'Kraut (a holiday favorite!)</li>
  <li>'Quatshakes</li>
</ul>
There are so many more to please every palate!
```

Figure 7-1. A simple unordered list



Tricky HTML authors sometimes use nested unordered lists, with and without `<li>`-tagged items, to take advantage of the automatic, successive indenting. You can produce some fairly slick text segments that way. Just don't depend on it for all browsers, including future ones. Rather, it's best to use the `border` property with a style definition in the paragraph (`<p>`) or division (`<div>`) tag to indent nonlist sections of your document (see Chapter 8).

Function:

Define an unordered list

Attributes:

CLASS	ONKEYUP
COMPACT	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	TYPE

End tag:

; never omitted

Contains:

list_content

Used in:

block

7.1.1.1 The type attribute

The graphical browsers automatically bullet each `<li>`-tagged item in an unordered list. Netscape and Internet Explorer use a solid circle, for example. Browsers that support HTML 3.2 and later versions, including 4.0 and 4.01, as well as XHTML 1.0, let you use the `type` attribute to specify which bullet symbol you'd rather have precede items in an unordered list. This attribute may have a value of either `disc`, `circle`, or `square`. All the items within that list will thereafter use the specified bullet symbol, unless an individual item overrides the list bullet type, as described later in this chapter.

With the advent of standard Cascading Style Sheets, the W3C has deprecated the `type` attribute in HTML 4 and in XHTML. Expect it to disappear.

7.1.1.2 Compact unordered lists

If you like wide open spaces, you'll hate the optional `compact` attribute for the `<ul>` tag. It tells the browser to squeeze the unordered list into an even smaller, more compact text block. Typically, the browser reduces the line spacing between list items. And it may reduce the indentation between list items, if it does anything at all with indentation (usually it doesn't).

Some browsers ignore the `compact` attribute, so you shouldn't depend on its formatting attributes. Also, the attribute is deprecated in the HTML 4 and XHTML standards, so it hasn't long to live.

7.1.1.3 The class and style attributes

The `style` and `class` attributes bring Cascading Style Sheet-based display control to lists, providing far more comprehensive control than you would get through individual attributes like `type`. Combine the `style` attribute with the `<ul>` tag, for instance, to assign your own bullet icon image, rather than use the default circle, disc, or square. The `class` attribute lets you apply the style of a predefined class of the `<ul>` tag to the contents of the unordered list. The value of the `class` attribute is the name of a style defined in some document-level or externally defined style sheet. For more information, see Chapter 8. Section 8.1.1 / Section 8.3

7.1.1.4 The lang and dir attributes

The `lang` attribute lets you specify the language used within a list, and `dir` lets you advise the browser as to which direction the text ought to be displayed. The value of the `lang` attribute is any of the ISO standard two-character language abbreviations, including an optional language modifier. For example, adding `lang=en-UK` tells the browser that the list is in English ("en") as spoken and written in the United Kingdom (UK). Presumably, the browser may make layout or typographic decisions based upon your language choice. Section 3.6.1.2

The `dir` attribute tells the browser which direction to display the list contents, from left-to-right (`dir=ltr`) like English or French, or from right-to-left (`dir=rtl`), such as with Hebrew or Chinese. Section 3.6.1.1

7.1.1.5 The id and title attributes

Use the `id` attribute to specially label the unordered list. An acceptable value is any quote-enclosed string that uniquely identifies the list and can later be used to unambiguously reference the list in a hyperlink target, for automated searches, as a style sheet selector, and for a host of other applications. Section 4.1.1.4

Use the optional title attribute and quote-enclosed string value also to identify the list. Unlike an `id` attribute, a `title` does not have to be unique. Section 4.1.1.5

id and `title` attributes

7.1.1.6 The event attributes

The many user-related events that may happen in and around a list, such as when a user clicks or double-clicks within its display space, are recognized by current browsers. With the respective "on" attribute and value, you may react to that event by displaying a user dialog box or activating some multimedia event. Section 12.3.3

7.2 Ordered Lists

Use an ordered list when the sequence of the list items is important. A list of instructions is a good example, as are tables of contents and lists of document footnotes or endnotes.

7.2.1 The <ol> Tag

The typical browser formats the contents of an ordered list just like an unordered list, except that the items are numbered instead of bulleted. The numbering starts at one and is incremented by one for each successive ordered list element tagged with `<li>`. Section 7.3

HTML 3.2 introduced a number of features that provide a wide variety of ordered lists. You can change the start value of the list and select any of five different numbering styles. Here is a sample XHTML ordered list:

```
<h3>Pickled Kumquats</h3>
Here's an easy way to make a delicious batch of pickled 'quats:
<ol>
  <li>Rinse 50 pounds of fresh kumquats</li>
  <li>Bring eight gallons white vinegar to rolling boil</li>
  <li>Add kumquats gradually, keeping vinegar boiling</li>
  <li>Boil for one hour, or until kumquats are tender</li>
  <li>Place in sealed jars and enjoy!</li>
</ol>
```

This is rendered by Netscape as shown in Figure 7-2.

Function:

Define an ordered list

Attributes:

CLASS	ONMOUSEDOWN
COMPACT	ONMOUSEMOVE
DIR	ONMOUSEOUT
ID	ONMOUSEOVER
LANG	ONMOUSEUP
ONCLICK	START
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	TYPE
ONKEYUP	

End tag:

; never omitted

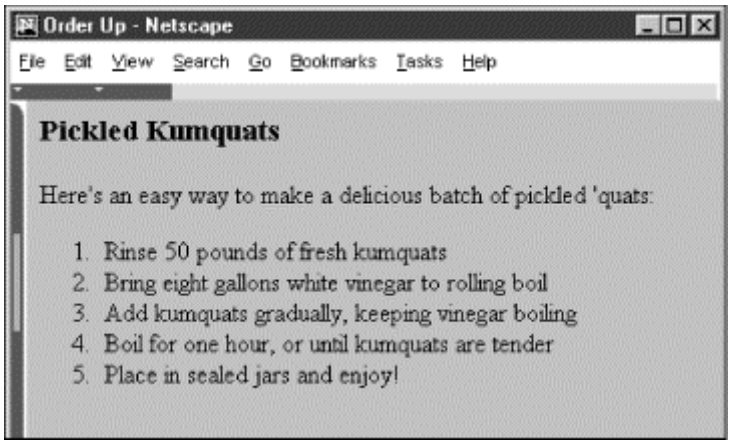
Contains:

list_content

Used in:

block

Figure 7-2. An ordered list



7.2.1.1 The start attribute

Normally, browsers automatically number ordered list items beginning with the Arabic numeral 1. The `start` attribute for the `<ol>` tag lets you change that beginning value. To start numbering a list at 5, for example:

```
<ol start=5>
  <li> This is item number 5.</li>
  <li> This is number six!</li>
  <li> And so forth...</li>
</ol>
```

7.2.1.2 The type attribute

By default, browsers number ordered list items with a sequence of Arabic numerals. Besides being able to start the sequence at some number other than 1, you also can use the `type` attribute with the `<ol>` tag to change the numbering style itself. With the `<ol>` tag, the `type` attribute may have a value of `A` for numbering with capital letters, `a` for numbering with lowercase letters, `I` for capital Roman numerals, `i` for lowercase Roman numerals, or `1` for common Arabic numerals. (See Table 7-1.)

Table 7-1, HTML Type Values for Numbering Ordered Lists

Type Value	Generated Style	Sample Sequence
`A`	Capital letters	A, B, C, D
`a`	Lowercase letters	a, b, c, d
`I`	Capital Roman numerals	I, II, III, IV
`i`	Lowercase Roman numerals	i, ii, iii, iv
`1`	Arabic numerals	1, 2, 3, 4

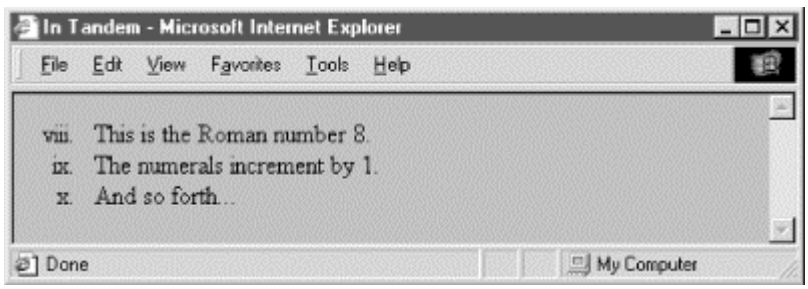
The `start` and `type` attribute extensions work in tandem. The `start` attribute sets the starting value of the item integer counter at the beginning of an ordered list. The `type` attribute sets the actual numbering style. For example, the following ordered list starts numbering items at 8, but because the style of numbering is set to i, the first number is the lowercase Roman numeral "viii." Subsequent items are numbered with the same style, and each value is incremented by 1 as shown in this HTML example:[1]

[1] Notice that we don't include the `</li>` end tag in the HTML example, but do in all the XHTML ones? Some end tags are optional with HTML, but must be included in all XHTML documents.

```
<ol start=8 type="i">
  <li> This is the Roman number 8.
  <li> The numerals increment by 1.
  <li> And so forth...
</ol>
```

The results are shown in Figure 7-3.

Figure 7-3. The start and type attributes work in tandem



The type and value of individual items in a list can be different from the list as a whole, described in Section 7.3.1. As mentioned earlier, the `start` and `type` attributes are deprecated in HTML 4 and XHTML. Consider using style sheets instead.

7.2.1.3 Compact ordered lists

Like the unordered list, the ordered list has an optional `compact` attribute that is deprecated in the HTML 4 and XHTML standards. Unless you absolutely need to use it, don't.

7.2.1.4 The class, dir, id, lang, event, style, and title attributes

These attributes are applicable as well with ordered lists and have identical effects as for unordered lists. Section 7.1.1.3 / Section 7.1.1.4 / Section 7.1.1.5 / Section 7.1.1.6

7.3 The <li> Tag

It should be quite obvious to you by now that the `<li>` tag defines an item in a list. It's the universal tag for list items in ordered (`<ol>`) and unordered (`<ul>`) lists, as we discussed earlier, and for directories (`<dir>`) and menus (`<menu>`), which we discuss in detail later in this chapter.

Function:

Define an item within an ordered, unordered, directory, or menu list

Attributes:

CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	TYPE
ONKEYUP	VALUE

End tag:

; often omitted in HTML

Contains:

flow

Used in:

list_content

Because the end of a list element can always be inferred by the surrounding document structure, most authors omit the ending `</li>` tags for their list elements. That makes sense because it becomes easier to add, delete, and move elements around within a list. However, XHTML requires the end tag, so it's best to get used to including it in your documents.

Although universal in meaning, there are some differences and restrictions to the use of the <li> tag for each list type. In unordered and ordered lists, what follows the <li> tag may be nearly anything, including other lists and multiple paragraphs. Typically, if it handles indentation at all, the browser successively indents nested list items, and the content in those items is justified to the innermost indented margin.

Directory and menu lists are another matter. They are lists of short items like a single word or simple text blurb and nothing else. Consequently, <li> items within <dir> and <menu> tags may not contain other lists or other block elements, including paragraphs, preformatted blocks, or forms.

Clean documents, fully compliant with the HTML and XHTML standards, should not contain any text or other document item inside the unordered, ordered, directory, or menu lists that is not contained within an <li> tag. Most browsers are tolerant of violations to this rule, but then you can't hold the browser responsible for compliant rendering for exceptional cases, either.

7.3.1 Changing the Style and Sequence of Individual List Items

Just as you can change the bullet or numbering style for all of the items in an unordered or ordered list, you also can change the style for individual items within those lists. With ordered lists, you also can change the value of the item number. As you'll see, the combinations of changing style and numbering can lead to a variety of useful list structures, particularly when included with nested lists. Do note, however, that the standards have deprecated these attributes in deference to their CSS counterparts.

7.3.1.1 The type attribute

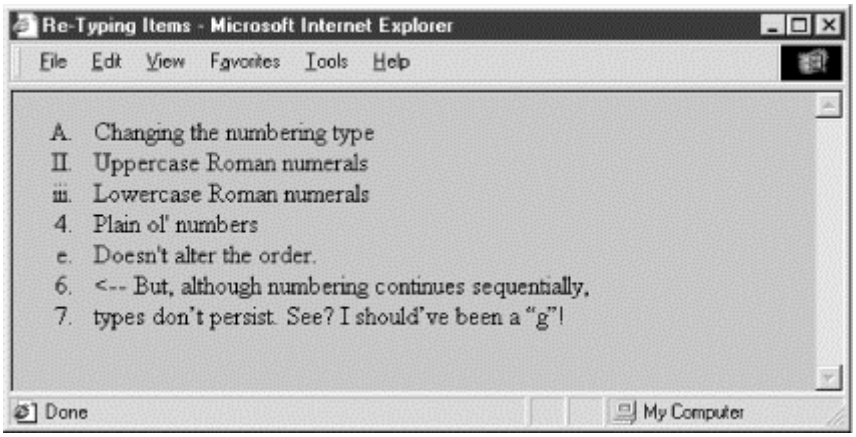
Acceptable values for the type attribute in the <li> tag are the same as the values for the appropriate list type: items within unordered lists may have their type set to circle, square, or disc, while items in an ordered list may have their type set to any of the values shown previously in Table 7-1.

Careful. With earlier browsers, such as Netscape Navigator and Internet Explorer versions 4 and earlier, a change in the bullet or numbering type in one list item similarly affected subsequent items in the list. Not so for HTML 4-compliant browsers, such as Netscape version 6 and Internet Explorer version 5! The type attribute effects are acute and limited to only the current <li> tag. Subsequent items revert to the default type; each must contain the specified type.

Figure 7-4 shows the effect changing the type for an individual item in an ordered list has on subsequent items, as rendered by Internet Explorer from the following source:

```
<ol>
  <li type=A>Changing the numbering type</li>
  <li type=I>Uppercase Roman numerals</li>
  <li type=i>Lowercase Roman numerals</li>
  <li type=1>Plain ol' numbers</li>
  <li type=a>Doesn't alter the order.</li>
  <li> &lt;-- But, although numbering continues sequentially,</li>
  <li> types don't persist. See? I should've been a "g"!</li>
</ol>
```

Figure 7-4. Changing the numbering style for each item in an ordered list



type changes the display style of the number, but not the value of the number.

You may use the style sheet-related style and class attributes to effect individual type changes in ordered and unordered lists that may or may not affect subsequent list items. Please see Chapter 8 for details, particularly Section 8.4.7.5.

7.3.1.2 The value attribute

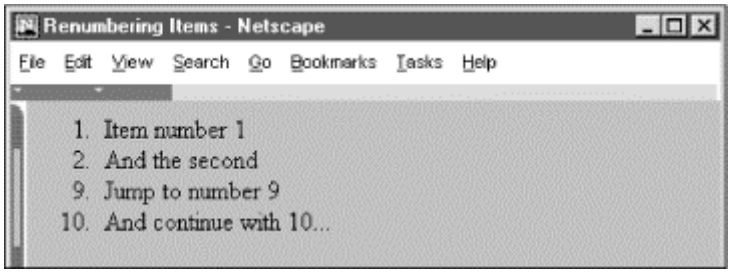
The `value` attribute changes the number of a specific list item and those that follow it. Since the ordered list is the only type with sequentially numbered items, the `value` attribute is valid only when used within an `<li>` tag inside an ordered list.

To change the current and subsequent numbers attached to each item in an ordered list, simply set the `value` attribute to any integer. The following source uses the `value` attribute to jump the numbering on items in an XHTML ordered list:

```
<ol>
  <li>Item number 1</li>
  <li>And the second</li>
  <li value=9> Jump to number 9</li>
  <li>And continue with 10...</li>
</ol>
```

The results are shown as rendered by Netscape in Figure 7-5.

Figure 7-5. The value attribute lets you change individual item numbers in an ordered list



7.3.1.3 The style and class attributes

The `style` attribute for the `<li>` tag creates an inline style for the elements enclosed by the tag, overriding any other style rule in effect. The `class` attribute lets you format the content according to a predefined class of the `<li>` tag; its value is the name of that class. Section 8.1.1 / Section 8.3

7.3.1.4 The class, dir, id, lang, event, style, and title attributes

These attributes can be applied to individual list items and have similar effects for ordered and unordered lists. Section 7.1.1.3 / Section 7.1.1.4 / Section 7.1.1.4 / Section 7.1.1.6

7.4 Nesting Lists

Except inside directories or menus, lists nested inside other lists are fine. Menu and directory lists can be embedded within other lists.

Indents for each nested list are cumulative, so take care not to nest lists too much; the list contents could quickly turn into a thin ribbon of text flush against the right edge of the browser document window.

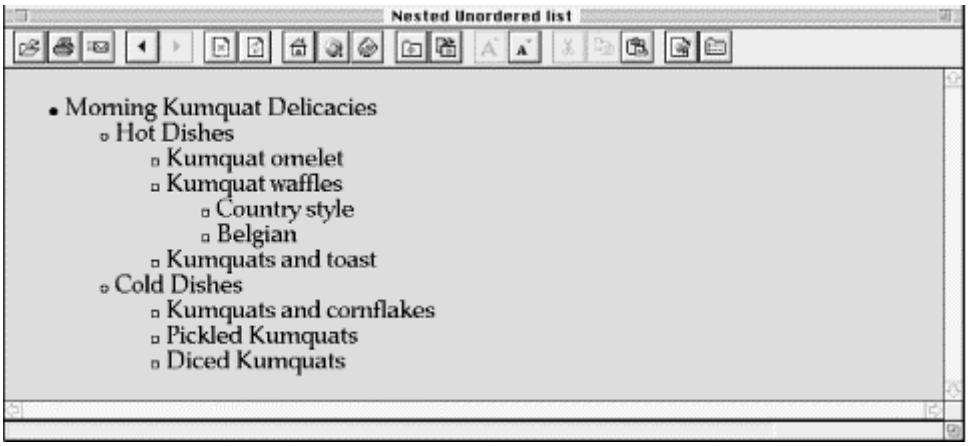
7.4.1 Nested Unordered Lists

The items in each nested unordered list may be preceded by a different bullet character at the discretion of the browser. For example, Internet Explorer Version 2 for Macintosh used an alternating series of hollow, solid circular, and square bullets for the various nests in the following source fragment as shown in Figure 7-6 (other browsers to date haven't been as inventive):

```
<ul>
  <li>Morning Kumquat Delicacies
  <ul>
    <li>Hot Dishes
    <ul>
      <li>Kumquat omelet</li>
      <li>Kumquat waffles
      <ul>
        <li>Country style</li>
        <li>Belgian</li>
      </ul>
      </li>
      <li>Kumquats and toast</li>
    </ul>
    </li>
```

```
    <li>Cold Dishes
    <ul>
      <li>Kumquats and cornflakes</li>
      <li>Pickled Kumquats</li>
      <li>Diced Kumquats</li>
    </ul>
   </li>
  </ul>
 </li>
</ul>
```

Figure 7-6. Bullets change for nested unordered list items



You can change the bullet style for each unordered list and even for individual list items (see the `type` attribute discussion earlier in this chapter), but the repertoire of bullets is limited. For example, Internet Explorer for Windows uses a solid disc regardless of the nesting level.

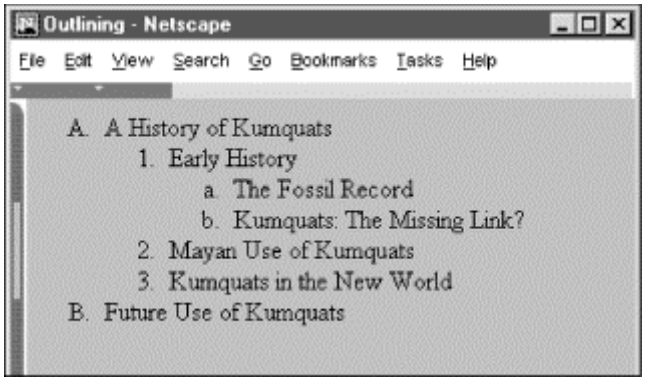
7.4.2 Nested Ordered Lists

By default, browsers number the items in ordered lists beginning with the Arabic numeral 1, nested or not. It would be great if the standards numbered nested ordered lists in some rational, consecutive manner. For example, the items in the second nest of the third main ordered list might be successively numbered "3.2.1," "3.2.2," "3.2.3," and so on.

With the `type` and `value` attributes, however, you do have a lot more latitude in how you create nested ordered lists. An excellent example is the traditional style for outlining, which uses the many different ways of numbering items offered by the `type` attribute (see Figure 7-7):

```
<ol type="A">
  <li>A History of Kumquats</li>
  <ol type="1">
    <li>Early History</li>
    <ol type="a">
      <li>The Fossil Record</li>
      <li>Kumquats: The Missing Link?</li>
    </ol>
    <li>Mayan Use of Kumquats</li>
    <li>Kumquats in the New World</li>
  </ol>
  <li>Future Use of Kumquats</li>
</ol>
```

Figure 7-7. The type attribute lets you do traditional outlining with ordered lists



7.5 Definition Lists

HTML and XHTML also support a list style entirely different from the ordered and unordered lists we've discussed so far: definition lists. Like the entries you find in a dictionary or encyclopedia, complete with text, pictures, and other multimedia elements, the definition list is the ideal way to present a glossary, list of terms, or other name/value lists.

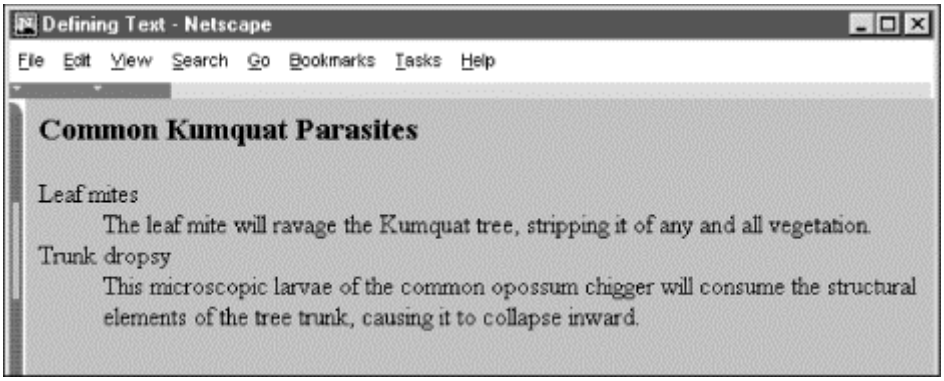
7.5.1 The <dl> Tag

The definition list is enclosed by the `<dl>` and `</dl>` tags. Within the tags, each item in a definition list is composed of two parts: a term followed by its definition or explanation. Instead of `<li>`, each item name in a `<dl>` list is marked with the `<dt>` tag, followed by the item's definition or explanation as it is marked by the `<dd>` tag.

Unless you change the display attributes with style sheet rules, browsers typically render the item or term name at the left margin and render the definition or explanation below it and indented. If the definition terms are very short (typically less than three characters), the browser may choose to place the first portion of the definition on the same line as the term. See how the source XHTML definition list below gets displayed by Netscape in Figure 7-8:

```
<h3>Common Kumquat Parasites</h3>
<dl>
  <dt>Leaf mites</dt>
  <dd>The leaf mite will ravage the Kumquat tree, stripping it
      of any and all vegetation.</dd>
  <dt>Trunk dropsy</dt>
  <dd>This microscopic larvae of the common opossum
      chigger will consume the structural elements of the
      tree trunk, causing it to collapse inward.</dd>
</dl>
```

Figure 7-8. A definition list as presented by Netscape



As with other list types, you can add more space between the definition list items by inserting paragraph `<p>` tags at the end of their content or by defining a spacious style for the respective tags.

7.5.1.1 More compact definition lists

The `<dl>` tag supports the `compact` attribute, advising the browser to make the list presentation as small as possible. Few browsers, if any, honor this attribute, and it has been deprecated in HTML 4 and XHTML.

7.5.1.2 The class, dir, id, lang, style, title, and event attributes

The many other attributes for the `<dl>` tag should be quite familiar by now. The `style` and `class` attributes, of course, let you control the display style; the `id` and `title` tag attributes let you uniquely label its contents; the `dir` and `lang` attributes let you specify its native language; and the many on-event attributes let you react to user-initiated mouse and keyboard actions on the contents. Not all are implemented by the currently popular browsers for this tag or for many others. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.4 / Section 4.1.1.5 / Section 8.1.1 / Section 8.3 / Section 12.3.3

<dl>

Function:

Define a definition list

Attributes:

CLASS	ONKEYUP
COMPACT	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	

EndTag:

</dl>; never omitted

Contains:

dl_content

Used in:

block

7.5.2 The <dt> Tag

This <dt> tag defines the term component of a definition list. It is valid only when used within a definition <dl> list preceding the term or item, before the <dd> tag and the term's definition or explanation.

Traditionally, the definition term that follows the <dt> tag is short and sweet - a word or few. Technically, it can be any length. If long, the browser may exercise the option of extending the item beyond the display window, or wrap it onto the next line where the definition begins.

Since the end of the <dt> tag immediately precedes the start of the matching <dd> tag, it is unambiguous and so not required. However, the XHTML standard insists that it be present. So get used to including it in your documents.

7.5.2.1 Formatting text with <dt>

In practice, browsers are either too lenient or too dumb to enforce the rules, so some tricky HTML authors misuse the <dt> tag to shift the left margin right and left, respectively, for fancy text displays. (Remember, tab characters and leading spaces don't usually work with regular text.) We don't condone violating the HTML, and certainly not the XHTML standard, and caution you once again about tricked-up documents. Use style sheets instead.

<dt>

Function:

Define a definition list term

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

</dt>; may be omitted in HTML

Contains:

text

Used in:

dl_content

7.5.2.2 The class, dir, id, lang, style, title, and event attributes

The `<dt>` tag supports the standard HTML 4/XHTML tag attributes. The `style` and `class` attributes, of course, let you control the display style; the `id` and `title` tag attributes let you uniquely label its contents; the `dir` and `lang` attributes let you specify its native language; and the many on-event attributes let you react to user-initiated mouse and keyboard actions on the contents. Not all are implemented by the currently popular browsers for this tag or for many others. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.4 / Section 4.1.1.5 / Section 8.1.1 / Section 8.3 / Section 12.3.3

7.5.3 The <dd> Tag

The `<dd>` tag marks the start of the definition portion of an item in a definition list. According to the HTML and XHTML standards, `<dd>` belongs only inside a definition `<dl>` list, immediately following the `<dt>` tag and term and preceding the definition or explanation.

The content that follows the `<dd>` tag may be any HTML construct, including other lists, block text, and multimedia elements. Although treating it otherwise identically as conventional content, browsers typically indent definition list `<dd>` definitions. And since the start of another term and definition (`<dt>`) or the required end tag of the definition (`</dl>`) unambiguously terminates the preceding definition, the `</dd>` tag is not needed and its absence makes your source text more readable. However, and once again, XHTML insists that the end tag appear in your documents, so you may as well get used to adding `</dd>` to your documents.

<dd>

Function:

Define a definition list term

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

</dd>; always omitted in HTML

Contains:

flow

Used in:

dl_content

7.5.3.1 The class, dir, id, lang, style, title, and event attributes

The `<dt>` tag supports the standard tag attributes. The `style` and `class` attributes, of course, let you control the display style; the `id` and `title` tag attributes let you uniquely label its contents; the `dir` and `lang` attributes let you specify its native language; and the many on-event attributes let you react to user-initiated mouse and keyboard actions on the contents. Not all are implemented by the currently popular browsers for this tag or for many others. / Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.4 / Section 4.1.1.5 / Section 8.1.1 / Section 8.3 / Section 12.3.3

7.6 Appropriate List Usage

In general, use unordered lists for:

- Hotlists and other link collections
- Short, nonsequenced groups of text
- Emphasizing the high points of a presentation

In general, use ordered lists for:

- Tables of contents
- Instruction sequences
- Sets of sequential sections of text
- Assigning numbers to short phrases that can be referenced elsewhere

In general, use definition lists for:

- Glossaries
- Custom bullets (make the item after the <dt> tag an icon-sized bullet image)
- Any list of name/value pairs

7.7 Directory Lists

The directory list is a specialized form of the unordered list. It has been deprecated in the HTML 4 and XHTML standards. We don't recommend that you use it at all. Section 7.1.1

7.7.1 The <dir> Tag (Deprecated)

The designers of HTML originally dedicated the <dir> tag for displaying lists of files. As such, the browser, if it treats <dir> and <ul> differently at all (most don't), expects the various list elements to be quite short, possibly no longer than 20 characters or so. Some browsers display the elements in a multicolumn format and may not use a leading bullet.

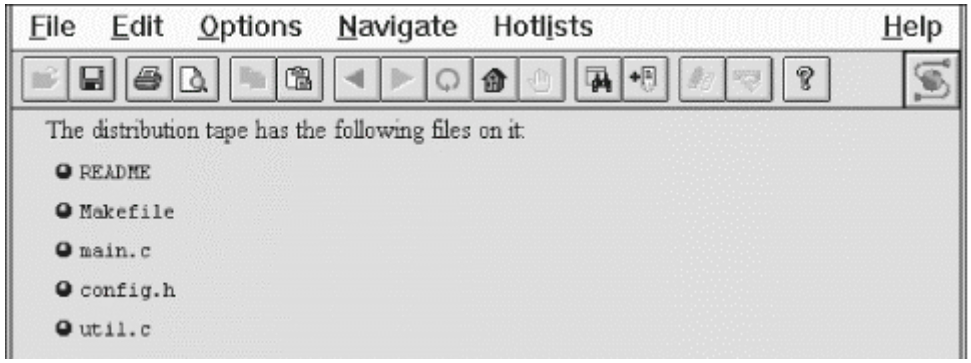
As with the unordered list, define directory list items with the <li> tag. When used within a directory list, however, the <li> tag may not contain any block element, including paragraphs, other lists, preformatted text, or forms.

The following example puts the directory tag to its traditional task of presenting a list of filenames:

```
The distribution tape has the following files on it:
<dir>
  <li><code>README</code></li>
  <li><code>Makefile</code></li>
  <li><code>main.c</code></li>
  <li><code>config.h</code></li>
  <li><code>util.c</code></li>
</dir>
```

Notice that we use the <code> tag to ensure that the filenames would be rendered in an appropriate manner (see Figure 7-9 as rendered by the now ancient Mosaic browser).

Figure 7-9. An example <dir> list



7.7.1.1 The <dir> attributes

The attributes for the <dir> tag are identical to those for <ul> with the same effects.

<dir>

Function:

Define a directory list

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

</dir>; never omitted

Contains:

list_content

Used in:

block

7.8 Menu Lists

The menu list is yet another specialized form of the unordered list. It, too, like <dir> is deprecated in the HTML 4 and XHTML standards, so we don't recommend using it. Section 7.1.1

7.8.1 The <menu> Tag (Deprecated)

The <menu> tag displays a list of short choices to the reader, such as a menu of links to other documents. The browser may use a special (typically more compact) representation of items in a menu list compared with the general unordered list, or even use some sort of graphical pull-down menu to implement the menu list. If the list items are short enough, the browser may even display them in a multicolumn format and may not precede each list item with a bullet.

Like an unordered list, define the menu list items with the <li> tag. When used within a menu list, however, the <li> tag may not contain any block element, including paragraphs, other lists, preformatted text, or forms.

Compare the source text below and the ancient Mosaic display (Figure 7-10) with the directory (Figure 7-9) and unordered (Figure 7-1) list displays presented earlier in the chapter:

```
Some popular kumquat recipes include:
<menu>
  <li>Pickled Kumquats</li>
  <li>'Quats and 'Kraut (a holiday favorite!)</li>
  <li>'Quatshakes</li>
</menu>
There are many more to please every palate!
```

<menu>

Function:

Define a menu list

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

</menu>; never omitted

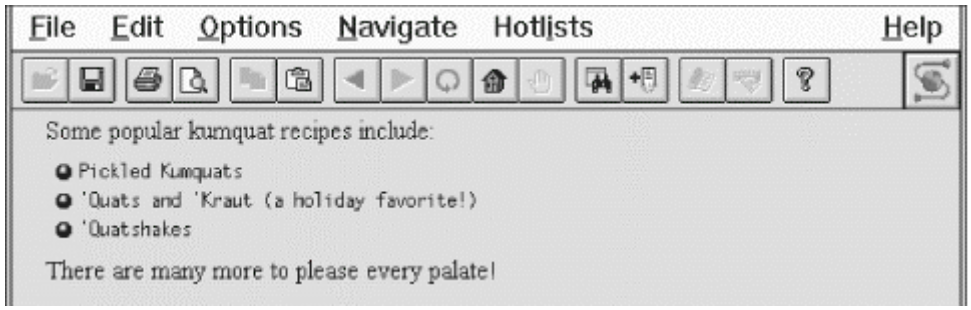
Contains:

list_content

Used in:

block

Figure 7-10. Sample <menu> list



Chapter 8. Cascading Style Sheets

Style sheets are the way publishing professionals manage the overall "look" of their publications - backgrounds, fonts, colors, and so on - from a single page to huge collections of documents. Most desktop publishing software support style sheets, as do the popular word processors. All desktop publishers and graphic designers worth their salt are out there making web pages. So the cry-to-arms was inevitable: "Whaddaya mean HTML has no style sheets?!"

From its origins, HTML focused on content over style. Authors are encouraged to worry about providing high quality information and leave it to the browser to worry about presentation. We strongly urge you, too - as we do throughout this book - to adopt that philosophy in your documents, especially those destined for the World Wide Web. Don't mistake style for substance.

However, while use of the `<font>` tag and related attributes like `color` produce acute presentation effects, style sheets, when judiciously applied, bring consistency and order to whole document collections, as well as to individual documents. Remember, too, that presentation is for the benefit of the reader. Even the original designers of HTML understood the interplay between style and readability. For instance, readers can quickly identify section heads in a document when they are enclosed in header tags like `<h2>`, which the modern browsers present in large and often bold type. Style sheets extend that presentation with several additional effects, including colors, a wider selection of fonts, even sounds so that users can even better distinguish elements of your document. But most importantly, style sheets let you control the presentation attributes for all the tags in a document - for a single document or a whole collection of many documents, and from a single master.

In early 1996, the World Wide Web Consortium put together a draft proposal defining Cascading Style Sheets (CSS) for HTML. This draft proposal quickly matured into a recommended standard, which the commercial browser manufacturers were quick to exploit. In mid-1998, the W3C extended the original specification to create CSS2 which includes presentation standards for a variety of media besides the familiar onscreen browser, along with a several other enhancements.

Up to now, however, no browser or web agent fully complies with the CSS2 standard. Since we realize that eventual compliance with the W3C standard is likely, we'll cover all the components of the standard in this chapter, even if they are not yet supported by any browser. As always, we'll denote clearly what is real, what is proposed, and what is actually supported.

What we can't do is tell you everything the CSS2 standard provides. Like JavaScript, the Cascading Style Sheet standard deserves a Definitive Guide of its own. Rather, we focus here on the elements of style sheets that impact HTML and XHTML in general and the popular GUI-based browsers, Internet Explorer and Netscape Navigator, in particular. These encompass the majority of the CSS2 standard. What's left out are discussions of other media. We tell you how to tailor your documents to other media, but we don't go into the specifics, such as the CSS2 properties that control paging devices like printers or aural style sheets that govern the presentation of content through speech synthesis. Don't get us wrong; these are fascinating and important topics. They just go beyond our charter.

8.1 The Elements of Styles

At the simplest level, a style is nothing more than a rule that tells the browser how to render[1] a particular tag's contents. Each tag has a number of style properties associated with it, whose values define how that tag is rendered by the browser. A rule defines a specific value for one or more properties of a tag. For example, most tags have a `color` property, the value of which defines the color Netscape or Internet Explorer may use to display the contents of the tag. Other properties include fonts, line spacing, margins, borders, sound volume, and voice, which we describe in detail later in this chapter.

> [1] We explicitly avoided the term "display" here because it connotes visual presentation, whereas the CSS2 standard works hard to suggest many different ways of presenting the tagged contents of a document.

There are three ways to attach a style to a tag: inline styles, document-level styles, and external style sheets. You may use one or more style sheets for your documents. The browser either merges the style definitions from each style or redefines the style characteristic for a tag's contents. Styles from these various sources are applied to your document, combining and defining style properties that cascade from external style sheets through local document styles, ending with inline styles. This cascade of properties and style rules gives rise to the standard's name: Cascading Style Sheets.

We cover the syntactic basics of the three style sheet techniques here. We delve more deeply into the appropriate use of inline, document-level, and external style sheets at the end of this chapter.

8.1.1 Inline Styles: The style Attribute

The inline style is the simplest way to attach a style to a tag - just include a `style` attribute with the tag along with a list of properties and their values. The browser uses those style properties and values to render the contents of just this instance of the tag.

For instance, the following style tells the browser to display the level-1 header text, "I'm so bluuuuoooo!", not only in the `<h1>` tag style characteristic of the browser, but also in the color blue and italicized (if the browser is capable):

```
<h1 style="color: blue; font-style: italic">I'm so bluuuuoooo!</h1>
```

This type of style definition is called "inline" because it occurs with the tag as it appears in the document. The scope of the style covers the contents of that tag only. Since inline styles are sprinkled throughout your document, they can be difficult to maintain. Use the `style` attribute sparingly and only in those rare circumstances when you cannot achieve the same effects otherwise.

8.1.2 Document-Level Style Sheets

The real power of style sheets becomes more evident when you place a list of presentation rules within the head of a document. Enclosed within their own `<style>` and `</style>` end tags, so-called "document-level" style sheets affect all the same tags within that document, except for tags that contain an overriding inline `style` attribute.[2]

[2] XHTML-based document-level style sheets get specially enclosed in CDATA sections of your documents. See Chapter 16 for details.

<style>

Function:

Define a document-level style sheet

Attributes:

DIR
LANG
MEDIA
TITLE
TYPE

End tag:

</style>; rarely omitted in HTML

Contains:

styles

Used in:

head_content

The `<style>` tag must appear within the `<head>` of a document. Everything between the `<style>` and `</style>` tags is considered part of the style rules to be applied to the document. To be perfectly correct, the contents of the `<style>` tag are not HTML or XHTML and are not bound by the normal rules for markup content. The `<style>` tag, in effect, lets you insert foreign content into your document that the browser uses to format your tags.

For example, a styles-conscious browser will display the contents of all <h1> tags as blue, italic text in a document that has the following document-level style sheet definition in its head:[3]

[3] This is an HTML example. An XHTML document would enclose styles in a CDATA section instead of in HTML comments. See Section 16.3.7 for details.

```
<head>
<title>All True Blue</title>
<style type="text/css">
  <!--
  /* make all level-1 headers blue */
  h1 {color: blue; font-style: italic}
  -->
</style>
</head>
<body>
<h1>I'm so bluuuuoooo!</h1>
...
<h1>I am ba-looooo, tooooo!</h1>
```

8.1.2.1 The type attribute

The type attribute defines the types of styles you are including within the tag. Cascading style sheets are all type text/css ; JavaScript style sheets use the type text/javascript. You may omit the type attribute and hope the browser will figure out the kind of styles you are using. We prefer to include the type attribute so that there is no opportunity for confusion. Section 12.4

8.1.2.2 The media attribute

HTML and XHTML documents can wind up in the strangest places these days, even on cellular phones. To help the browser figure out the best way to render your documents, the HTML 4 and XHTML standards let you include the media attribute within the <style> tag. The value of this attribute is the medium for which this document is intended; the default value is screen . Other defined values are tty, tv, projection, handheld, print, braille, aural, and all.

A document intended for multiple media can use a quote-enclosed, comma-separated list of media types as the value of this attribute. For example:

```
<style type="text/css" media="screen,print">
```

tells the browser that your document is layed out using CSS for display on both print pages and on a computer or other intelligent display screen.

By specifying media, the browser applies the styles you define within the <style> tag only if the document is being displayed on that medium. Thus, the browser would not apply our example set of styles designed for media="screen,print" if the user is, for instance, connected to the Web with a handheld computer.

The CSS2 standard also lets you define media-specific style sheets through its extension to the @import at-rule and through the @media at-rule. More about this in Section 8.1.4.

8.1.2.3 The dir, lang, and title attribute

As with any HTML/XHTML element, you can associate a descriptive title with the <style> tag. If the browser chose to display this title to the user, it would use the values of the dir and lang attributes to correctly render the title. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.5

8.1.3 External Style Sheets

You may also place style definitions, like our document-level style sheet example for the <h1> tags, into a text file with the MIME type of text/css and import this "external" style sheet into your documents. Because an external style sheet is a separate file and is loaded by the browser over the network, you can store it anywhere, reuse it often, and even use others' style sheets. But most important, external style sheets give you the power to influence the display styles not only of all related tags in a single document, but for an entire collection of documents.

For example, suppose we create a file named gen_styles.css containing the style rule:

```
h1 {color: blue; font-style: italic}
```

For each and every one of the documents in our collections, we can tell the browser to read the contents of the gen_styles.css file, which in turn will color all the <h1> tag contents blue and render the text in italic. Of course, that will be true only if the user's machine is capable of these style tricks, they are using a styles-conscious browser like Netscape or Internet Explorer, and the style isn't overridden by a document-level or inline style definition.

You can load external style sheets into your document in two different ways: linked or imported.

8.1.3.1 Linked external style sheets

One way to load an external style sheet is to use the <link> tag:

```
<head>
<title>Style linked</title>
<link rel=stylesheet type="text/css"
      href="http://www.kumquats.com/styles/gen_styles.css"
      title="The blues">
</head>
<body>
<h1>I'm so bluuuuoooo!</h1>
...
<h1> I am ba-looooo, tooooo!<h1>
```

Recall that the <link> tag creates a relationship between the current document and some other document on the Web. In the example, we tell the browser that the document named in the href attribute is a stylesheet and that its contents conform to the CSS2 standard, as indicated by the type attribute. We also provide a title for the style sheet, making it available for later reference by the browser. Section 6.7.2

The <link> tag must appear in the <head> of a document. The URL of the style sheet may be absolute or relative to the document's base URL. The type may also be text/javascript, indicating (for Netscape only) that the style rules are written in JavaScript instead of CSS2 syntax. Section 12.4

8.1.3.2 Imported external style sheets

The second technique for loading an external style sheet imports the files with a special command (aka "at-rule") within the <style> tag:

```
<head>
<title>Imported style sheet</title>
<style>
  <!--
    @import url(http://www.kumquats.com/styles/gen_styles.css);
    @import "http://www.kumquats.com/styles/spec_styles.css";
    body {background: url(backgrounds/marble.gif)}
  -->
</style>
</head>
```

The @import at-rule expects a single URL parameter that names the network path to the external style sheet. The URL may be a string enclosed in double-quotes and ending with a semicolon, or be the contents of the url keyword, enclosed in parentheses and with a trailing semicolon. The URL may be absolute or relative to the document's base URL.

The @import at-rule must appear *before* any conventional style rules, either in the <style> tag or in an external style sheet. Otherwise, the standard insists that the browser ignore the errant @import. By first importing all the various style sheets, then processing document-level style rules, the CSS2 standard cascades: the last one standing wins. Section 8.4.1.4

The @import at-rule can appear in a document-level style definition or even in another external style sheet, letting you create nested style sheets.

8.1.4 Media-Specific Styles

Besides the media attribute for the <style> tag, the CSS2 standard has two other features that let you apply different style sheets depending on the agent or device that will render your document. This way, for instance, you can have one style or whole style sheet take effect when your document gets rendered on a computer screen, and another set of styles for when the contents get punched out on a braille printer. And what about those cell phones on the Web?

Like the media attribute for the style tag that affects the entire style sheet, you can specify whether the user's document processor[4] will load and use an imported style sheet.

> [4] A.k.a. "user agent." Web documents get rendered on all kinds of devices these days, including the popular browser, braille printers, televisions, and projectors, to name just a few.

Do that by adding a media-type keyword or a series of comma-separated keywords to the end of the @import at-rule. For instance, the following example lets the user-agent decide to import and use the speech-synthesis style sheet or a common PC display and print style sheet if it is able to render the specified media types:

```
@import url(http://www.kumquats.com/styles/visual_styles.css) screen,print;
@import "http://www.kumquats.com/styles/speech_styles.css" aural;
```

The CSS2 media types include all, aural (speech synthesizers, for example), braille (tactile), embossed (braille printers), handheld, print, projection, screen, tty (fixed-width fonts), and tv.

Another CSS2 way to select media is through the explicit @media at-rule, which lets you include media-specific rules within the same style sheet, either at the document level or in an external style sheet. At the document level, like @import, the @media at-rule must appear within the <style> tag contents. And the at-rules may not appear within another rule. Unlike @import, @media may appear subsequent to other style rules, and indeed its style-rule contents may override previous rules according to the cascading standard.

The contents of @media include one or more comma-separated media-type keywords followed by a curly-braces ({}) enclosed set of style rules. For example:

```
body {background: white}
@media tv, projection        {
     body {background: lt_blue}
     }
```

The general style rule for the document's body background color of white gets changed to light blue by the one within the @media at-rule, but only if the document gets rendered on a television or projection system instead of some other medium. (Notice the extra set of curly braces that contain the @media style rules?)

8.1.5 Linked Versus Imported Style Sheets

At first glance, it may appear that linked and imported style sheets are equivalent, using different syntax for the same functionality. This is true if you use just one <link> tag in your document. However, special CSS2-standard rules come into play if you include two or more <link> tags within a single document, even though the current browsers don't abide by the rules yet.

With one <link> tag, the browser should load the styles in the referenced style sheet and format the document accordingly, with any document-level and inline styles overriding the external definitions. With two or more <link> tags, the browser should present the user with a list of all the <link>ed style sheets. The user then selects one of the linked sheets, which the browser loads and uses to format the document; the other <link>ed style sheets get ignored.

On the other hand, the styles-conscious browser merges, as opposed to separating, multiple @imported style sheets to form a single set of style rules for your document. The last imported style sheet takes precedence if there are duplicate definitions among the style sheets. Hence, if the external gen_styles.css style sheet specification tells the browser to make <h1> contents blue and italic, and spec_styles.css tells the browser to make <h1> text red, then the <h1> tag contents will appear red and italic. And if we later define another color, say yellow, for <h1> tags in a document-level style definition, the <h1> tags will all be yellow, and italic. Cascading effects. See?

In practice, the popular browsers treat linked style sheets just like imported ones by cascading their effects. The browsers do not currently let you choose from among linked choices. Imported styles override linked external styles, just as the document-level and inline styles override external style definitions. To bring this all together, consider the example:

```
<html>
<head>
<link rel=stylesheet href=sheet1.css type=text/css>
<link rel=stylesheet href=sheet2.css type=text/css>
<style>
<!--
  @import url(sheet3.css);
  @import url(sheet4.css);
-->
</style>
</head>
```

Using the CSS2 model, the browser should prompt the user to choose sheet1.css or sheet2.css. It should then load the selected sheet, followed by sheet3.css and sheet4.css. Duplicate styles defined in sheet3.css or sheet4.css and in any inline styles will override styles defined in the selected sheet. In practice, the popular browsers cascade the style sheet rules as defined in the example order sheet1 through sheet4.

8.1.6 Limitations of Current Browsers

Internet Explorer 4 and Netscape Navigator 4 and beyond support the <link> tag to apply an external style sheet to a document. Neither Netscape Navigator nor Internet Explorer support multiple <link>ed style sheets as proposed by the CSS2 standard. Instead, they cascade all the <link>ed style sheets, with rules in later sheets overriding rules in earlier sheets.

Netscape Navigator ignores all at-rules and their contents, including @import and @media, but does process other style rules that you may include before or after the at-rule within the <style> tag. Internet Explorer honors the @import as well as the @media at-rules, for both document-level and external sheets, allowing sheets to be nested.

Achieving media-specific styles through external style sheets with current Netscape browsers is hopeless. Assume, therefore, that most people who have Netscape will render your documents on a common PC screen, so make that medium the default one. Then embed all other media-specific styles, such as those for print or braille, within `@media` at-rules, so that Internet Explorer and other CSS-compliant agents will properly select styles based on the rendering medium. The only other alternative is to create media-specific `<style>` tags within each document. Run, do not walk, away from that idea.

We just hope the CSS2 standard will prevail soon so that style sheets, already mystifying to most, will become at least that much less confusing.

8.1.7 Style Comments

Comments are welcome inside the `<style>` tag and in external style sheets, but don't use standard HTML comments; style sheets aren't HTML. Rather, enclose style comments beginning with the sequence `/*` and ending with `*/`, as we did in the example in Section 8.1.2. (Those of you who are familiar with the C programming language will recognize these comment markings.) Use this comment syntax for both document-level and external style sheets. Comments may not be nested.

We recommend documenting your styles whenever possible, especially in external style sheets. Whenever the possibility exists that your styles may be used by other authors, comments make it much easier to understand your styles.

8.1.8 Handling Style-less Browsers

In our document-level style examples, you probably noticed that we placed the style definition inside a comment tag (`<!--` and `-->`). That's because although the older, style-less browsers will ignore the `<style>` tag itself, they will display the style definitions. Needless to say, your documents will not go over well when the first half of the display contains all your style rules.

The newer, styles-conscious browsers ignore HTML comments within a `<style>` tag. Style-less browsers may be with us for some time to come, so it's probably best to place your document-level style rules inside comments. HTML comments should not be used in external style sheets.

For XHTML, document-level styles must be enclosed in a CDATA section instead of in HTML comments. See Section 16.3.7 for details.

8.1.9 Style Precedence

You may import more than one external style sheet and combine them with document-level and inline style effects in many different ways. Their effects cascade (hence, the name, of course). You may specify the font type for our example `<h1>` tag, for instance, in an external style definition, whereas its color may come from a document-level style sheet.

Style sheet effects are not cumulative, however: of the many styles which may define different values for the same property - colors for the contents of our example tag, for instance - the one that takes precedence can be found by following these rules, listed here in order.

Sort by origin

> A style defined "closer" to a tag takes precedence over a more "distant" style. So an inline style takes precedence over a document-level style, which takes precedence over the effects of an external style.

If more than one applicable style exists, sort by class

> Properties defined as a class of a tag (see Section 8.3) take precedence over a property defined for the tag in general.

If multiple styles still exist, sort by specificity

> The properties for a more specific contextual style (see Section 8.2.3) take precedence over properties defined for a less specific context.

If multiple styles still exist, sort by order

> The property specified latest takes precedence.

The relationship between style properties and conventional tag attributes is almost impossible to predict. Style sheet-dictated background and foreground colors - whether defined externally, at the document level, or inline - override the various `color` attributes that may appear within a tag. But the `align` attribute of an inline image usually takes precedence over a style-dictated alignment.

There is an overwhelming myriad of style and tag presentation-attribute combinations. You need a crystal ball to predict which combination wins and which loses the precedence battle. The rules of redundancy and style versus attribute precedence are not clearly elucidated in the W3C CSS2 standard, nor is there a clear pattern of precedence implemented in the styles-conscious browsers. This is particularly unfortunate since there will be an extended period, perhaps several years, in which users may or may not use a styles-conscious browser. Authors will have to implement both styles and non-style presentation controls to achieve the same effects.

Nonetheless, our recommendation is to run - as fast as you can - away from one-shot, inline, localized kinds of presentation effects like those afforded by the `<font>` tag and `color` attribute. They have served their temporary purpose; it's now time to bring consistency (without the pain!) back into your document presentation. Use styles. It's the HTML way.

8.2 Style Syntax

The syntax of a style, its "rule" as you may have gleaned from our previous examples, is very straightforward.

8.2.1 The Basics

A style rule is made up of at least three basic parts: a *selector*, which is the name of the markup element that the style rule affects, followed by a curly brace (`{}`) enclosed, semicolon-separated list of one or more style `property:value` pairs:

```
selector {property1:value1; property2:value1 value2 value3; ...}
```

For instance, we might define the color for the contents of all the level-1 header elements of our document:

```
h1 {color: green}
```

In this example, `h1` is the selector which is also the name of the level-1 header element, color is the style property, and green is the value. Neat and clean. Try it. It really works!

Properties require at least one value, but may include two or more values. Separate multiple values with a space, as is done for the three values that define `property2` in our first example. Some properties require that multiple values be separated with commas.

Current styles-conscious browsers ignore letter case in any element of a style rule. Hence, `H1` and `h1` are the same selector, and `COLOR`, `color`, `ColOR`, and `cOLor` are equivalent properties. At one time, convention dictated that HTML authors write selector names in uppercase characters, such as `H1`, `P`, and `STRONG`. This convention is still common and is used in the W3C's own CSS2 document.

Current standards dictate, however, particularly for XML-compliant documents, that element names be capitalized exactly as defined by their respective DTDs. With XHTML, for instance, all element names (`h1`, `p`, or `strong`, for instance) are lowercase, so their respective CSS2 selectors must be in lowercase. We'll abide by these latter conventions.

Any valid element name (a tag name minus its enclosing < and > characters and attributes) can be a selector. You may include more than one tag name in the list of selectors, as we explain in the following sections.

8.2.2 Multiple Selectors

When separated by commas, all the elements named in the selector list are affected by the property values in the style rule. This can make life very easy for authors. For instance:

```
h1, h2, h3, h4, h5, h6 {text-align: center}
```

does exactly the same thing as:

```
h1 {text-align: center}
h2 {text-align: center}
h3 {text-align: center}
h4 {text-align: center}
h5 {text-align: center}
h6 {text-align: center}
```

Both styles tell the browser to center the contents of headers levels 1-6. Clearly, the first version is easier to type, understand, and modify. And it takes less time and fewer resources to transmit across a network, albeit a trivial consideration in this instance.

8.2.3 Contextual Selectors

Normally, the styles-conscious browser applies document-level or imported styles to a tag's contents wherever they appear in your document, without regard to context. However, the CSS2 standard defines a way to have a style applied only when a tag occurs within a certain context within a document, such as when it is nested within other tags.

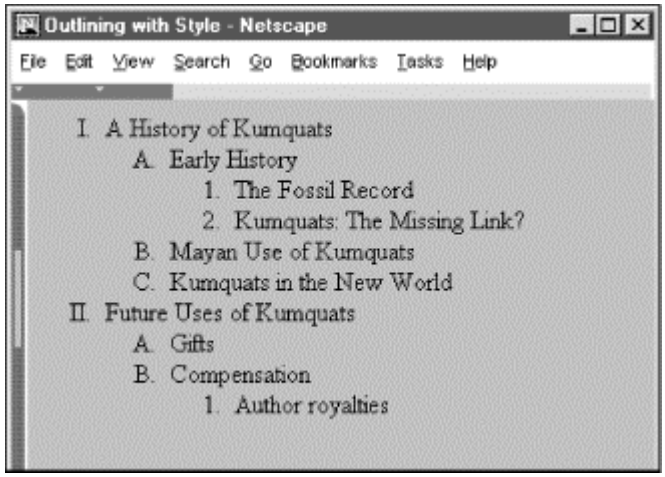
To create a contextual selector, list the tags in the order in which they should be nested in your document, outermost tag first. When that nesting order is encountered by the browser, the style properties will be applied to the last tag in the list.

For example, here's how you might use contextual styles to define the classic numbering sequence used for outlines: upper-case Roman numerals for the outer level, capital letters for the next level, Arabic numerals for the next, and lower-case letters for the innermost level:

```
ol li {list-style: upper-roman}
ol ol li {list-style: upper-alpha}
ol ol ol li {list-style: decimal}
ol ol ol ol li {list-style: lower-alpha}
```

According to the example style sheet, when the styles-conscious browser encounters the `<li>` tag nested within one `<ol>` tag, it uses the `upper-roman` value for the `list-style` property of the `<li>` tag. When it sees an `<li>` tag nested within two `<ol>` tags, the same browser will use the `upper-alpha list-style`. Nest an `<li>` tag within three and four `<ol>` tags, and you'll see the `decimal` and `lower-alpha list-styles` used, respectively. That's exactly what Netscape Navigator does, as shown in Figure 8-1 (Internet Explorer does the same thing). Compare Figure 8-1 with using the ordered list tag's `type` attribute to achieve similar effects as shown in Figure 7-7.

Figure 8-1. Nested ordered list styles



Similarly, you may impose a specific style on tags related only by context. For instance, this contextual style definition will color the emphasis tag's (`<em>`) contents red only when it appears inside a level-1 header tag (`<h1>`), and not elsewhere in the document:

```
h1 em {color: red}
```

If there is potential ambiguity between two contextual styles, the more specific context prevails.

Like individual tags, you may also have several contextual selectors mixed with individual selectors, each and all separated by commas, sharing the same list of style declarations. For example:

```
h1 em, p strong, address {color: red}
```

means that you'll see red whenever the `<em>` tag appears within an `<h1>` tag, or when the `<strong>` tag appears within a `<p>` tag, and for the contents of the `<address>` tag.

The nesting need not be exact to match the rule. For example, if you nest the `<strong>` tag within a `<ul>` tag within a `<p>` tag, you'll still match the rule for `p strong` that we defined above. If a particular nesting matches several style rules, the most specific rule is used. For example, if you defined two contextual selectors:

```
p strong {color: red}
p ul strong {color: blue}
```

and use the sequence `<p><ul><strong>` in your document, the second, more specific rule applies, coloring the contents of the `<strong>` tag blue.

8.2.4 Universal, Child, and Adjacent Selectors

The CSS2 standard defines additional patterns for selectors, besides commas and spaces, as illustrated in the following examples:

```
* {color: purple; font: ZapfDingBats}
ol > li {font-size: 200%; font-style: italic}
h1 + h2 {margin-top: +4mm}
```

In the first example, the universal asterisk selector applies the style to all elements of your document, so that any text gets displayed in ZapfDingBat characters.[5] The second example selects a particular child/parent relationship, in this case items in an ordered list.

[5] Assuming, of course, that the style is not overridden by a subsequent rule.

The third example illustrates the adjacent selector type which selects for one tag immediately following another in your document. In this case, the special selector adds vertical space to instances in which your document has a level-2 header immediately following a level-1 header.

8.2.5 Pseudo-Elements

There are elemental relationships in your documents you cannot explicitly tag. The drop-cap is a common print style, but how do you select the first letter in a paragraph? There are ways, but you have to identify each and every instance separately. There is no tag for the first line in a paragraph. And there are occasions where you might want the browser to automatically generate content, such as to add the prefix "Item #" and automatically number each item in an ordered list.

CSS2 introduces four new pseudo-elements that let you define special relationships and styles for their display: first-line, first-letter, before and after. Declare each as a colon-separated suffix of a standard markup element. For example:

```
p:first-line {font-size: 200%; font-style: italic}
```

means that the browser should display the first line of each paragraph italicized and twice as large as the rest of the text. Similarly:

```
p:first-letter {font-size: 200%; float: left}
```

tells the browser to make the first letter of a paragraph twice as large as the remaining text and float the letter to the left, allowing the first two lines of the paragraph to float around the larger initial letter.[6]

[6] The properties that can be specified for the `first-letter` and `first-line` pseudo-classes are the font properties, color and background properties, `text-decoration`, `vertical-align`, `text-transform`, `line-height`, and `clear`. In addition, the `first-letter` pseudo-class accepts the margin properties, padding properties, border properties, and `float`. The `first-line` pseudo-class also accepts the `word-spacing` and `letter-spacing` properties.

The `:before` and `:after` pseudo-elements let you identify where in your document you insert generated content, such as list numbers, special lead-in headers, and so forth. Hence, these pseudo-elements go hand-in-hand with the CSS2 content and counter properties. To whet your appetite, consider this example:

```
ol {counter-reset: item}
ol li:before {content: "Item #" counter(item) " ";
                counter-increment: item}
```

Too bad none of the current browsers support pseudo-elements.

8.3 Style Classes

CSS2 classes let you create, at the document level or in an external style sheet, several different styles for the same elements, each distinguished by a class name. To apply the style class, name it as the value of the `class` attribute in its corresponding tag.

8.3.1 Regular Classes

In a technical paper you might want to define one paragraph style for the abstract, another for equations, and a third for centered quotations. None of the paragraph tags may have an explicit context in the document so you could distinguish it from the others. Rather, you may define each as a different style class:

```
<style>
<!--
p.abstract {font-style: italic; margin-left: 0.5cm; margin-right: 0.5cm}
p.equation {font-family: Symbol; text-align: center}
h1, p.centered {text-align: center; margin-left: 0.5cm; margin-right: 0.5cm}
-->
</style>
```

Notice first in the example that defining a class is simply a matter of appending a period-separated class name as a suffix to the tag name as the selector in a style rule. Unlike the XHTML-compliant selector, which is the name of the standard tag and must be in lowercase, the class name can be any sequence of letters, numbers, and hyphens, but must begin with a letter.[7] Careful, though, case does matter, so that `abstract` is not the same as `AbsTRact`. And classes, like selectors, may be included with other selectors, separated by commas, as in the third example. The only restriction on classes is that they cannot be nested: `p.equation.centered` is not allowed, for example.

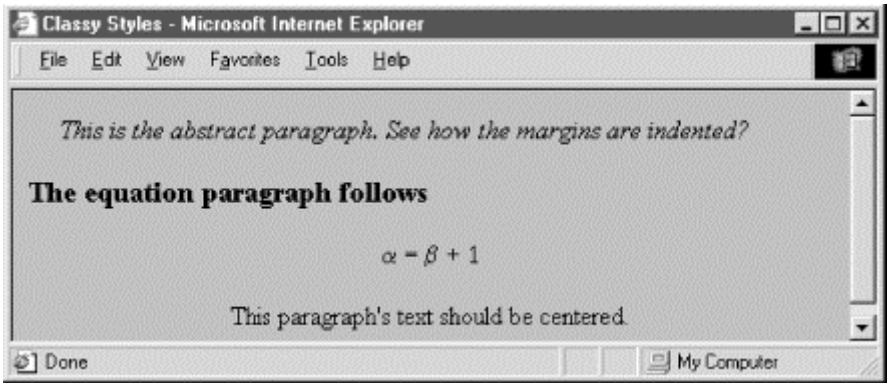
[7] Due to its support of JavaScript style sheets, Netscape cannot handle class names that happen to match JavaScript keywords. The class "abstract," for instance, generates an error in Netscape.

Accordingly, the first rule in the example creates a class of paragraph styles named "abstract" whose text will be italic and indented from the left and right margins by a half-centimeter. Similarly, the second paragraph style-class "equation" instructs the browser to center the text and to use the Symbol typeface to display the text. The last style rule creates a style with centered text and half-centimeter margins, applying this style to all level one headers as well as creating a class of the `<p>` tag named `centered` with that style.

To use a particular class of a tag, you add the `class` attribute to the tag, as in this example, as rendered by Internet Explorer in Figure 8-2:

```
<p class=abstract>
This is the abstract paragraph.  See how the margins are indented?
</p>
<h3>The equation paragraph follows</h3>
<p class=equation>
a = b + 1
</p>
<p class=centered>
This paragraph's text should be centered.
</p>
```

Figure 8-2. Use classes to distinguish different styles for the same tag



For each paragraph, the value of the `class` attribute is the name of the class to be used for that tag.

8.3.2 Generic Classes

You may also define a class without associating it with a particular tag, and apply that class selectively through your documents for a variety of tags. For example:

```
.italic {font-style: italic}
```

creates a generic class named `italic`. To use it, simply include its name with the `class` attribute. So, for instance, use `<p class=italic>` or `<h1 class=italic>` to create an italic paragraph or header.

Generic classes are quite handy and make it easy to apply a particular style to a broad range of tags. Netscape Navigator and Internet Explorer support CSS2 generic classes.

8.3.3 ID Classes

Almost all HTML tags accept the `id` attribute, which assigns an identifier to the element that is unique within the document. This identifier can be the target of a URL, used by automated document processing tools and can also be used to specify a style rule for the element.

To create a style class that can be applied with the `id` attribute, follow the same syntax used for style classes, except with a `#` character before the class name instead of a period. This style creates such classes:

```
<style>
<!--
#yellow {color : yellow}
h1#blue {color : blue}
-->
</style>
```

Within your document, you could use `<h1 id=blue>` to create a blue heading, or add `id=yellow` to almost any tag to turn it yellow. You can mix and match both `class` and `id` attributes, giving you a limited ability to apply two independent style rules to a single element.

There is a dramatic drawback to using classes defined this way: the value of the `id` attribute must be unique within the document. You cannot legally reuse the class, although the browser might let you get away with it.

For this reason, we strongly discourage creating and using these kinds of classes. Stick to the conventional style of classes to create correct, robust documents.

8.3.4 Pseudo-Classes

In addition to conventional style classes, the CSS2 standard defines pseudo-classes, which allow you to define the display style for certain tag *states*. Pseudo-classes are like regular classes, with two notable differences: they are attached to the tag name with a colon instead of a period, and they have predefined names, not arbitrary ones you may give them.

There are seven pseudo-classes, three of which are explicitly associated with the `<a>` tag.

8.3.4.1 Hyperlink pseudo-classes

The popular CSS2-compliant browsers distinguish three special states for the hyperlinks created by the `<a>` tag: not visited, being visited, and visited. The browser may change the appearance of the tag's contents to indicate its state, such as with underlining or color. Through pseudo-classes, you can control how these states get displayed by defining styles for `a:link` (not visited), `a:active` (being visited), and `a:visited`.

The `:link` pseudo-class controls the appearance of links that are not selected by the user and have not yet been visited. The `:active` pseudo-class defines the appearance of links that are currently selected by the user and are being processed by the browser. The `:visited` pseudo-class defines those links that have already been visited by the user.

To completely define all three states of the `<a>` tag, you might write:

```
a:link {color: blue}
a:active {color: red; font-weight: bold}
a:visited {color: green}
```

Unvisited links will be shown in blue. When the user selects a link, the browser will change its text color to red and make it bold. Once visited, the link will revert to conventional green text.

8.3.4.2 Interaction pseudo-classes

The CSS2 standand defines two new pseudo-classes, which along with `:active`, relate to user actions and advise the interactive agent, such as a browser, how to display the affected element as the user interacts with the element. In other words, these two pseudo-classes are dynamic: `hover` and `focus`.

For instance, when you drag the mouse over a hyperlink in your document, the browser may change the mouse pointer icon. Hovering can be associated with a style that is only in effect while the mouse is over the element. For example, if you add the `:hover` pseudo-class to our example list of hyperlink style rules:

```
a:hover {color: yellow}
```

the text associated with unvisited links will be rendered in blue, turn yellow when you point to it with the mouse, go red while you visit it, and turn green after you're done visiting.

Similarly, the `:focus` pseudo-class lets you change the style for an element when it becomes the object of attention. An element may be under focus when you tab to it, click on it, or, dependng on the browser, advance the cursor to it. Regardless of how the focus got to the element, the style rules associated with the focus pseudo-class are only applied while the element has the focus.

8.3.4.3 Nesting and language pseudo-classes

The new CSS2 `:first-child` pseudo-class lets you specify how an element may be rendered when it is the first child of the containing element. For instance, the following rule selects only those paragraphs which are the first child of a division; there can be no intervening elements. Then, and only then, will the paragraph's text contents be rendered in italics:

```
div > p:first-child  {font-style: italic}
```

Accordingly, the first paragraph in the following HTML fragment would be rendered in italics by a CSS2-compliant browser since it is the first child element of its division. Conversely, the second paragraph comes after a level-2 header, which is the first child of the second division. So, that second paragraph in the example gets rendered in plain text because it is not the first child of its division:

```
<div>
  <p>
    I get to be in italics.
  </p>
</div>
<div>
  <h2> New Division</h2>
  <p>
    I'm in plain text because my paragraph is a second child of the division.
```

Finally, the CSS2 standard defines a new pseudo-class that lets you select an element based on its language. For instance, you might include the `lang=fr` attribute in a `<div>` tag to instruct the browser that the division contains French language text. The browser may specially treat the text. Or, you may impose a specific style with the pseudo-class `:lang`. For example:

```
div:lang(it) {font-family: Roman}
```

says that text in divisions of a document that contain the Italian language should use the Roman font family. Appropriate, don't you think? Notice that you specify the language in parentheses immediately after the `lang` keyword. Use the same two-letter ISO standard code for the pseudo-class `:lang` as you do for the lang attribute. Section 3.6.1.2

8.3.4.4 Browser support of pseudo-classes

None of the popular browsers support the `:lang`, `:first-child`, or `:focus` pseudo-classes yet. All of the popular browsers support the `:link`, `:active`, and `:visited` pseudo-classes for the hyperlink tag (`<a>`). Even though `:active` also may be used for other elements, none of the browsers yet support applications beyond the `<a>` tag. The `:hover` pseudo-class is great for special effects on links and other elements, but only Internet Explorer supports it and only for hyperlinks.

8.3.5 Mixing Classes

You may mix pseudo-classes with regular classes by appending the pseudo-class name to the selector's class name. For example, here are some rules that define plain, normal, and fancy anchors:

```
a.plain:link, a.plain:active, a.plain:visited {color: blue}
a:link {color: blue}
a:visited {color: green}
a:active {color: red}
a.fancy:link {font-style: italic}
a.fancy:visited {font-style: normal}
a.fancy:active {font-weight: bold; font-size: 150%}
```

The `plain` version of `<a>` is always blue, no matter the state of the link. Normal links start out blue, turn red when active, and convert to green when visited. The `fancy` link inherits the color scheme of the normal `<a>` tag, but adds italic text for unvisited links, converts back to normal text after being visited, and actually grows 50 percent in size and becomes bold when active.

A word of warning about that last property of the `fancy` class: specifying a font size change for a transient display property will result in lots of browser redisplay activity when the user clicks on the link. Given that some browsers run on slow machines, this redisplay may be annoying to your readers. Given also that implementing that sort of display change is something of a pain, it is unlikely that most browsers will support radical appearance changes in `<a>` tag pseudo-classes.

8.3.6 Class Inheritance

Classes inherit the style properties of their generic base tag. For instance, all the properties of the plain `<p>` tag apply to a specially defined paragraph class, except where the class overrides a particular property.

Classes cannot inherit from other classes, only from the unclassed version of the tag they represent. In general, therefore, you should put as many common styles into the rule for the basic version of a tag and create classes only for those properties that are unique to that class. This makes maintenance and sharing of your style classes easier, especially for large document collections.

8.4 Style Properties

At the heart of the CSS2 specification are the many properties that let you control how the styles-conscious browser presents your documents to the user. The standard collects these properties into six groups: fonts, colors and backgrounds, text, boxes and layout, lists, and tag classification. We'll stick with that taxonomy and preface the whole shebang with a discussion of property values and inheritance before diving into the properties themselves.

You'll find a summary of the style properties in Appendix C.

8.4.1 Property Values

There are five distinct kinds of property values: keywords, length values, percentage values, URLs, and colors.

8.4.1.1 Keyword property values

A property may have a `keyword` value that expresses action or dimension. For instance, the effects of `underline` and `line-through` are obvious property values. And you can express property dimensions with keywords like `small` and `xx-large`. Some keywords are even relational: `bolder`, for instance, is an acceptable value for the `font-weight` property. Keyword values are not case-sensitive: `Underline`, `UNDERLINE`, and `underline` are all acceptable keyword values.

8.4.1.2 Length property values

So-called `length` values (a term taken from the CSS2 standard) explicitly state the size of a property. They are numbers, some with decimals, too. Length values may have a leading + or - sign to indicate that the value is to be added to or subtracted from the immediate value of the property. Length values must be followed immediately by a two-letter unit abbreviation - with no intervening spaces.

There are three kinds of length-value units: relative, pixels, and absolute. Relative units specify a size that is relative to the size of some other property of the content. Currently, there are only two relative units: em, which is the width of m in the current font (written as `em`); and x-height, which is the height of the letter "x" in the current font (abbreviated `ex`). The pixels unit, abbreviated `px`, is equal to the size of a pixel on the browser's display. Absolute property value units are more familiar to us all. They include inches (abbreviated `in`), centimeters (`cm`), millimeters (`mm`), points (`pt`, 1/72 of an inch), and picas (`pc`, twelve points).

All of the following are valid length values, although not all units are recognized by the styles-conscious browsers:

```
1in
1.5cm
+0.25mm
-3pt
-2.5pc
+100em
-2.75ex
250px
```

8.4.1.3 Percentage property values

Similar to the relative length-value type, a percentage value describes a proportion relative to some other aspect of the content. It has an optional sign and decimal portion to its numeric value, and must have the percent sign (`%`) suffix. For example:

```
line-height: 120%
```

computes the separation between lines to be 120 percent of the current line height (usually relative to the text font height). Note that this value is not dynamic, though: changes made to the font height after the rule has been processed by the browser will not affect the computed line height.

8.4.1.4 URL property values

Some properties also accept, if not expect, a URL as a value. The syntax for using a URL in a style property is different from conventional HTML/XHTML:

```
url(service://server.com/pathname)
```

The keyword `url` is required, as are the opening and closing parentheses. Do not leave any spaces between `url` and the opening parenthesis. The `url` value may contain either an absolute or a relative URL. However, note that the URL is relative to the immediate style sheet's URL, the one in which it is declared. This means that if you use a `url` value in a document-level or inline style, the URL is relative to the HTML document containing the style document. Otherwise, the URL is relative to the `@import`ed or `<link>`ed external style sheet's URL.

8.4.1.5 Color property values

Color values specify colors in a property (surprised?). You can specify a color as a color name or a hexadecimal RGB triple, as is done for common attributes, or as a decimal RGB triple unique to style properties. Both color names and hexadecimal RGB triple notation are described in Appendix G.

Unlike regular HTML or XHTML, style sheets will accept three-digit hexadecimal color values. The single digit is doubled to create a conventional six-digit triple. Thus, the color `#78C` is equivalent to `#7788CC`. In general, three-digit color values are handy only for simple colors.

The decimal RGB triple notation is a bit different:

`rgb(`*`red`*`, `*`green`*`, `*`blue`*`)`

The `red`, `green`, and `blue` intensity value are integers in the range zero to 255 or integer percentages. As with a URL value, do not leave any spaces between `rgb` and the opening parenthesis.

For example, in decimal RGB convention, the color white is `rgb(255, 255, 255)` or `rgb(100%, 100%, 100%)`, and a medium yellow is `rgb(127, 127, 0)` or `rgb(50%, 50%, 0%)`.

8.4.2 Property Inheritance

In lieu of a specific rule for a particular element, properties and their values for tags within tags are inherited from the parent tag. Thus, setting a property for the `<body>` tag effectively applies that property to every tag in the body of your document, except for those that specifically override it. So, to make all the text in your document blue, you need only say:

```
body {color: blue}
```

rather than create a rule for every tag you use in your document.

This inheritance extends to any level. If you later created a `<div>` tag with text of a different color, the styles-conscious browser will display all the text contents of the `<div>` tag and all its enclosed tags in that new color. When the `<div>` tag ends, the color reverts to that of the containing `<body>` tag.

In many of the following property descriptions, we refer to the tag containing the current tag as the "parent element" of that tag.

8.4.3 Font Properties

The loudest complaint we hear is that HTML and its progeny XHTML lack font styles and characteristics that even the simplest of text editors implement. The various `<font>` attributes address part of the problem, but they are tedious to use since each text font change requires a different `<font>` tag.

Style sheets change all that, of course. The CSS2 standard provides seven font properties that modify the appearance of text contained within the affected tag: `font-family`, `font-size`, `font-size-adjust`, `font-style`, `font-variant`, `font-stretch`, and `font-weight`. In addition, there is a universal `font` property that lets you declare all of the font changes with a single property.

Please be aware that style sheets cannot overcome limitations of the client system nor can the browser conjure effects if the fonts it uses do not provide the means.

8.4.3.1 The font-family property

The `font-family` property accepts a comma-separated list of font names, one of which will be selected by the styles-conscious browser for display of the tag's text. The browser uses the first font named in the list that also is installed and available for display on the client machine.

Font name values are for specific font styles, such as Helvetica or Courier, or a generic font style as defined by the CSS2 standard: `serif`, `sans-serif`, `cursive`, `fantasy`, and `monospace`. Each browser defines which actual font name is to be used for each generic font. For instance, Courier is the most popular choice for a monospace font.

Since fonts vary wildly among browsers, when specifying a font style, you should always provide several choices, ending with a suitable generic font.

For example:

```
h1 {font-family: Helvetica, Univers, sans-serif}
```

causes the browser to look for and use Helvetica, and then Univers. If neither font is available for the client display, the browser will use the generic sans serif typeface.

Enclose font names that contain spaces - New Century Schoolbook, for example - in quotation marks. For example:

```
p {font-family: Times, "New Century Schoolbook", Palatino, serif}
```

That extra set of double quotation marks in an inline style rule will cause problems. Accordingly, use single quotation marks in an inline style:

```
<p style="font-family: Times, 'New Century Schoolbook', Palatino, serif">
```

In practice, you need not use quotation marks: the browser will ignore spaces before and after the font name, and convert multiple internal spaces to a single space. Thus:

```
p {font-family: Times, New Century Schoolbook, Palatino, serif}
<p style="font-family: Times, New Century Schoolbook, Palatino, serif">
```

are both legal, but we recommend that you use quotation marks anyway, just in case things change.

8.4.3.2 The font-size property

The `font-size` property lets you prescribe absolute or relative length values, percentages, and keywords to define the font size. For example:

```
p {font-size: 12pt}
p {font-size: 120%}
p {font-size: +2pt}
p {font-size: medium}
p {font-size: larger}
```

The first rule is probably the most used because it is the most familiar: it sets the font size to a specific number of points (12 in this example). The second example rule sets the font size to be 20 percent larger than the parent element's font size. The third increases the font's normal size by two points.

The fourth example selects a predefined size set by the browser, identified by the `medium` keyword. Valid absolute-size keywords are `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, and `xx-large`, and usually correspond to the seven font sizes used with the `size` attribute of the `<font>` tag.

The last `font-size` rule selects the next size larger than the font associated with the parent element. Thus, if the size were normally `medium`, it would be changed to `large`. You can also specify `smaller`, with the expected results.

None of the current browsers handle the incremented font-size correctly. Rather, they ignore the increment/decrement sign and use its value as an absolute size. So, for instance in the middle example in this section, the font-size would end up as two points, not two points larger than the normal size.

8.4.3.3 The font-stretch property

Besides different sizes, font families sometimes contain condensed and expanded versions in which the characters are squeezed or stretched, respectively. Use the font-stretch property to choose more compressed or stretched-out characters from your font.

Use the property value of `normal` to, of course, select the normal-sized version of the font. The relative values `wider` and `narrower` select the next wider or next narrower variant of the font's characters, respectively, but not wider or narrower than the most ("ultra") expanded or contracted one in the family.

The remaining font-stretch property values choose specific variants from the font family. Starting from the most condensed and ending with the most expanded, the values are `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `extra-expanded`, and `ultra-expanded`.

The font-stretch property, of course, assumes that your display fonts support stretchable fonts. Even so, the current popular browsers ignore this property.

8.4.3.4 The font-size-adjust property

Without too many details, the legibility and display size of a font depends principally on its aspect ratio: the ratio of its rendered size to its x-height, which is a measure of the font's lowercase glyph height. Fonts with aspect ratios approaching 1.0 tend to be more legible at smaller sizes than fonts with aspect ratios approaching zero.

Also, because of aspect ratios, the actual display size of one font may be apparently smaller or larger than another font at the same size. So, when one font is not available for rendering, the substituted font may distort the presentation.

The font-size-adjust property lets you readjust the substituted font's aspect ratio so that it will better fit the display. Use the property value of `none` to ignore the aspect ratio. Otherwise, include your desired aspect ratio (a decimal value less than one), typically the aspect ratio for your first-choice display font. The styles-conscious browser thereby will compute and display the substituted font at a size adjusted to your specified aspect ratio:

```
s = (n/a) * fs
```

where `s` is the new, computer font-size for display of the substituted font, calculated as the `font-size-adjust` value `n` divided by the substituted font's aspect ratio `a` times the current font-size `fs`. For example, let's imagine that your first-choice font Times New Roman, which has an aspect ratio of 0.45, is not available, so the browser then substitutes Comic Sans MS, which has an aspect ratio of 0.54. So that the substitution maintains nearly equivalent sizing for the font display, say at a 18 px font-size, with the `font-size-adjust` property set to 0.45, the CSS2-compliant browser would display or print the text with the substituted Comic Sans MS font at the smaller (0.45/0.54 x 18 px) = 15 px.

Sorry that we can't show you how the popular browsers would do this because they don't support it.

8.4.3.5 The font-style property

Use the `font-style` property to slant text. The default style is `normal` and may be changed to `italic` or `oblique`. For example:

```
h2 {font-style: italic}
```

makes all level-2 header text italic. As of this writing, Netscape 4 supports only the `italic` value for `font-style`; Internet Explorer 4 and 5 support both values, although it is usually hard to distinguish italic from oblique.

8.4.3.6 The font-variant property

The `font-variant` property lets you select a variant of the desired font. The default value for this property is `normal`, indicating the conventional version of the font. You may also specify `small-caps` to select a version of the font in which the lowercase letters have been replaced with small capital letters.

This property is not supported by Netscape; Internet Explorer versions 4 and 5 incorrectly implements `small-caps` as all uppercase letters.

8.4.3.7 The font-weight property

The `font-weight` property controls the weight or boldness of the lettering. The default value of this property is `normal`. You may specify `bold` to obtain a bold version of a font, or use the relative `bolder` and `lighter` values to obtain a version of the font that is bolder or lighter than the parent element's font.

To specify varying levels of lightness or boldness, set the value to a multiple of 100, between the values `100` (lightest) and `900` (boldest). The value `400` is equal to the `normal` version of the font, and `700` is the same as specifying `bold`.

Internet Explorer and Netscape Navigator support the `normal` and `bold` values. Only Internet Explorer supports the `lighter` and `bolder` values, as well. Both browsers support the numeric boldness values.

8.4.3.8 The font property

More often than not, you'll find yourself specifying more than one font-related property at a time for a tag's text content display. A complete font specification can get somewhat unwieldy; for example:

`font` property

```
p {font-family: Times, Garamond, serif;
   font-weight: bold;
   font-size: 12pt;
   line-height: 14pt}
```

To mitigate this troublesome and potentially unreadable collection, use the comprehensive `font` property and group all the attributes into one set of declarations:

```
p {font: bold 12pt/14pt Times, Garamond, serif}
```

The grouping and ordering of font attributes is important within the `font` property. The font style, weight, and variant attributes must be specified first, followed by the font size and the line height separated by a slash character, and ending with the list of font families. Of all the properties, the size and family are required; the others may be omitted.

Here are some more sample `font` properties:

```
em {font: italic 14pt Times}
h1 {font: 24pt/48pt sans-serif}
code {font: 12pt Courier, monospace}
```

The first example tells the styles-conscious browser to emphasize `<em>` text using a 14-point italic Times face. The second rule has `<h1>` text displayed in the boldest 24-point sans-serif font available, with an extra 24 points of space between the lines of text. Finally, text within a `<code>` tag is set in 12-point Courier or the browser-defined monospace font.

We leave it to your imagination to conjure up examples of the abuses you could foster with the font styles. Perhaps a recent issue of *Wired* magazine, notorious for avant-garde fonts and other print-related abuses, would be helpful in that regard?

8.4.4 Color and Background Properties

Every element in your document has a foreground and a background color. In some cases, the background is not one color, but a colorful image. The color and background style properties control these colors and images.

The children of an HTML/XHTML element normally inherit the foreground color of their parent. For instance, if you make `<body>` text red, the styles-conscious browser also will display header and paragraph text in red.

Background properties behave differently, however - they are not inherited. Instead, each element has a default background that is transparent, allowing the parent's background to show through. Thus, setting the background image of the `<body>` tag does not cause that image to be reloaded for every element within the body tag. Instead, the browser loads the image once and displays it behind the rest of the document, serving as the background for all elements which do not themselves have an explicit background color or image.

8.4.4.1 The background-attachment property

If you specify a background image for an element, use the `background-attachment` property to control how that image is attached to the browser's display window. With the default value `scroll` , the browser moves the background image with the element as the user scrolls through the document. A value of `fixed` prevents the image from moving.

Netscape does not support this style property.

8.4.4.2 The background-color property

The `background-color` property controls the (you guessed it!) background color of an element. Set it to a color value or to the keyword `transparent`. The default value is `transparent`. The effects should be obvious.

While you may have become accustomed to setting the background color of an entire document through the special attributes for the `<body>` tag, the `background-color` style property can be applied to any element. For example, to set the background color of one item in a bulleted list, you could use:

```
<li style="background-color: blue">
```

Similarly, all the table header cells in a document could be given a reverse video effect with:

```
th {background-color: black; color: white}
```

If you really want your emphasized text to stand out, paint its background red:

```
em {background-color: red}
```

Netscape does not support this CSS2 property explicitly, but you may achieve the same effects through its support of the general `background` property, as discussed in Section 8.4.4.6.

8.4.4.3 The background-image property

The `background-image` property puts an image behind the contents of an element. Its value is either a URL or the keyword `none`. The default value is `none`.

As with background colors, you can place a background image behind the entire document or behind selected elements of a document. With this style property, effects like placing an image behind a table or selected text are now simple:

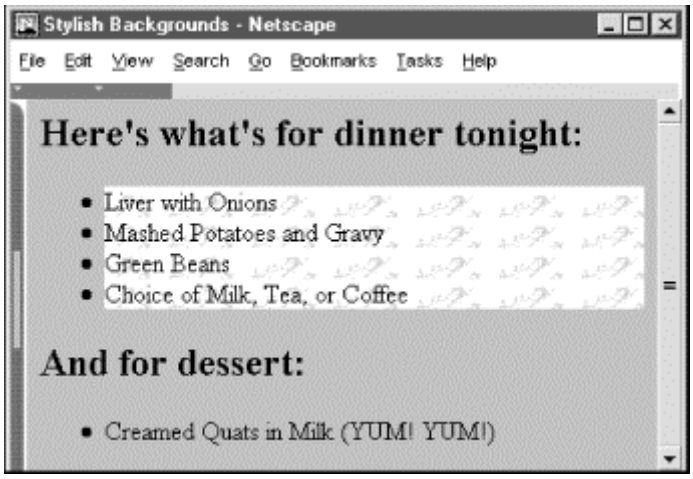
```
<table style="background-image: url(backgrounds/woodgrain.gif)">
li.marble {background-image: url(backgrounds/marble.gif)}
```

The first example uses an inline style to place a woodgrain finish behind a table. The second defines a list item class that places a marble background behind `<li>` tags that use the `class=marble` attribute. For example, in XHTML:

```
<h2>Here's what's for dinner tonight:</h2>
<ul>
   <li class="marble">Liver with Onions</li>
   <li class="marble">Mashed Potatoes and Gravy</li>
   <li class="marble">Green Beans</li>
   <li class="marble">Choice of Milk, Tea, or Coffee</li>
</ul>
<h2>And for dessert:</h2>
<ul>
   <li>Creamed Quats in Milk  (YUM! YUM!)</li>
</ul>
```

will produce a result like that in Figure 8-3.

Figure 8-3. Placing a background image behind an element



If the image is larger than the containing element, it will be clipped to the area occupied by the element. If the image is smaller, the image will be repeated to tile the area occupied by the element, as dictated by the value of the `background-repeat` attribute.

You control the position of the image within the element with the `background-position` property. The scrolling behavior of the image is managed by the `background-attachment` property.

While it may seem that a background color and a background image are mutually exclusive, you should usually define a background color even if you are using a background image. That way, if the image is unavailable, such as when the user doesn't automatically download images, the browser will display the background color instead. In addition, if the background image has transparent areas, the background color will be used to fill in those areas.

8.4.4.4 The background-position property

By default, the styles-conscious browser renders a background image starting in the upper-left corner of the allotted display area and tiled (if necessary) down and over to the lower-right corner of that same area. With the `background-position` property, you can offset the starting position of the background image down and to the right of that default point by an absolute (length) or relative (percentage or keyword) offset. The resulting image fills the area from that offset starting point to the lower-right corner of the display space.

You may specify one or two values for the `background-position` property. If you use a single value, it applies to both the vertical and horizontal positions. With two values, the first is the horizontal offset, and the second is the vertical offset.

Length values (with their appropriate units; see Section 8.4.1.2) indicate an absolute distance from the upper-left corner of the element behind which you display the background image. For instance:

```
table {background-image: url(backgrounds/marble.gif);
       background-position: 10px 20px}
```

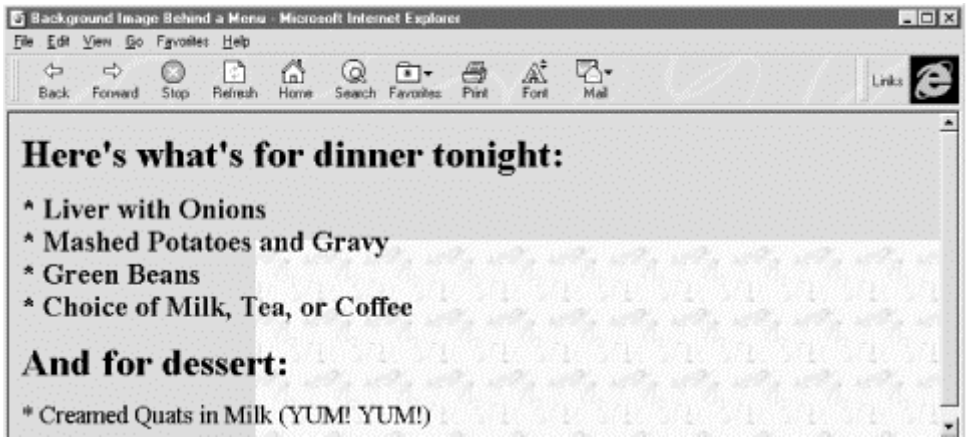
offsets the marble background 10 pixels to the right and 20 pixels down from the upper-left corner of any `<table>` element in your document.

Percentage values are a bit trickier, but somewhat easier to use. Measured from to 100 percent from left to right and top to bottom, the center of the element's content display space is at 50%, 50%. Similarly, the position one-third of the way across the area and two-thirds of the way down is at 33%, 66%. So, to offset the background for our example dinner menu to the center of the element's content display space, we use:

```
background-position: 50% 50%
```

Notice that the browser places the first *marble.gif* tile at the center of the content display area and tiles to the right and down the window, as shown in Figure 8-4.

Figure 8-4. Marbled background offset to the center of the display



So why use a number when a single word will do? You can use the keywords `left`, `center`, and `right`, as well as `top`, `center`, and `bottom`, for `0%`, `50%`, and `100%`, respectively. To center an image in the tag's content area write:

```
background-position: center center
```

You can mix and match length and percentage values,[8]

[8] That is, if the browser supports the value units. So far, Internet Explorer and Netscape support only a meager repertoire of length units - pixels and percents.

so that:

```
background-position: 1cm center
```

places the image one centimeter to the right of the tag's left edge, centered vertically in the tag's area.

8.4.4.5 The background-repeat property

Normally, the browser tiles a background image to fill the allotted space, repeating the image both down and to the right. Use the `background-repeat` property to alter this "repeat" (default value) behavior. To have the image repeat horizontally but not vertically, use the value `repeat-x`. For only vertical repetition, use `repeat-y`. To suppress tiling altogether, use `no-repeat`.

A common use of this property is to place a watermark or logo in the background of a page without repeating the image over and over. For instance:

```
body {background-image: url(backgrounds/watermark.gif);
      background-position: center center;
      background-repeat: no-repeat
     }
```

will place the watermark image in the background at the center of the page.

A popular trick is to create a vertical ribbon down the right-hand side of the page:

```
body {background-image: url(backgrounds/ribbon.gif);
      background-position: top right;
      background-repeat: repeat-y
     }
```

8.4.4.6 The background property

Like the various font properties, the many background CSS2 properties can get cumbersome to write and hard to read later. So, like the `font` property, there also is a general `background` property.

The background property accepts values from any and all of the `background-color`, `background-image`, `background-attachment`, `background-repeat`, and `background-position` properties, in any order. If you do not specify values for some of the properties, that property is explicitly set to its default value. Thus:

```
background: red
```

sets the `background-color` property to red and resets the other background properties to their default values. A more complex example:

```
background: url(backgrounds/marble.gif) blue repeat-y fixed center
```

sets all the background image and color properties at once, resulting in a marble image on top of a blue background (blue showing through any transparent areas). The image repeats vertically, starting from the center of the content display area, and does not scroll when the user scrolls the display. Notice that we included just a single position value (`center`) and the browser used it for both the vertical and horizontal positions.

Although Netscape Navigator version 6 provides full support, the browser's version 4 supports only the `background` property and does not honor any of the individual background ones. For this reason, you may want to use the `background` property to achieve the broadest acceptance of your background image and color properties.

8.4.4.7 The color property

The `color` property sets the foreground color for a tag's contents - the color of the text lettering, for instance. Its value is either the name of a color, a hexadecimal RGB triple, or a decimal RGB triple, as outlined in Section 8.4.1.5. Thus, the following are all valid property declarations:

```
color: mauve
color: #ff7bd5
color: rgb(255, 125, 213)
color: rgb(100%, 49%, 84%)
```

Generally, you'll use the `color` property with text, but you may also modify non-textual content of a tag. For example, the following example produces a green horizontal rule:

```
hr {color: green}
```

If you don't specify a color for an element, it inherits the color of its parent element.

8.4.5 Text Properties

Cascading style sheets make a distinction between font properties, which control the size, style, and appearance of text, and text properties, which control how text is aligned and presented to the user.

8.4.5.1 The letter-spacing property

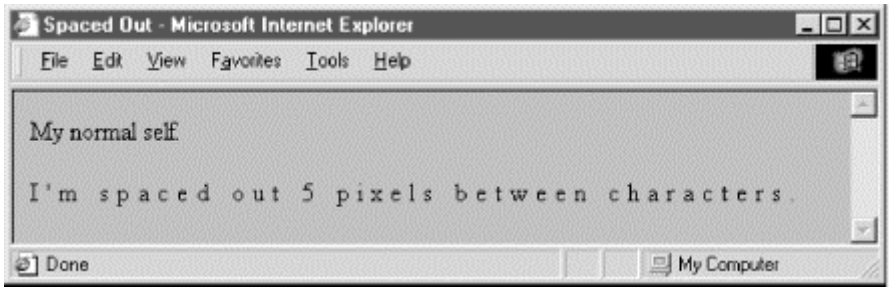
The `letter-spacing` property puts additional space between text letters as they are displayed by the browser. Set the property with either a length value or the default keyword `normal`, indicating that the browser should use normal letter spacing. For example:

```
blockquote {letter-spacing: 2px}
```

puts an additional two pixels between adjacent letters within the `<blockquote>` tag. Figure 8-5 illustrates what happens when you put 5 pixels between characters.

This property currently is supported by Internet Explorer and by the latest Netscape Version 6.

Figure 8-5. The letter-spacing property lets you stretch text out



8.4.5.2 The line-height property

Use the `line-height` property to define the spacing between lines of a tag's text content. Normally, browsers single-space text lines - the top of the next line is just a few points below the last line. By adding to that line height, you increase the amount of space between lines.

The `line-height` value can be an absolute or relative length, a percentage, a scaling factor, or the keyword `normal`. For example:

```
p {line-height: 14pt}
p {line-height: 120%}
p {line-height: 2.0}
```

The first example sets the line height to exactly 14 points between baselines of adjacent lines of text. The second computes the line height to 120 percent of the font size. The last example uses a scaling factor to set the line height to twice as large as the font size, creating double-spaced text. The value `normal`, the default, is usually equal to a scaling factor of 1.0 to 1.2.

Keep in mind that absolute and percentage values for `line-height` compute the line height based upon the value of the `font-size` property when the `line-height` property is defined. The computed property value will be inherited by children of the element. Subsequent changes to `font-size` by either the parent or child elements will not change the computed `line-height`.

Scaling factors, on the other hand, defer the line-height computation until the text is actually displayed. Hence, varying `font-size`s affect `line-height` locally. In general, it is best to use a scaling factor for the `line-height` property so that the line height will change automatically when the font size is changed.

Although usually considered separate from font properties, you may include this text-related `line-height` property's value as part of the shorthand notation of the `font` property. Section 8.4.3.8

8.4.5.3 The text-align property

Text justified with respect to the page margins is a rudimentary feature of nearly all text processors. The `text-align` property brings that capability to HTML for any block-level tag. (The W3C standards people prefer that you use CSS2 `text-align` styles rather than the explicit `align` attribute for those block-level tags like `<div>` and `<p>`.) Use one of four values: `left`, `right`, `center`, or `justify`. The default value is, of course, `left`. For example:

```
div {text-align: right}
```

tells the styles-conscious browser to align all the text inside `<div>` tags against the right margin. The `justify` value tells the browser to align the text to both the left and right margins, spreading the letters and words in the middle to fit.

8.4.5.4 The text-decoration property

The `text-decoration` property produces text embellishments, some of which also are available with the original physical style tags. Its value is one or more of the keywords `underline`, `overline`, `line-through`, and `blink`. The value `none` is the default and tells the styles-conscious browser to present text normally.

The `text-decoration` property is handy for defining different link appearances. For example:

```
a:visited, a:link, a:active {text-decoration: underline overline}
```

puts lines above and below the links in your document.

This text property is not inherited, and non-textual elements are not affected by the `text-decoration` property.

Like the `<blink>` tag, Netscape Navigator supports the `blink` text-decoration property value, but not Internet Explorer.

8.4.5.5 The text-indent property

Although less common today, it still is standard practice to indent the first line of a paragraph of text.[9]

[9] But not, obviously, in this book.

And some text blocks, such as definitions, typically "out-dent" the first line, creating what is called a *hanging indent*.

The CSS2 `text-indent` property lets you apply these features to any block tag and thereby control the amount of indentation of the first line of the block. Use length and percentage values; negative values create the hanging indent. Percentage values compute the indentation as a percentage of the parent element's width. The default value is zero.

To indent all the paragraphs in your document, for example:

```
p {text-indent: 3em}
```

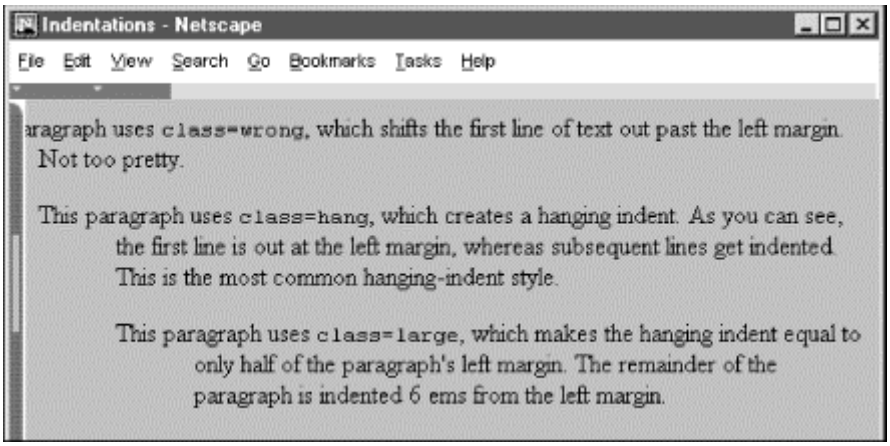
The length unit `em` scales the indent as the font of the paragraph changes in size on different browsers.

Hanging indents are a bit trickier because you have to watch out for the element borders. Negative indentation does not shift the left margin of the text; it simply shifts the first line of the element left, possibly into the margin, border, or padding of the parent element. For this reason, hanging indents only work as expected if you also shift the left margin of the element to the right by an amount equal to or greater than the size of the hanging indent. For example:

```
p.wrong {text-indent: -3em}
p.hang  {text-indent: -3em; margin-left: 3em}
p.large {text-indent: -3em; margin-left: 6em}
```

creates three paragraph styles. The first creates a hanging indent that extends into the left margin. The second creates a conventional hanging indent. And the third creates a paragraph whose body is indented more than the hanging indent. All three styles are shown in use in Figure 8-6.

Figure 8-6. The effects of text-indent and margin-left on a paragraph



8.4.5.6 The text-shadow property

The text-shadow property lets you give your text a three-dimensional appearance through the time-honored use of shadowing. Values for the property include a required offset, and optional blur-radius and color. The property may include more than one set of values, separated with commas, to achieve a stack of shadows, with each subsequent set of values layered on top the previous one, but always beneath the original text.

The property's required offset is comprised of two length values. The first specifies the horizontal offset; the second specifies the vertical offset. Positive values place the shadow to the right and below the respective length distance from the text. Negative values move the shadow left and up, respectively.

The optional blur-radius also is a length value that specifies the boundaries for blurring, which effect depends on the rendering agent. The other shadow value is color. This, of course, may be an RGB triple or color name, as for other properties, and specifies the shadow color. Otherwise, text-shadow uses the color value of the color property.

```
h1 {text-shadow; 10px 10px 2px yellow}
p:first-letter {text-shadow: -5px -5px purple, 10px 10px orange}
```

The first text-shadow example puts a 2-pixel blurred-yellow shadow behind, 10 pixels below, and 10 pixels to the right of level-1 headers in your document. The second example puts two shadows behind the first letter of each paragraph. The purple shadow sits five pixels above and five pixels to the left of that first letter. The other shadow, like in the first example although orange in this case, goes 10 pixels to the right and 10 pixels below the first letter of each paragraph.

Sorry, we can't show you any of these effects since none of the popular browsers support this property, nor do they support the first-letter pseudo-element.

8.4.5.7 The vertical-align property

The `vertical-align` property controls the relative position of an element with respect to the line containing the element. Valid values for this property include:

`baseline`

Align the baseline of the element with the baseline of the containing element.

`middle`

Align the middle of the element with the middle (usually the x-height) of the containing element.

`sub`

Subscript the element.

`super`

Superscript the element.

`text-top`

Align the top of the element with the top of the font of the parent element.

`text-bottom`

Align the bottom of the element with the bottom of the font of the parent element.

`top`

Align the top of the element with the top of the tallest element in the current line.

`bottom`

Align the bottom of the element with the bottom of the lowest element in the current line.

In addition, a percentage value indicates a position relative to the current baseline, so that a position of `50%` puts the element halfway up the line height above the baseline. A position value of `-100%` puts the element an entire line-height below the baseline of the current line.

Netscape supports all but the `sub`, `super`, and `baseline` values only when this property is applied to the `<img>` tag. Internet Explorer supports only `sub` and `super` when applied to text elements.

8.4.5.8 The word-spacing property

Use the `word-spacing` property to add additional space between words within a tag. You can specify a length value or the keyword `normal` to revert to normal word spacing. For example:

```
h3 {word-spacing: 25px}
```

places an additional 25 pixels of space between words in the `<h3>` tag.

Netscape 6 supports the word-spacing property, but Internet Explorer does not.

8.4.5.9 The text-transform property

The text-transform property lets you automatically convert portions or all of your document's text into uppercase or lowercase lettering. Acceptable values are `capitalize`, `uppercase`, `lowercase`, or `none`.

`Capitalize` renders each first letter of each word in the text into uppercase, even if the source document's text is in lowercase. `Uppercase` and `lowercase` values repectively render all the text in the corresponding case. `None`, of course, cancels any transformations.

For example:

```
h1 {text-transform: uppercase}
```

makes all the letters in level-1 headers, presumably titles, appear in uppercase text, whereas:

```
h2 {text-transform: capitalize}
```

makes sure that each word in level-2 headers begins with a capital letter, appropriate for section heads, for instance.

Note that while uppercase and lowercase affect the entire text, `capitalize` affects only the first letter of each word in the text. Consequently, transforming the word htMl with `capitalize` will make it appear HtMl.

The `text-transform` property is supported by both Internet Explorer and Netscape Navigator.

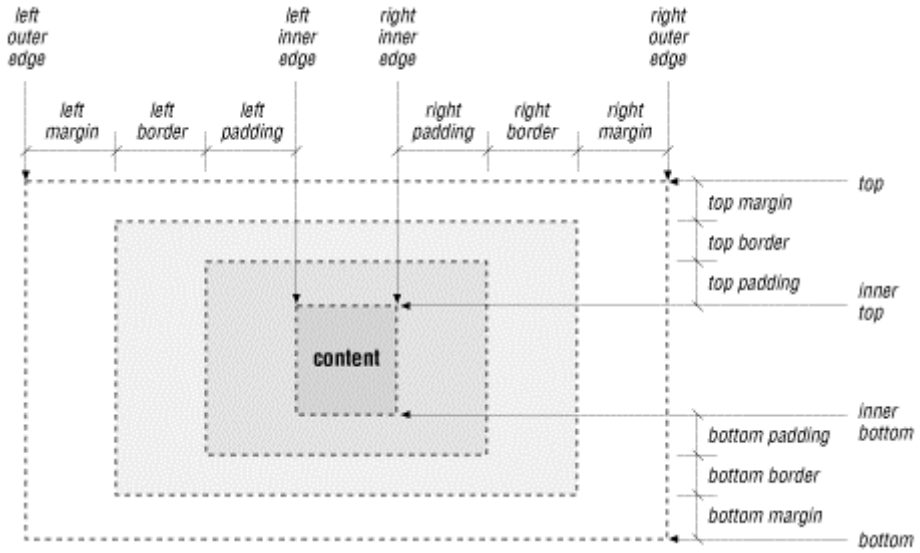
8.4.6 Box Properties

The CSS2 model assumes that HTML and XHTML elements always fit within a rectangular box. Using the properties defined in this section, you can control the size, appearance, and position of the boxes containing the elements in your documents.

8.4.6.1 The CSS2 formatting model

Each element in a document can fit in a rectangular box. The CSS2 authors call this box the "core content area" and surround it with three more boxes: the padding, the border, and the margin. Figure 8-7 shows these boxes and defines some useful terminology.

Figure 8-7. The CSS2 formatting model and terminology



The top, bottom, left-outer, and right-outer edges bound the content area of an element and all of its padding, border, and margin spaces. The inner-top, inner-bottom, left-inner, and right-inner edges define the sides of the core content area. The extra space around the element is the area between the inner and outer edges, including the padding, border, and margin. A browser may omit any and all of these extra spaces for any element, and for many, the inner and outer edges are the same.

When elements are vertically adjacent, the bottom margin of the upper elements and the top margin of the lower elements overlap, so that the total space between the elements is the greater of the adjacent margins. For example, if one paragraph has a bottom margin of one inch, and the next paragraph has a top margin of one-half inch, the greater of the two margins, one inch, will be placed between the two paragraphs. This practice is known as *margin collapsing* and generally results in better document appearance.

Horizontally adjacent elements do not have overlapping margins. Instead, the CSS2 model adds together adjacent horizontal margins. For example, if a paragraph has a left margin of 1 inch and is adjacent to an image with a right margin of 0.5 inch, the total space between the two will be 1.5 inches. This rule also applies to nested elements, so that a paragraph within a division will have a left margin equal to the sum of the division's left margin and the paragraph's left margin.

As shown in Figure 8-7, the total width of an element is equal to the sum of seven items: the left and right margins, the left and right borders, the left and right padding, and the element's content itself. The sum of these seven items must equal the width of the containing element. Of these seven items, only three (the element's width and its left and right margins) can be given the value `auto`, indicating that the browser can compute a value for that property. When this becomes necessary, the browser follows these rules:

- If none of these properties is set to `auto` and the total width is less than the width of the parent element, the `margin-right` property will be set to `auto` and made large enough to make the total width equal to the width of the parent element.

- If exactly one property is set to `auto`, that property will be made large enough to make the total width equal to the width of the parent element.

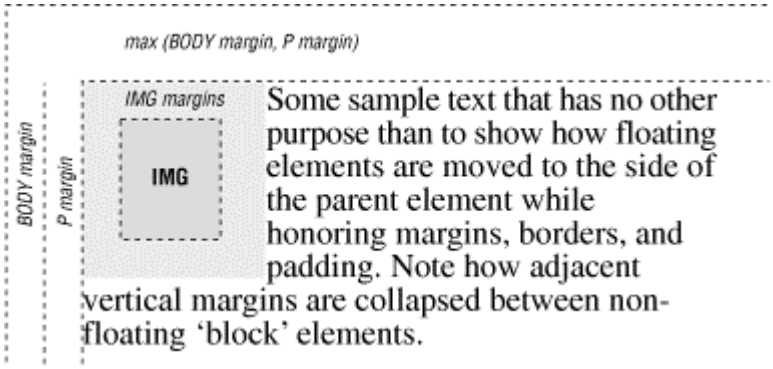
- If width, `margin-left` and `margin-right` are set to auto, the CSS2-compliant browser will set both `margin-left` and `margin-right` to zero and set `width` large enough to make the total equal to the width of the parent element.

- If both the left and right margins are set to `auto`, they will always be set to equal values, centering the element within its parent.

There are special rules for floating elements. A floating element (such as an image with `align=left` specified) will not have its margins collapsed with the margins of containing or preceding elements, unless the floating element has negative margins. Figure 8-8 shows how this bit of HTML might be rendered:

```
<body>
<p>
<img align=left src="pics/img.gif">
Some sample text...
</body>
```

Figure 8-8. Handling the margins of floating elements



The browser moves the image, including its margins, as far as possible to the left and towards the top of the paragraph without overlapping the left and top margins of the paragraph or the document body. The left margins of the paragraph and the containing body are added, while their top margins are collapsed.

8.4.6.2 The border properties

The border surrounding an element has a color, a thickness, and a style. You can use various properties to control these three aspects of the border on each of the four sides of an element. Shorthand properties make it easy to define the same color, thickness, and style for the entire border, if desired. Border properties are not inherited; you must explicitly set them for each element that has a border.

8.4.6.3 The border-color property

Use the `border-color` property to set the border color. If not specified, the browser draws the border using the value of the element's `color` property.

The `border-color` property accepts from one to four color values. The number of values determines how they are applied to the borders (summarized in Table 8-1). If you include just one property value, all four sides of the border are set to the specified color. Two values set the top and bottom borders to the first value and the left and right borders to the second value. With three values, the first is the top border, the second sets the right and left borders, and the third color value is for the bottom border. Four values specify colors for each side, clockwise from the top, then right, bottom, and left borders, in that order.

Table 8-1, Order of Effects for Multiple Border, Margin, and Padding Property Values

Number of Values	Affected Border(s), Margin(s), or Padding
1	All items have the same value.
2	First value sets *top* and *bottom*; second value sets *left* and *right*.
3	First value sets *top* ; second sets both *left* and *right*; third value sets *bottom*.
4	First value sets *top* ; second sets *right*; third sets *bottom*; fourth value sets *left*.

8.4.6.4 The border-width property

The `border-width` property lets you change the width of the border. Like the `border-color` property, it accepts from one to four values that are applied to the various borders in a similar manner (Table 8-1).

Besides a specific length value, you may also specify the width of a border as one of the keywords `thin`, `medium`, or `thick`. The default value, if the width is not explicitly set, is `medium`. Some typical border widths are:

```
border: 1px
border: thin thick medium
border: thick 2mm
```

The first example sets all four borders to exactly one pixel. The second makes the top border `thin`, the right and left borders `thick`, and the bottom border `medium`. The last example makes the top and bottom borders `thick`, while the right and left borders will be two millimeters wide.

If you are uncomfortable defining all four borders with one property, you can use the individual `border-top-width`, `border-bottom-width`, `border-left-width`, and `border-right-width` properties to define the thickness of each border. Each property accepts just one value; the default is `medium`.

Netscape Navigator and Internet Explorer 5 support this property even when used alone; Internet Explorer 4 honors this property only if borders are enabled through other border properties.

8.4.6.5 The border-style property

According to the CSS2 model, there are a number of embellishments that you may apply to your HTML element borders.

The `border-style`property values include `none` (default), `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, and `outset`. The border-style-conscious browser applies one to four values for the property to each of the borders in the same order as for the border colors and widths, as described in Table 8-1.

The browser draws `dotted`, `dashed`, `solid`, and `double` borders as flat lines atop the tag's background. The `groove`, `ridge`, `inset`, and `outset` values create three-dimensional borders: the `groove` is an incised line, the `ridge` is an embossed line, the `inset` border makes the entire tag area appear set into the document, and the `outset` border makes the entire tag area appear raised above the document. The effect of the three-dimensional nature of these last four styles upon the tag's background image is undefined and left up to the browser. Netscape supports three-dimensional effects.

Neither Internet Explorer nor Netscape supports the `dotted` or `dashed` values.

8.4.6.6 Borders in shorthand

Since specifying a complex border can get tedious, the CSS2 standard provides five shorthand properties that accept any or all of the width, color, and style values for one or all of the border edges. The `border-top`, `border-bottom`, `border-left`, and`border-right` properties affect their respective borders' sides; the comprehensive `border` property controls all four sides of the border simultaneously. For example:

```
border-top: thick solid blue
border-left: 1ex inset
border-bottom: blue dashed
border: red double 2px
```

The first property makes the top border a thick, solid, blue line. The second sets the left border to use an inset effect that is as thick as the x-height of the element's font, while leaving the color the same as the element's color. The third property creates a blue dashed line at the bottom of the element, using the default medium thickness. Finally, the last property makes all four borders a red double line two pixels thick.

That last property raises two issues. First, you cannot supply multiple values to the `border` property to selectively affect certain borders like you can with the individual `border-color`, `border-width`, and `border-style` properties. The `border` property always affects all four borders around an element.

Secondly, a bit of reflection should reveal that it is not possible to create a double-line border just two pixels thick. In cases like this, the browser is free to adjust the thickness to render the border properly.

While we usually think of borders surrounding block elements like images, tables, and text flows, borders can also be applied to inline tags. This lets you put a box around a word or phrase within a text flow. The implementation of borders on inline tags that span multiple lines is undefined and left to the browser.

Both Netscape and Internet Explorer support the `border` property, but only Internet Explorer supports the individual side properties.

8.4.6.7 The clear property

Like its cousin attribute for the
 tag, the clear property tells the browser whether to place a tag's contents adjacent to a "floating" element or on the first line below. Text flows around floating elements like images and tables with an align=left or align=right attribute or any HTML element with its float property set to anything but none. Section 4.7.1 / Section 8.4.6.8

The value of the clear property can be none, left, right, or both. A value of none, the default, means that the browser acts normally and places the tag's contents adjacent to floating elements on either side if there is room to do so. The value left prevents contents from being placed adjacent to a floating element on its left; right prevents placement against a floating element on the right; and both prevents the tag's contents from appearing adjacent to any floating element.

The effect of this style is the same as having preceded the tag with a
 tag with its clear attribute. Hence:

```
h1 {clear: left}
```

has the same effect as preceding every <h1> tag with <br clear=left>.

8.4.6.8 The float property

The float property designates a tag's display space as a floating element and causes text to flow around it in a specified manner. It is generally analogous to the align attribute for images and tables but can be applied to any element, including text, images, and tables. Section 5.2.6.4 / Section 10.2.1.1

The float property accepts one of three values: left, right, or none, the default. Using none disables the float property; the others work like their align attribute-value counterparts, telling the browser to place the content to either side of the flow and allow other content to be rendered next to it.

Accordingly, the styles-conscious browser will place a tag's contents specified with float: left against the left margin of the current text flow, and subsequent content will flow to its right, down and below the tag's contents. The float: right pair puts the tag contents against the right edge of the flow and flows other content on its left, down and below the tag's contents.

Although most commonly used with tables and images, it is perfectly acceptable to apply the float property to a text element. For example, the following would create a "run-in" header, with the text flowing around the header text:

```
h1 {float: left}
```

This property is supported by Internet Explorer only for images. Netscape honors it for textual elements as well.

8.4.6.9 The height property

As you might suspect, the height property controls the height of the associated tag's display region. You'll find it most often used with images and tables, but it can be used to control the height of other document elements as well.

The value of the height property is either a length value or the keyword auto, the default. Using auto implies that the affected tag has an initial height that should be used when displaying the tag. Otherwise, the height of the tag is set to the desired height. If an absolute value is used, the height is set to that length value. For example:

```
img {height: 100px}
```

tells the browser to display the image referenced by the <img> tag scaled so that it is 100 pixels tall. If you use a relative value, the base size to which it is relative is browser- and tag-dependent.

When scaling elements to a specific height, the aspect ratio of the object can be preserved by also setting the width property of the tag to auto. Thus:

```
img {height: 100px; width: auto}
```

ensures that the images are always 100 pixels tall, with an appropriately scaled width. Section 8.4.6.12

This property is fully supported by Internet Explorer and Netscape 6. Netscape Navigator 4 honors it only when used with the <img> tag.

8.4.6.10 The margin properties

Like the border properties, the various margin properties let you control the margin space around an element, just outside of its border (Figure 8-7). Margins are always transparent, allowing the background color or image of the containing element to show through. As a result, you can specify only the size of a margin; it has no color or rendered style.

The margin-left , margin-right, margin-top, and margin-bottom properties all accept a length or percentage value indicating the amount of space to reserve around the element. In addition, the keyword auto tells the styles-conscious browser to revert to the margins it normally would place around an element. Percentage values are computed as a percentage of the containing element's width. The default margin, if not specified, is zero.

These are all valid margin settings:

```
body {margin-left: 1in; margin-top: 0.5in; margin-right: 1in}
p {margin-left: -0.5cm}
img {margin-left: 10%}
```

The first example creates one-inch margins down the right and left edges of the entire document and a half-inch margin across the top of the document. The second example shifts the <p> tag one-half centimeter left into the left margin. The last example creates a margin to the left of the <img> tag equal to ten percent of the parent element's width.

Like the shorthand border property, you can use the shorthand margin property to define all four margins, using from one to four values which affect the margins in the order described in Table 8-1. Using this notation, our <body> margins in the previous example also could have been specified as:

```
body {margin: 0.5in 1in}
```

The margin-left and margin-right properties interact with the width property to determine the total width of an element, as described in Section 8.4.6.1.

8.4.6.11 The padding properties

Like the margin properties, the various padding properties let you control the padding space around an element, between the element's content area and its border (Figure 8-7). Padding is always rendered using the background color or image of the element. As a result, you can specify only the size of the padding; it has no color or rendered style.

The padding-left, padding-right, padding-top, and padding-bottomproperties all accept a length or percentage value indicating the amount of space the styles-conscious browser should reserve around the element. Percentage values are computed as a percentage of the containing element's width. The default padding is zero.

These are valid padding settings:

```
p {padding-left: 0.5cm}
img {padding-left: 10%}
```

The first example creates half a centimeter of padding between the contents of the <p> tag and its left border. The last example creates padding to the left of the <img> tag equal to ten percent of the parent element's width.

Like the shorthand margin and border properties, you can use the shorthand padding property to define all four padding amounts, using one to four values to effect the padding sides as described in Table 8-1. The padding property is not supported by Internet Explorer, but is supported by Netscape Navigator.

8.4.6.12 The width property

The width property is the companion to the height property and controls the width of an associated tag. Specifically, it defines the width of the element's content area, as shown in Figure 8-7. You'll see it most often used with images and tables, but you could conceivably use it to control the width of other elements as well.

The value for width property is either a length or percentage value or the keyword auto. The value auto is the default and implies that the affected tag has an initial width that should be used when displaying the tag. If a length value is used, the width is set to that value; percentage values compute the width to be a percentage of the width of the containing element. For example:

```
img {width: 100px}
```

displays the image referenced by the <img> tag scaled to 100 pixels wide.

When scaling elements to a specific width, the aspect ratio of the object is preserved if the height property of the tag is set to auto. Thus:

```
img {width: 100px; height: auto}
```

makes the images all 100 pixels wide and scales their heights appropriately. Section 8.4.6.9

The width property interacts with the margin-left and margin-right properties to determine the total width of an element, as described in Section 8.4.6.1.

8.4.7 List Properties

The CSS2 standard lets you also control the appearance of list elements - specifically, ordered and unordered lists.

Browsers format list items just like any other block item, except that the block has some sort of marker preceding the contents. For unordered lists, the marker is a bullet of some sort; for numbered lists, the marker is a numeric or alphabetic character or symbol. The CSS2 list properties let you control the appearance and position of the marker associated with a list item.

8.4.7.1 The list-style-image property

The `list-style-image` property defines the image that the browser uses to mark a list item. The value of this property is the URL of an image file, or the keyword `none`. The default value is `none`.

The image is the preferred list marker. If it is available, the browser will display it in place of any other defined marker. If the image is unavailable or if the user has disabled image loading, the marker defined by the `list-style-type` property (see Section 8.4.7.3) will be used.

Authors can use this property to define custom bullets for their unordered lists. While any image could conceivably be used as a bullet, we recommend that you keep your marker GIF or JPEG images small to ensure attractively rendered lists.

For example, by placing the desired bullet image in the file *mybullet.gif* on your server, you could use that image:

```
li {list-style-image: url(pics/mybullet.gif); list-style-type: square}
```

In this case, the image will be used if the browser successfully downloads *mybullet.gif*. Otherwise, the browser will use a conventional square bullet.

This property is supported by Internet Explorer and by the latest version of Netscape Navigator (version 6). However, they differ in where they position the list marker: Netscape 6 puts it outside, and Internet Explorer 5 puts it inside, the item. Read the next section for an explanation.

8.4.7.2 The list-style-position property

There are two ways to position the marker associated with a list item: inside the block associated with the item or outside the block. Accordingly, the `list-style-position` property accepts one of two values: `inside` or `outside`.

The default value is `outside`, meaning that the item marker will hang to the left of the item like this:

- This is a bulleted list with an "outside" marker

The value `inside` causes the marker to be drawn with the list item flowing around it, much like a floating image:

- This is a bulleted list with an "inside" marker

Notice how the second line of text is not indented, but instead lines up with the left edge of the marker.

None of the popular browsers support this property.

8.4.7.3 The list-style-type property

The `list-style-type` property serves double-duty in a sense, determining how both ordered and unordered list items are rendered by a styles-conscious browser. This property has the same effect on a list item as its `type` attribute does. Section 7.3.1.1

When used with items within an unordered list, the `list-style-type` property accepts one of four values: `disc`, `circle`, `square`, or `none`. The browser marks the unordered list items with the corresponding specified dingbat. The default value is `disc`; browsers change that default depending on the nesting level of the list.

When used with items within an ordered list, the `list-style-type` property accepts one of six values: `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`, or `none`. These values format the item numbers as decimal values, lowercase Roman numerals, uppercase Roman numerals, lowercase letters, or uppercase letters, respectively. Most browsers will use decimal numbering schemes if you don't set this property.

8.4.7.4 The list-style property

The list-style property is the shorthand version for all the other list-style properties. It accepts any or all of the values allowed for the list-style-type, list-style-position, and list-style-image properties, in any order and with values appropriate for the type of list they are to affect. These are valid list-style properties:

```
li {list-style: disc}
li {list-style: lower-roman inside}
li {list-style: url(http://www.kumquat.com/images/tiny-quat.gif)
     square}
```

The first example creates list items that use a disc as the bullet image. The second causes numbered list items to use lowercase Roman numerals, drawn inside the list item's block. In the last example, a square will be used as the bullet image if the referenced image is unavailable.

8.4.7.5 Using list properties effectively

Although you may apply list properties to any element, they will affect only the appearance of elements whose display property is set to list-item. Normally, the only tag with this property is the <li> tag. Section 8.4.8.1

This shouldn't deter you from using these properties elsewhere, particularly with the <ul> and <ol> tags. Since these properties are inherited by elements whose parents have them set, modifying a list property for the <ul> and <ol> tags will subsequently modify it for all the <li> tags contained within that list. This makes it much easier to define lists with a particular appearance.

For example, suppose you want to create a list style that uses lowercase Roman numerals. One way is to define a class of the <li> tag with the appropriate list-style-type defined:

```
li.roman {list-style-type: lower-roman}
```

Within your list, you'll need to specify each list element using that class:

```
<ol>
  <li class=roman>Item one</li>
  <li class=roman>Item two</li>
  <li class=roman>And so forth</li>
</ol>
```

Having to repeat the class name is tedious and error-prone. A better solution is to define a class of the <ol> tag:

```
ol.roman {list-style-type: lower-roman}
```

Any <li> tag within the list will inherit the property and use lowercase Roman numerals:

```
<ol class=roman>
  <li>Item one</li>
  <li>Item two</li>
  <li>And so forth</li>
</ol>
```

This is much easier to understand and manage. If at a later date you want to change the numbering style, you need only change the <ol> tag properties, rather than find and change each instance of the <li> tag in the list.

You can use these properties in a much more global sense as well. Setting a list property on the <body> tag will change the appearance of all lists in the document; setting it on a tag will change all the lists within that division.

8.4.8 Classification Properties

Classification properties are the most esoteric of the CSS2 style properties. They do not directly control how a styles-conscious browser will render HTML or XHTML elements. Instead, they tell the browser how to classify and handle various tags and their contents as they are encountered.

For the most part, you should not set these properties on an element unless you are trying to achieve a specific effect. Even then, it is unlikely that the property will be supported by most browsers.

8.4.8.1 The display property

Every element in an HTML or XHTML document can be classified, for display purposes, as a block item, an inline item, or a list item. Block elements, like headings, paragraphs, tables, and lists, are formatted as a separate block of text, separated from the previous and next block items. Inline items, like the physical and content-based style tags and hyperlink anchors, are rendered within the current line of text within a containing block. List items, specifically the <li> tag, are rendered like a block item, with a bullet or number known as the *marker*.

The display property lets you change an element's display type to block, inline, list-item, or none. The first three values change the element's classification accordingly; the value none turns off the element, preventing it or its children from being displayed in the document.

Conceivably, you could wreak all sorts of havoc by switching element classifications, forcing paragraphs to be displayed as list items and converting hyperlinks to block elements. In practice, this is just puerile monkey business, and we don't recommend that you change element classifications without having a very good reason to do so.

Netscape Navigator fully supports this property; Internet Explorer supports only the `block` and `none` values.

8.4.8.2 The white-space property

The `white-space` property defines how the styles-conscious browser treats whitespace (tabs, spaces, and carriage returns) within a block tag. The keyword value `normal` - the default - collapses whitespace, so that one or more spaces, tabs, and carriage returns are treated as a single space between words. The value `pre` emulates the `<pre>` tag, in that the browser retains and displays all spaces, tabs, and carriage returns. And, finally, the `nowrap` value tells the browser to ignore carriage returns and not insert automatic line breaks; all line-breaking must be done with explicit `<br>` tags.

Like the `display` property, the `white-space` property is rarely used for good instead of evil. Don't change how elements handle whitespace without having a compelling reason for doing so.

This property is not supported by Internet Explorer; Netscape Navigator supports the `pre` and `normal` values.

8.5 Tag-less Styles: The Tag

Up to now, we have used Cascading Style Sheets to change the appearance of content that is within a designated tag. In some cases, however, you may want to alter the appearance of only a portion of a tag's contents - usually text. Designate these special segments with the `<span>` tag.

Function:

Delimit arbitrary amount of text

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

; never omitted

Contains:

html_content

Used in:

body_content

The tag simply delimits a portion of content (constrained by normal tag nesting rules, of course). Browsers treat the tag as another physical or content-based style tag. The only difference, of course, is that the default meaning of the tag is to leave the text alone.

Although it may serve some other function in a future version of HTML, the tag was introduced so that you can apply style, display, and event management to an arbitrary section of document content. Display and event management are addressed later; to define a style for the tag, treat it like any other HTML tag:

```
span {color: purple}
span.bigger {font-size: larger}
```

and use it like any other style tag:

```
Quat harvest projections are <span class=bigger>bigger than ever</span>!
```

In a similar manner, the appearance of a tag can be changed using an inline style:

```
Quat harvest projections are <span style="font-size: larger">bigger than ever</span>!
```

Like any other physical or content-based style tag, tags can be nested and may contain other tags.

Although deprecated, the tag also supports the standard tag attributes. The style and class attributes, of course, let you control the display style; the id and title tag attributes let you uniquely label its contents; the dir and lang attributes let you specify its native language; and the many on-event attributes let you react to user-initiated mouse and keyboard actions on the contents. Not all are implemented by the currently popular browsers for this tag or for many others. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.4 / Section 4.1.1.5 / Section 8.1.1 / Section 8.3 / Section 12.3.3

8.6 Applying Styles to Documents

There are several issues you should consider before, during, and after you use styles in your web documents and document collections. The first, overarching issue is whether to use them at all. Frankly, few of the style effects are unique; most can be achieved, albeit less easily and with much less consistency, via the physical and content-based style tags like <i> and and the various tag attributes like color and background.

8.6.1 To Style or Not to Style

We think the CSS2 standard is a winner, not only over just JavaScript-based standards, but mostly for the convenience and effectiveness of all your mark-up documents, including HTML, XHTML, and any and most other XML-compliant ones. The majority of browsers in use today support CSS2 styles. The benefits are clear. So, why wouldn't you use styles?

Although we strongly urge that you learn and use CSS2 style sheets for your documents, we realize that creating style sheets is an investment of time and energy that pays off over the long run. Designing a style sheet for a one- or two-page document is probably not time-effective, particularly if you won't be reusing the style sheet for any other documents. In general, however, we believe the choice is not *if* you should use CSS2 style sheets, but when.

8.6.2 Which Type of Style Sheet and When

Once you have decided to use Cascading Style Sheets (for pain or pleasure), the next question is which type of style sheet - inline, document-level, or external - should you apply and when? Each has its pros and cons; each is best applied under certain circumstances.

8.6.2.1 The pros and cons of external styles

Since style sheets provide consistency in the presentation of your documents, external style sheets are the best and the easiest way to manage styles for your entire document collection. Simply place the desired style rules in a style sheet and apply those styles to the desired documents. And since all the documents are affected by a single style sheet, conversion of the entire collection to a new style is as simple as changing a single rule in the corresponding external style sheet.

Even in cases where documents may differ in style, it is often possible to collect a few basic style rules in a single sheet that can be shared among several otherwise different documents, including:

- Background color
- Background image
- Font sizes and faces
- Margins
- Text alignment

Another benefit of external style sheets is that other web authors who want to copy your style can easily access that sheet and make their pages look like yours. Imitation being the sincerest form of flattery, you should not be troubled when someone elects to emulate the look and feel of your pages. More to the point, you can't stop them from linking to your style sheets, so you might as well learn to like it. Like conventional HTML documents, it is not possible to encrypt or otherwise hide your style sheets so that others cannot view and use them.

The biggest problem with external style sheets is that they increase the amount of time needed to access a given web page. Not only must the browser download the page itself, it must also download the style sheet before the page can be displayed to the user. While most style sheets are relatively small, their existence can definitely be felt when accessing the Web over a slow connection.

Without appropriate discipline, external style sheets can become large and unwieldy. When creating style sheets, remember to include only those styles that are common to the pages using the sheet. If a set of styles is needed only for one or two sheets, you are better off isolating them in a separate sheet or adding them to a document using document-level styles. Otherwise, you may find yourself expending an exorbitant amount of effort counteracting the effects of external styles in many individual documents.

8.6.2.2 The pros and cons of document-level styles

Document-level styles are most useful when creating a custom document. They let you override one or more rules in your externally defined style to create a slightly different document.

You might also want to use document-level styles to experiment with new style rules before moving them to your style sheets. By adding and changing rules using document-level styles, you eliminate the risk of adding a broken style to your style sheets, breaking the appearance of all the documents that use that sheet.

The biggest problem with document styles is that you may succumb to using them in lieu of creating a formal, external style sheet to manage your document collection. It is easy to simply add rules to each document, cutting and pasting as you create new documents. Unfortunately, managing a collection of documents with document-level styles is tedious and error-prone. Even a simple change can result in hours of editing and potential mistakes.

As a rule of thumb, any style rule that impacts three or more documents should be moved to a style sheet and applied to those documents using the `<link>` tag or `@import` at-rule. Adhering to this rule as you create your document families will pay off in the long run when it is time to change your styles.

8.6.2.3 The pros and cons of inline styles

And at the end of the cascade, inline styles override the more general styles. Get into the habit now of using inline styles rarely and just for that purpose, too. Inline styles cannot be reused, making style management difficult. Moreover, such changes are spread throughout your documents, making finding and altering inline styles error-prone. (That's why we might eschew tag- and attribute-based styles in the first place, no?)

Any time you use an inline style, think long and hard as to whether the same effect might be accomplished using a style class definition. For instance, you are better off defining:

```
<style type="text/css">
<!--
  p.centered {text-align: center}
  em.blue {color: blue}
-->
</style>
```

and later using:

```
<p class=centered>
<em class=blue>
```

instead of:

```
<p style="text-align: center">
<em style="color: blue">
```

Your styles are easier to find and manage and can be easily reused throughout your documents.

Chapter 9. Forms

Forms, forms, forms, forms: we fill 'em out for nearly everything, from the moment we're born, 'til the moment we die. Pretty mundane, really. So what's to explain all the hoopla and excitement over forms? Simply this: they make HTML and, of course, XHTML truly interactive.

When you think about it, interacting with a web page is basically a lot of button pushing: click here, click there, go here, go there - there's no real user feedback, and it's certainly not personalized. Applets provide extensive user-interaction capability, but they can be difficult to write and are still not standardized for all browsers. Forms, on the other hand, are supported by almost every browser and make it possible to create documents that collect and process user input and to formulate personalized replies.

This powerful mechanism has far-reaching implications, particularly for electronic commerce. It finishes an online catalog by giving buyers a way to immediately order products and services. It gives nonprofit organizations a way to sign up new members. It lets market researchers collect user data. It gives you an automated way to interact with your readers.

Mull over the ways you might want to interact with your readers while we take a look at both the client- and server-side details of creating forms.

9.1 Form Fundamentals

Forms are comprised of one or more text input boxes, clickable buttons, multiple-choice checkboxes, and even pull-down menus and image maps, all placed inside the `<form>` tag. You can have more than one form in a document, and within each you may also put regular body content, including text and images. The text is particularly useful for providing instructions to the users on how to fill out the form and for form element labels and prompts. And, within the various form elements, you can use JavaScript event handlers for a variety of effects like testing and verifying form contents and calculating a running sum.

A user fills out the various fields in the form, then clicks a special "Submit" button (or, sometimes, presses the Enter or Return key) to submit the form to a server. The browser packages up the user-supplied values and choices and sends them to a server or to an email address.[1] The server passes the information along to a supporting program or application that processes the information and creates a reply, usually in HTML. The reply may be simply a thank you or it might prompt the user on how to fill out the form correctly or to supply missing fields. The server sends the reply to the browser client, which then presents it to the user. With emailed forms, the information is simply put into someone's mailbox; there is no notification of the form being sent.

> [1] Some browsers, Netscape and Internet Explorer in particular, may also encrypt the information, securing it from credit-card thieves, for example. However, the encryption facility must also be supported on the server-side as well: contact the web server manufacturer for details.

The server-side, data-processing aspects of forms are not part of the HTML or XHTML standards; they are defined by the server's software. While a complete discussion of server-side forms programming is beyond the scope of this book, we'd be remiss if we did not include at least a simple example to get you started. To that purpose, we've included at the end of this chapter a few skeletal programs that illustrate some of the common styles of server-side forms programming.

9.2 The <form> Tag

Place a form anywhere inside the body of a document with its elements enclosed by the `<form>` tag and its respective end tag `</form>`. You can, and we recommend you often do, include regular body content inside a form to specially label user-input fields and to provide directions.

Browsers flow the special form elements into the containing paragraphs as if they were small images embedded into the text. There aren't any special layout rules for form elements, so you need to use other elements, like tables and style sheets, to control the placement of elements within the text flow.

You must define at least two special form attributes, which provide the name of the form's processing server and the method by which the parameters are to be sent to the server. A third, optional attribute lets you change how the parameters get encoded for secure transmission over the network.

<form>

Function:

Defines a form

Attributes:

ACCEPT	
ACCEPT-CHARSET	ONKEYPRESS
ACTION	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ENCTYPE	ONMOUSEOUT
ID	ONMOUSEOVER
LANG	ONMOUSEUP
METHOD	ONRESET
NAME	ONSUBMIT
ONCLICK	STYLE
ONDBLCLICK	TARGET
ONKEYDOWN	TITLE

End tag:

</form>; never omitted

Contains:

form_content

Used in:

block

9.2.1 The action Attribute

The required `action` attribute for the `<form>` tag gives the URL of the application that is to receive and process the form's data.

Most webmasters keep their forms-processing applications in a special directory on their web server, usually named *cgi-bin*, which stands for Common Gateway Interface-binaries.[2] Keeping these special forms-processing programs and applications in one directory makes it easier to manage and secure the server.

[2] The Common Gateway Interface (CGI) defines the protocol by which servers interact with programs that process form data.

A typical <form> tag with the action attribute looks like this:

```
<form action="http://www.kumquat.com/cgi-bin/update">
...
</form>
```

The example URL tells the browser to contact the web server named *www* in the *kumquat.com* domain and pass along the user's form values to the application named *update* located in the *cgi-bin* directory.

In general, if you see a URL that references a document in a directory named *cgi-bin*, you can be pretty sure that the document is actually an application that creates the desired page dynamically each time it's invoked.

9.2.2 The enctype Attribute

The browser specially encodes the form's data before it passes that data to the server so that it does not become scrambled or corrupted during the transmission. It is up to the server either to decode the parameters or to pass them, still encoded, to the application.

The standard encoding format is the Internet Media Type " application/x-www-form-urlencoded." You can change that encoding with the optional enctype attribute in the <form> tag. The only optional encoding formats currently supported are "multipart/form-data" and "text/plain."

The multipart/form-data alternative is required for those forms that contain file-selection fields for upload by the user. The text/plain format should be used in conjunction with a mailto URL in the action attribute for sending forms to an email address instead of a server. Unless your forms need file-selection fields or you must use a mailto URL in the action attribute, you probably should ignore this attribute and simply rely upon the browser and your processing server to use the default encoding type. Section 9.5.1.3

9.2.2.1 The application/x-www-form-urlencoded encoding

The standard encoding - application/x-www-form-urlencoded - converts any spaces in the form values to a plus sign (+), nonalphanumeric characters into a percent sign (%) followed by two hexadecimal digits that are the ASCII code of the character, and the line breaks in multiline form data into %0D%0A.

The standard encoding also includes a name for each field in the form. (A "field" is a discrete element in the form, whose value can be nearly anything from a single number to several lines of text - the user's address, for example.) If there is more than one value in the field, the values are separated by ampersands.

For example, here's what the browser sends to the server after the user fills out a form with two input fields labeled name and address; the former field has just one line of text, while the latter field has several lines of input:

```
name=O'Reilly+and+Associates&address=101+Morris+Street%0D%0A
Sebastopol,%0D%0ACA+95472
```

We've broken the value into two lines for clarity in this book, but in reality, the browser sends the data in an unbroken string. The name field is "O'Reilly and Associates" and the value of the address field, complete with embedded newline characters, is:

```
101 Morris Street
Sebastopol,
CA 95472
```

9.2.2.2 The multipart/form-data encoding

The multipart/form-data encoding encapsulates the fields in the form as several parts of a single MIME-compatible compound document. Each field has its own section in the resulting file, set off by a standard delimiter. Within each section, one or more header lines define the name of the field, followed by one or more lines containing the value of the field. Since the value part of each section can contain binary data or otherwise unprintable characters, no character conversion or encoding occurs within the transmitted data.

This encoding format is by nature more verbose and longer than the application/x-www-form-urlencoded format. As such, it can be used only when the method attribute of the <form> tag is set to post, as described in Section 9.2.4.

A simple example makes it easy to understand this format. Here's our previous example, when transmitted as multipart/form-data:

```
-----------------------------146931364513459
Content-Disposition: form-data; name="name"

O'Reilly and Associates
-----------------------------146931364513459
Content-Disposition: form-data; name="address"

101 Morris Street
Sebastopol,
CA 95472
-----------------------------146931364513459--
```

The first line of the transmission defines the delimiter that will appear before each section of the document. It always consists of thirty dashes and a long random number that distinguishes it from other text that might appear in actual field values.

The next lines contain the header fields for the first section. There will always be a `Content-Disposition` field indicating the section contains form data and providing the name of the form element whose value is in this section. You may see other header fields; in particular, some file-selection fields include a `Content-Type` header field that indicates the type of data contained in the file being transmitted.

After the headers, there is a single blank line followed by the actual value of the field on one or more lines. The section concludes with a repeat of the delimiter line that started the transmission. Another section follows immediately, and the pattern repeats until all of the form parameters have been transmitted. The end of the transmission is indicated by an extra two dashes at the end of the last delimiter line.

As we pointed out earlier, use multipart/form-data encoding only when your form contains a file-selection field. Here's an example of how the transmission of a file-selection field might look:

```
-----------------------------146931364513459
Content-Disposition: form-data; name="thefile"; filename="test"
Content-Type: text/plain

First line of the file
...
Last line of the file
-----------------------------146931364513459--
```

The only notable difference is that the `Content-Disposition` field contains an extra element, `filename`, that defines the name of the file being transmitted. There might also be a `Content-Type` field to further describe the file's contents.

9.2.2.3 The text/plain encoding

Use this encoding only when you don't have access to a form-processing server and need to send the form information by email (the form's `action` attribute is a `mailto` URL). The conventional encodings are designed for computer consumption; text/plain is designed with people in mind.

In this encoding, each element in the form is placed on a single line, with the name and value separated by an equal sign. Returning to our name and address example, the form data would be returned as:

```
name=O'Reilly and Associates
address=101 Morris Street%0D%0ASebastopol,%0D%0ACA 95472
```

As you can see, the only characters still encoded in this form are the carriage return and line feed characters in multiline text input areas. Otherwise, the result is easily readable and generally parsable by simple tools.

9.2.3 The accept-charset Attribute

The `accept-charset` attribute was introduced in the HTML 4.0 standard. It lets you specify a list of character sets that the server must support to properly interpret the form data. The value of this attribute is a quote-enclosed list of one or more ISO character set names. The browser may choose to disregard the form or handle it differently if the acceptable character sets do not match the character set in use by the user. The default value of this attribute is `unknown`, implying that the form character set is the same as the document containing the form.

9.2.4 The method Attribute

The other required attribute for the `<form>` tag sets the method by which the browser sends the form's data to the server for processing. There are two ways: the POST method and the GET method.

With the POST method, the browser sends the data in two steps: the browser first contacts the form-processing server specified in the `action` attribute and, once contact is made, sends the data to the server in a separate transmission.

On the server side, POST-style applications are expected to read the parameters from a standard location once they begin execution. Once read, the parameters must be decoded before the application can use the form values. Your particular server will define exactly how your POST-style applications can expect to receive their parameters.

The GET method, on the other hand, contacts the form-processing server and sends the form data in a single transmission step: the browser appends the data to the form's `action` URL, separated by the question mark character.

The common browsers transmit the form information by either method; some servers receive the form data by only one or the other method. You indicate which of the two methods - POST or GET - your forms-processing server handles with the `method` attribute in the `<form>` tag. Here's the complete tag including the GET transmission `method` attribute for the previous form example:

```
<form method=GET
   action="http://www.kumquat.com/cgi-bin/update">
  ...
</form>
```

9.2.4.1 POST or GET?

Which one to use if your form-processing server supports both the POST and GET methods? Here are some rules of thumb:

- For best form-transmission performance, send small forms with a few short fields via the GET method.
- Because some server operating systems limit the number and length of command-line arguments that can be passed to an application at once, use the POST method to send forms that have many fields or that have long text fields.
- If you are inexperienced in writing server-side form-processing applications, choose GET. The extra steps involved in reading and decoding POST-style transmitted parameters, while not too difficult, may be more than you are willing to tackle.
- If security is an issue, choose POST. GET places the form parameters directly in the application URL where they easily can be captured by network sniffers or extracted from a server log file. If the parameters contain sensitive information like credit card numbers, you may be compromising your users without their knowledge. While POST applications are not without their security holes, they can at least take advantage of encryption when transmitting the parameters as a separate transaction with the server.
- If you want to invoke the server-side application outside the realm of a form, including passing it parameters, use GET because it lets you include form-like parameters as part of a URL. POST-style applications, on the other hand, expect an extra transmission from the browser after the URL, something you can't do as part of a conventional `<a>` tag.

9.2.4.2 Passing parameters explicitly

The foregoing bit of advice warrants some explanation. Suppose you had a simple form with two elements named x and y. When the values of these elements are encoded, they look like this:

```
x=27&y=33
```

If the form uses `method=GET`, the URL used to reference the server-side application looks something like this:

```
http://www.kumquat.com/cgi-bin/update?x=27&y=33
```

There is nothing to keep you from creating a conventional `<a>` tag that invokes the form with any parameter value you desire, like so:

```
<a href="http://www.kumquat.com/cgi-bin/update?x=19&y=104">
```

The only hitch is that the ampersand that separates the parameters is also the character-entity insertion character. When placed within the `href` attribute of the `<a>` tag, the ampersand will cause the browser to replace the characters following it with a corresponding character entity.

To keep this from happening, you must replace the literal ampersand with its entity equivalent, either `&` or `&`. With this substitution, our example of the nonform reference to the server-side application looks like this:

```
<a href="http://www.kumquat.com/cgi-bin/update?x=19&amp;y=104">
```

Because of the potential confusion that arises from having to escape the ampersands in the URL, server implementors are encouraged to also accept the semicolon as a parameter separator. You might want to check your server's documentation to see if the server honors this convention. See Appendix F.

9.2.5 The target Attribute

With the advent of frames, it is possible to redirect the results of a form to another window or frame. Simply add the `target` attribute to your `<form>` tag and provide the name of the window or frame to receive the results.

Like the `target` attribute used in conjunction with the `<a>` tag, you can use a number of special names with the `target` attribute in the `<form>` tag to create a new window or to replace the contents of existing windows and frames. Section 11.7.1

9.2.6 The id, name, and title Attributes

The `id` attribute lets you attach a unique string label to your form for reference by programs (applets) and hyperlinks. Before `id` was introduced in HTML 4.0, Netscape Navigator used the `name` attribute to achieve similar effects, although it cannot be used in a hyperlink. To be compatible with the broadest range of browsers, we recommend that for now you include both `name` and `id` with `<form>`, if needed. In the future, you should use only the `id` attribute for this purpose.

The `title` attribute defines a quote-enclosed string value to label the form. However, it entitles only the form segment; its value cannot be used in an applet reference or hyperlink. Section 4.1.1.4 / Section 4.1.1.5

9.2.7 The class, style, lang, and dir Attributes

The `style` attribute creates an inline style for the elements enclosed by the form, overriding any other style rule in effect. The `class` attribute lets you format the content according to a predefined class of the `<form>` tag; its value is the name of that class. Section 8.1.1 / Section 8.3

The actual effects of `style` with `<form>` are hard to predict, however. In general, style properties affect the body content - text, in particular - that you may include as part of the form's contents, but `<form>` styles do affect the display characteristics of the form elements.

For instance, you may create a special font face and background color style for the form. The form's text labels, but not the text inside a text input form element, will appear in the specified font face and background color. Similarly, the text labels you put beside a set of radio buttons will be in the form-specified style, but not the radio buttons themselves.

The `lang` attribute lets you specify the language used within the form, with its value being any of the ISO standard two-character language abbreviations, including an optional language modifier. For example, adding `lang=en-UK` tells the browser that the list is in English ("en") as spoken and written in the United Kingdom (UK). Presumably, the browser may make layout or typographic decisions based upon your language choice.

Similarly, the `dir` attribute tells the browser which direction to display the list contents, from left to right (`dir=ltr`) like English or French, or from right to left (`dir=rtl`), such as with Hebrew or Chinese.

The `dir` and `lang` attributes are supported by the popular browsers, even though there are no behaviors defined for any specific language. Section 3.6.1.1 / Section 3.6.1.2

9.2.8 The Event Attributes

As for most other elements in a document, the `<form>` tag honors the standard mouse and keyboard event-related attributes the compliant browser will recognize. We describe the majority of these attributes in detail in Chapter 12. Section 12.3.3

Forms have two special event-related attributes: `onSubmit` and `onReset`. The value of these event attributes is - enclosed in quotation marks - one or a sequence of semicolon-separated JavaScript expressions, methods, and function references. With `onSubmit`, the browser executes these commands before it actually submits the form's data to the server or sends it to an email address.

You may use the `onSubmit` event for a variety of effects. The most popular is for a client-side form-verification program that scans the form data and prompts the user to complete one or more missing elements. Another popular and much simpler use is to inform users when a `mailto` URL form is being processed via email.

The `onReset` attribute is used just like the `onSubmit` attribute, except that the associated program code is executed only if the user presses a "Reset" button in the form.

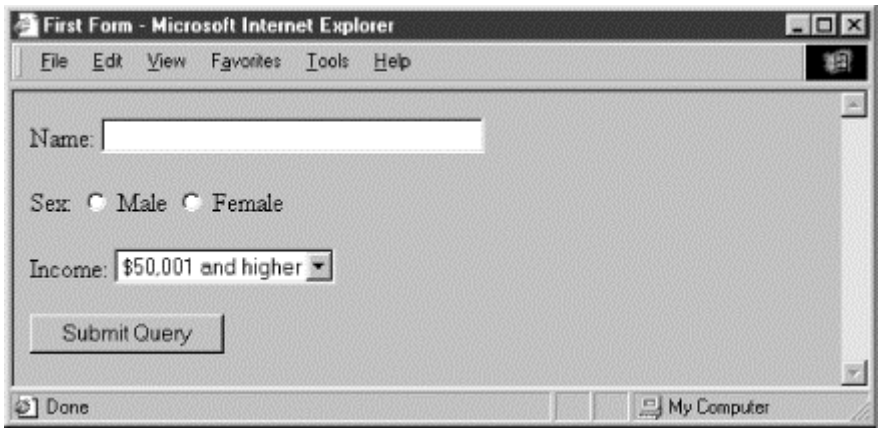
9.3 A Simple Form Example

In a moment we'll examine each of the many form controls in detail. Let's first take a quick look at a simple example to see how forms are put together. This one (shown in Figure 9-1) gathers basic demographic information about a user:

```
<form method=POST action="http://www.kumquat.com/demo">
  Name:
    <input type=text name=name size=32 maxlength=80>
  <p>
  Sex:
    <input type=radio name=sex value="M"> Male
    <input type=radio name=sex value="F"> Female
  <p>
  Income:
    <select name=income size=1>
      <option>Under $25,000
      <option>$25,001 to $50,000
      <option>$50,001 and higher
    </select>
  <p>
  <input type=submit>
</form>
```

The first line of the example starts the form and indicates we'll be using the POST method for data transmission to the form-processing server. The form's user-input controls follow, each defined by an `<input>` tag and `type` attribute. There are three controls in the simple example, each contained within its own paragraph.

Figure 9-1. A simple form



The first control is a conventional text entry field, letting the user type up to 80 characters, but displaying only 32 of them at a time. The next one is a multiple-choice option, which lets the user select only one of two radio buttons. This is followed by a pull-down menu for choosing one of three options. The final control is a simple submission button, which, when clicked by the user, sets the form's processing in motion.

9.4 Using Email to Collect Form Data

It is increasingly common to find authors who have no access to a web server other than to upload their documents. Consequently, they have no ability to create or manage CGI programs. In fact, some Internet Service Providers (ISPs), particularly those hosting space for hundreds or even thousands of sites, typically disable CGI services to limit their server's processing load or as a security precaution.

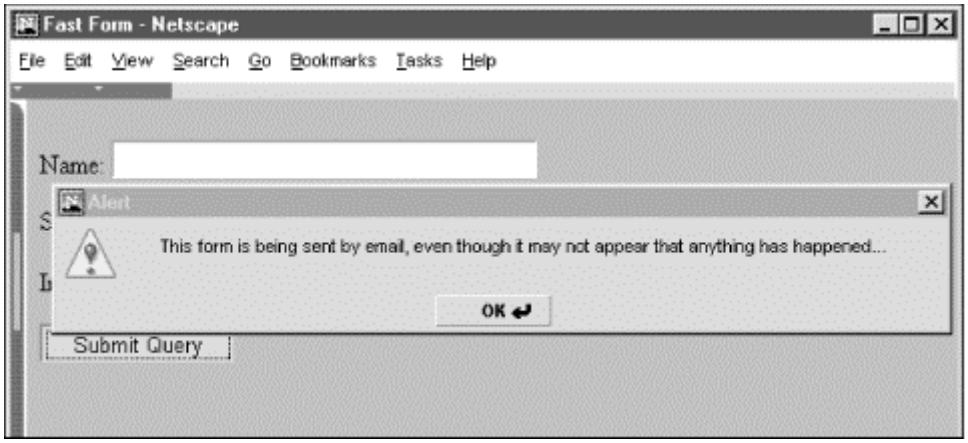
If you are working with one of the many sites where you cannot get a form processed to save your life, all is not lost: you can use a mailto URL as the value of the form's `action` attribute. The latest browsers automatically will email the various form parameters and values to the address supplied in the URL. The recipient of the mail can then process the form and take action accordingly.

By substituting the following for the `<form>` tag in our previous example:

```
<form method=POST action="mailto:chuckandbill@oreilly.com"
    enctype="text/plain"
    onSubmit="window.alert('This form is being sent by email, even
    though it may not appear that anything has happened...')">
```

the form data gets emailed to `chuckandbill` when submitted by the user, not otherwise processed by a server. Notice, too, that we have a simple JavaScript alert message that appears when the browser gets ready to send out the form data. The alert tells the user not to expect confirmation that the form data was sent (see Figure 9-2). Also, unless disabled by the user or if you omit the `method=POST` attribute, the browser typically will warn the user that they are about to send unencrypted ("text/plain") and thereby unsecured information over the network and gives them the option to cancel the submission. Otherwise, the form is sent via email without incident or notification.

Figure 9-2. A warning about a mailto form submission



The body of the resulting emailed form message looks something like this:

```
name=wild bill
sex=M
income=Under $25,000
```

9.4.1 Problems with Email Forms

If you choose to use either mailto or a form-to-email facility, there are several problems you may have to deal with:

- Your forms won't work on browsers that don't support a mailto URL as a form action.
- Some browsers, including some early versions of Internet Explorer, do not properly place the form data into the email message body and may even open an email dialog box, confusing the user.
- A mailto doesn't present users with a confirmation page to assure them that their form has been processed. After executing the mailto form, the user is left looking at the form, as if nothing had happened. (Use JavaScript to overcome this dilemma with an `onSubmit` or `onClick` event handler.) Section 12.3.3
- Your data may arrive in a form that is difficult, if not impossible, to read, unless you use a readable `enctype`, such as `text/plain`.
- You will lose whatever security protections that may have been provided by the server with the form.

The last problem deserves additional explanation. Some web providers support special "secure" web servers that attach an encryption key to your web page when sent to the user browser. The popular browsers use that key to encrypt any data your document may send back to that same server, including the user's form data. Since only the client's browser and the server know the key, only that server will be able to decipher the information coming back to it from the client browser, effectively securing the information from nefarious eavesdroppers and hackers.

However, if you use email to retrieve the form data, the server decrypts it before packaging the form information into the body of an email message and sending it to you. Email normally is highly susceptible to eavesdropping and other types of snooping. Its contents are very insecure.

So, please, if you use an email method to retrieve sensitive form data, such as credit cards and personal information, be aware of the potential consequences. And don't be fooled or fool your users with a "secure" server when insecure email comes out the back end.

In spite of all these problems, email forms present an attractive alternative to the web author constrained by a restricted server. Our advice: use CGI scripts if at all possible and fall back on mailto URLs if all else fails.

9.5 The <input> Tag

Use the `<input>` tag to define any one of a number of common form "controls," as they are called in the HTML 4 and XHTML standards, including text fields, multiple-choice lists, clickable images, and submission buttons. Although there are many attributes for the `<input>` tag, only the `type` and `name` attributes are required for each element (only `type` for a submission or reset button; see following explanation). And as we describe in detail later, each type of input control uses only a subset of the allowed attributes. Additional `<input>` attributes may be required based upon which type of form element you specify.

Figure 9-Table1 summarizes the various form `<input>` types and attributes, required and optional.

Figure 9-Table1. Required and Some Common Form Element Attributes

Attributes (× = required; ▲ = optional; *blank* = not supported)

Form Tag or <input> Type	accept	accesskey	align	alt	border	cols	checked	disabled	maxlength	multiple	name	notab	onBlur	onChange	onClick	onFocus	onSelect	readonly	rows	size	src	tabindex	taborder	usemap	value	wrap
button		▲						▲			×	▲	▲		▲	▲						▲	▲		×	
checkbox		▲					▲	▲			×	▲			▲			▲				▲	▲		×	
file	▲	▲						▲	▲		×	▲	▲	▲	▲	▲		▲		▲		▲	▲		▲	
hidden											×														×	
image		▲	▲	▲	▲			▲			▲	▲			▲						×	▲	▲	▲		
password		▲						▲	▲		×	▲	▲	▲	▲	▲	▲	▲		▲		▲	▲		×	
radio		▲					▲	▲			×	▲			▲			▲				▲	▲		×	
reset		▲						▲				▲			▲							▲	▲		▲	
submit		▲						▲			▲	▲			▲							▲	▲		▲	
text		▲						▲	▲		×	▲	▲	▲	▲	▲	▲	▲		▲		▲	▲		▲	
<button>		▲						▲			×		▲		▲	▲						▲			▲	
<select>								▲		▲	×		▲	▲	▲	▲				▲		▲				
<textarea>		▲				▲		▲			×		▲	▲	▲	▲	▲		▲			▲				▲

You select the type of control to include in the form with the `<input>` tag's required `type` attribute, and you name the field (used during the form submission process to the server; see earlier description) with the `name` attribute. Although the value of the `name` attribute is technically an arbitrary string, we recommend that you use a name without embedded spaces or punctuation. If you stick to just letters and numbers (but no leading digits) and represent spaces with the underscore (_) character, you'll have fewer problems. For example, "cost_in_dollars" and "overhead_percentage" are good choices for element names; "$cost" and "overhead %" might cause problems.

In addition, notice that the name you give to a form control is directly associated with the data that the u ser inputs to that control and which gets passed to the form-processing server. It is not the same as and does not share the same namespace with the `name` attribute for a hyperlink fragment or a frame document.

<input>

Function:

Create an input element within a form

Attributes:

ACCEPT	ONFOCUS
ACCESSKEY	ONKEYDOWN
ALIGN	ONKEYPRESS
ALT	ONKEYUP
BORDER N	ONMOUSEDOWN
CHECKED	ONMOUSEMOVE
CLASS	ONMOUSEOUT
DIR	ONMOUSEUP
DISABLED	ONSELECT
ID	READONLY
LANG	SIZE
MAXLENGTH	SRC
NAME	STYLE
NOTAB O	TABINDEX
ONMOUSEOVER	TABORDER O
ONBLUR	TITLE
ONCHANGE	TYPE
ONCLICK	USEMAP
ONDBLCLICK	VALUE

End tag:

None in HTML; </input> or <input ... /> with XHTML

Contains:

Nothing

Used in:

form_content

9.5.1 Text Fields in Forms

The HTML and XHTML standards let you include four types of text entry controls in your forms: a conventional text entry field, a masked field for secure data entry, a field that names a file to be transmitted as part of your form data, and a special multiline text entry `<textarea>` tag. The first three types are `<input>`-based controls; the fourth is a separate tag that we describe in Section 9.7.

9.5.1.1 Conventional text fields

The most useful as well as the most common form input control is the text entry field. A text entry field appears in the browser window as an empty box on one line and accepts a single line of user input that becomes the value of the control when the user submits the form to the server. To create a text entry field inside a form in your document, set the `type` of the `<input>` form element to `text`. Include a `name` attribute as well; it's required.

What constitutes a line of text differs among the various browsers. Fortunately, HTML and XHTML give us a way, with `size` and `maxlength` attributes, to dictate the width, in characters, of the text input display box, and how many total characters to accept from the user, respectively. The value for either attribute is an integer equal to the maximum number of characters you'll allow the user to see and type in the field. If `maxlength` exceeds `size`, then text scrolls back and forth within the text entry box. If `maxlength` is smaller than `size`, there will be extra blank space in the text entry box to make up the difference between the two attributes.

The default value for `size` is dependent upon the browser; the default value for `maxlength` is unlimited. We recommend that you set them yourself. Adjust the `size` attribute so that the text entry box does not extend beyond the right margin of a typical browser window (about 60 characters with a very short prompt). Set `maxlength` to a reasonable number of characters; for example, 2 for state abbreviations, 12 for phone numbers, and so on.

A text entry field is usually blank until the user types something into it. You may, however, specify an initial default value for the field with the `value` attribute. The user may modify the default, of course. If the user presses a form's reset button, the value of the field is reset to this default value. Section 9.5.4.2

These are all valid text entry form controls:

```
<input type=text name=comments>
<input type=text name=zipcode size=10 maxlength=10>
<input type="text" name="address" size="30" maxlength="256" />
<input type="text" name="rate" size="3" maxlength="3" value="100" />
```

The first example is HTML and creates a text entry field set to the browser's default width and maximum length. As we argued, this is not a good idea, because defaults vary widely among browsers, and your form layout is sure to look bad with some of them. Rather, fix the width and maximum number of acceptable input characters as we do in the second example: it lets the user type in up to ten characters inside an input box ten characters wide. Its value will be sent to the server with the name "zipcode" when the user submits the form.

The third example is XHTML and tells the browser to display a text input box 30 characters wide into which the user may type up to 256 characters. The browser automatically scrolls text inside the input box to expose the extra characters.

The last text input control is XHTML, too, three characters wide so that the user can type in only three characters, and sets its initial value to 100.

Notice in the second and fourth examples, it is implied that certain kinds of data are to be entered by the user - a postal code or a numeric rate, respectively. Except for limiting *how many*, neither HTML nor XHTML provide a way for you to dictate *what* characters may be typed into a text input field. For instance, in the last example field, the user may type "ABC" even though you intend it to be a number less than 1,000. Your server-side application or an applet must trap erroneous or mistaken input, as well as check for incomplete forms, and send the appropriate error message to the user when things aren't right. That can be a tedious process, so we emphasize again: provide clear and precise instructions and prompts. Make sure your forms tell users what kinds of input you expect from them, thereby reducing the number of mistakes they may make when filling it out.

9.5.1.2 Masked text controls

Like the Lone Ranger, the mask is on the good guys in a masked text field. It behaves just like a conventional text control in a form, except that the user-typed characters don't appear onscreen. Rather, the browser obscures the characters in a masked text to keep such things as passwords and other sensitive codes away from prying eyes.

To create a masked text control, set the value of the `type` attribute to `password`. All other attributes and semantics of the conventional text control apply to the masked one. Hence, you must provide a name, and you may specify a `size` and `maxlength` for the field, as well as an initial `value` (we recommend it).

Don't be misled: a masked text control is not all that secure. The typed-in value is only obscured onscreen; the browser transmits it unencrypted when the form is submitted to the server, unless you are using a special secure-forms server. So, while prying eyes may not see them onscreen, devious bad guys may steal the information electronically.

9.5.1.3 File-selection controls

As its name implies, the file-selection control lets users select a file stored on their computer and send it to the server when they submit the form. The browser presents the file-selection form control to the user like other text fields, and accompanied by a button labeled "Browse" to its right. Users either type the pathname of the file directly as text into the field or, with the Browse option, select the name of a locally stored file from a system-specific dialog box.

Create a file-selection control in a form by setting the value of the `type` attribute to `file`. Like other text controls, the `size` and `maxlength` of a file-selection field should be set to appropriate values, with the browser creating a field 20 characters wide, if not otherwise directed. Since file and directory names differ widely among systems, it makes no sense to provide a default value for this control. As such, the `value` attribute should not be used with this kind of text control.

The Browse button opens a platform-specific file-selection dialog box that allows users to select a value for the field. In this case, the entire pathname of the selected file is placed into the field, even if the length of that pathname exceeds the control's specified `maxlength`.

Use the `accept` attribute to constrain the types of files that the browser lets the user select. Its value is a comma-separated list of MIME encodings; users can select only files whose type matches one of those in the list. For example, to restrict the selection to images, you might add `accept="image/*"` to the file selection `<input>` tag.

Unlike other form input controls, the file-selection field works correctly only with a specific form data encoding and transmission method. If you include one or more file-selection fields in your form, you must set the `enctype` attribute of the `<form>` tag to `multipart/form-data` and the `<form>` tag's `method` attribute to `post`. Otherwise, the file-selection field behaves like a regular text field, transmitting its value (that is, the file's pathname) to the server instead of the contents of the file itself.

This is all easier than it may sound. For example, here is an HTML form that collects a person's name and favorite file:

```
<form enctype="multipart/form-data" method=post
    action="cgi-bin/save_file">
Your name: <input type=text size=20 name=the_name>
<p>
Your favorite file: <input type=file size=20 name=fav_file>
</form>
```

The data transmitted from the browser to the server for this example form has two parts. The first contains the value for the name field, and the second contains the name and contents of the specified file:

```
-----------------------------6099238414674
Content-Disposition: form-data; name="the_name"

One line of text field contents
-----------------------------6099238414674
Content-Disposition: form-data; name="fav_file"; filename="abc"

First line of file
...
Last line of file
-----------------------------6099238414674--
```

The browsers don't check that a valid file has been specified by the user. If no file is specified, the filename portion of the `Content-Disposition` header will be empty. If the file doesn't exist, its name appears in the filename subheader, but there will be no `Content-Type` header or subsequent lines of file content. Valid files may contain nonprintable or binary data; there is no way to restrict user-selectable file types. In light of these potential problems, the form-processing application on the server should be robust enough to handle missing files, erroneous files, extremely large files, and files with unusual or unexpected formats.

9.5.2 Checkboxes

The checkbox form control gives users a way to select or deselect an item quickly and easily in your form. Checkboxes may also be grouped to create a set of choices, any of which may be selected or deselected by the user.

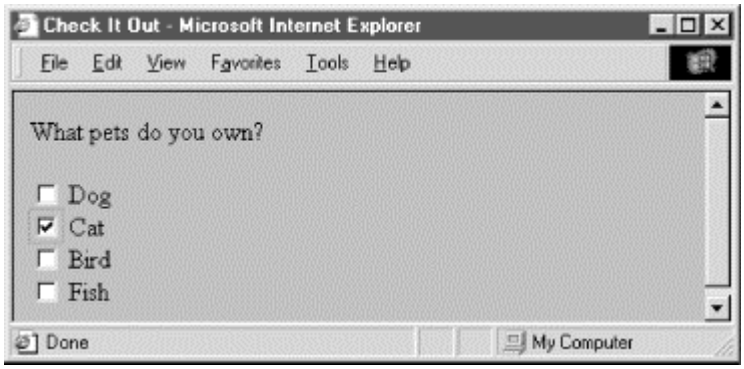
Create individual checkboxes by setting the `type` attribute for each `<input>` tag to `checkbox`. Include the required `name` and `value` attributes. If the item is selected by the user, it will contribute a value when the form is submitted. If it is not selected, that element will not contribute a value. The optional `checked` attribute (no value) tells the browser to display a checked checkbox and include the value when submitting the form to the server unless the user specifically clicks the mouse to deselect (uncheck) the box.

The popular browsers include the value of selected (checked) checkboxes with other form parameters when they are submitted to the server. The value of the checked checkbox is the text string you specify in the required `value` attribute. For example, in XHTML:

```
<form>
  What pets do you own?
  <p>
    <input type="checkbox" name="pets" value="dog" /> Dog
  <br />
    <input type="checkbox" checked="checked" name="pets" value="cat" /> Cat
  <br />
    <input type="checkbox" name="pets" value="bird" /> Bird
  <br />
    <input type="checkbox" name="pets" value="fish" /> Fish
  </p>
</form>
```

creates a checkbox group as shown in Figure 9-3.

Figure 9-3. A checkbox group



Although part of the group, each checkbox control appears as a separate choice onscreen. Notice too, with all due respect to dog, bird, and fish lovers, that we've preselected the cat checkbox with the `checked` attribute in its tag. We've also provided text labels; the similar value attributes don't appear in the browser's window, but are the values included in the form's parameter list if the checkbox is selected and the form is submitted to the server by the user. Also, you need to use paragraph or line-break tags to control the layout of your checkbox group, as you do for other form controls.

In the example, if "Cat" and "Fish" are checked when the form is submitted, the values included in the parameter list sent to the server would be:

```
pets=cat
pets=fish
```

9.5.3 Radio Buttons

Radio-button form controls are similar in behavior to checkboxes, except that only one in the group may be selected by the user.[3] Create a radio button by setting the `type` attribute of the `<input>` tag to `radio`. Like with checkbox controls, radio buttons each require a `name` and `value` attribute. Radio buttons with the same name are members of a group. One of them may be initially checked by including the `checked` attribute with that element. If no element in the group is checked, the browser automatically checks the first element in the group.

> [3] Some of us are old enough, while not yet senile, to recall when automobile radios had mechanical pushbuttons for selecting a station. Pushing in one button popped out the previously depressed one, implementing a mechanical one-of-many choice mechanism.

You should give each radio button element a different value, so that the form-processing server can sort them out after submission of the form.

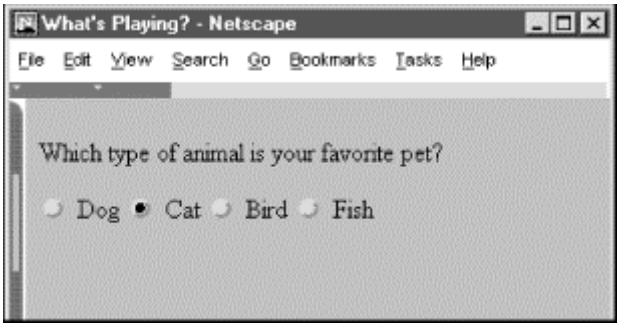
Here's the previous example reworked in HTML so you get to choose only one animal as a favorite pet (see Figure 9-4):

```
<form>
  Which type of animal is your favorite pet?
  <p>
    <input type=radio name=favorite value="dog"> Dog
    <input type=radio checked name=favorite value="cat"> Cat
    <input type=radio name=favorite value="bird"> Bird
    <input type=radio name=favorite value="fish"> Fish
</form>
```

Like the previous example with checkboxes, we've tipped our hat toward felines, making the "Cat" radio button the default choice. If you select an alternative - "Bird," for instance - the browser automatically deselects "Cat." When the user submits the form to the server, the browser includes only one value with the name "favorite" in the list of form parameters; `favorite=bird`, if that was your choice.

One of the controls in a group of radio buttons is always selected, so it makes no sense to create a single radio button; they should appear in your documents as groups of two or more. (Use checkboxes for on/off, yes/no types of form controls.)

Figure 9-4. Radio buttons allow only one selection per group



9.5.4 Action Buttons

Although the terminology is potentially confusing, there is another class of buttons for forms. Unlike the radio buttons and checkboxes described previously, these special types of form controls act immediately, their effects cannot be reversed, and they affect the entire contents of the form, not just the value of a single field. These "action" buttons (for lack of a better term) include submit, reset, regular, and clickable image buttons. When selected by the user, both the submit and image buttons cause the browser to submit all of the form's parameters to the form-processing server. A regular button does not submit the form, but can be used to invoke an applet to manipulate or validate the form. The reset button acts locally to return a partially filled-out form to its original (default) state. Section 12.3.3

In this section we describe the action buttons that you may create with the standard form `<input>` element. In the next section, we describe in detail the newer `<button>` tag that achieves identical effects and allows you greater control over the presentation and display of your form buttons.

9.5.4.1 Submission buttons

The submit button (`<input type=submit>`) does what its name implies, setting in motion the form's submission to the server from the browser. You may have more than one submit button in a form. You may also include `name` and `value` attributes with the submit type of input form button.

With the simplest submit button (that without a `name` or `value` attribute), the browser displays a small rectangle or oval with the default label "Submit." Otherwise, the browser labels the button with the text you include with the tag's `value` attribute. If you provide a `name` attribute, the `value` attribute for the submit button is added to the parameter list the browser sends along to the server. That's good, because it gives you a way to identify which button in a form was pressed, letting you process any one of several different forms with a single form processing application.

The following are all valid submission buttons:

```
<input type=submit>
<input type=submit value="Order Kumquats">
<input type="submit" value="Ship Overnight" name="ship_style" />
```

The first one is in HTML and is also the simplest: the browser displays a button, labeled "Submit," which activates the form-processing sequence when clicked by the user. It does not add an element to the parameter list that the browser passes to the form processing server and application.

The second example HTML button has the `value` attribute that makes the displayed button label "Order Kumquats," but like the first example does not include the button's value in the form's parameter list.

The last example, in XHTML, sets the button label and makes it part of the form's parameter list. When clicked by the user, the last example of the submission button adds the parameter `ship_style="Ship Overnight"` to the form's parameter list.

9.5.4.2 Reset buttons

The reset type of form <input> button is nearly self-explanatory: it lets the user reset - erase or set to some default value - all elements in the form. Unlike the other buttons, a reset button does not initiate form processing. Instead, the browser does the work of resetting the form elements. The server never knows (or cares, for that matter) if or when the user might have pressed a reset button.

By default, the browser displays a reset button with the label "Reset." You can change that by specifying a value attribute with your own button label.

Here are two sample reset buttons:

```
<input type=reset>
<input type="reset" value="Use Defaults" />
```

The first one creates a reset button labeled "Reset"; the browser labels the second example reset button with "Use Defaults." They both initiate the same reset response in the browser.

9.5.4.3 Custom image buttons

The image type of <input> form element creates a custom button that is a "clickable" image. It's a special button made out of your specified image that, when clicked by the user, tells the browser to submit the form to the server and includes the x,y coordinates of the mouse pointer in the form's parameter list, much like the mouse-sensitive image maps we discuss in Chapter 6. Image buttons require a src attribute with the URL of the image file, and you can include a name attribute and a descriptive alt attribute for non-graphical browsers. Although deprecated in HTML 4, you may also use align to control alignment of the image within the current line of text. Use the border attribute to control the width, if any, of the frame Netscape puts around the form image much like the border attribute for the <img> tag (Internet Explorer doesn't place a border around form <input> images).

Here are a couple of valid image buttons:

```
<input type="image" src="pics/map.gif" name="map" />
<input type=image src="pics/xmap.gif" align=top name=map>
```

The browser displays the designated image within the form's content flow. The second button's image will be aligned with the top of the adjacent text, as specified by the align attribute. Netscape Navigator adds a border, as it does when an image is part of an anchor (<a> tag), to signal that the image is a form button.

When the user clicks the image, the browser sends the horizontal offset, in pixels, of the mouse from the left edge of the image and the vertical offset from the top edge of the image to the server. These values are assigned the name of the image as specified with the name attribute, followed by .x and .y, respectively. Thus, if someone clicked the image specified earlier in the example, the browser would send parameters named map.x and map.y to the server.

Image buttons behave much like mouse-sensitive image maps (usemaps), and, like the programs or client-side <map> tags that process image maps, your form-processor may use the x,y mouse-pointer parameters to choose a special course of action. You should use an image button when you need additional form information to process the user's request. If an image map of links is all you need, use a mouse-sensitive image map. Mouse-sensitive images also have the added benefit of providing server-side support for automatic detection of shape selection within the image, letting you deal with the image as a selectable collection of shapes. Buttons with images require you to write code that determines where the user clicked on the image and how this position can be translated to an appropriate action by the server.

Oddly, the HTML 4 and XHTML standards allow the use of the usemap attribute with an image button, but do not explain how such a use might conflict with normal server processing of the x,y coordinates of the mouse position. We recommend not mixing the two, using mouse-sensitive images outside of forms and image buttons within forms.

9.5.4.4 Push buttons

Using the <input type=button> tag (or the <button> tag, described in Section 9.6), you can create a button that can be clicked by the user but that does not submit or reset the form. The value attribute can be used to set the label on the button; the name attribute, if specified, will cause the supplied value to be passed to the form processing script.

You might wonder what value such buttons provide: little or none, unless you supply one or more of the event attributes along with a snippet of JavaScript to be executed when the user interacts with the button. Thus empowered, regular buttons can be used to validate form contents, update fields, manipulate the document, and initiate all sorts of client-side activity.Section 12.3.3

9.5.4.5 Multiple buttons in a single form

You can have several buttons of the same or different types in a single form. Even simple forms have both reset and submit buttons, for example. To distinguish between them, make sure each has a different `value` attribute, which the browser uses for the button label. Depending on the way you program the form-processing application, you might make the `name` of each button different, but it is usually easier to name all similarly acting buttons the same and let the button handling subroutine sort them out by value. For instance (all in HTML):

```
<input type=submit name=action value="Add">
<input type=submit name=action value="Delete">
<input type=submit name=action value="Change">
<input type=submit name=action value="Cancel">
```

When the user selects one of these example buttons, a form parameter named `action` will be sent to the server. The value of this parameter will be one of the button names. The server-side application gets the value and behaves accordingly.

Since an image button doesn't have a `value` attribute, the only way to distinguish between several image buttons on a single form is to ensure that they all have different names.

9.5.5 Hidden Fields

The last type of form input control we describe in this chapter is hidden from view. No, we're not trying to conceal anything. It's a way to embed information into your forms that cannot be ignored or altered by the browser or user. Rather, the `<input type=hidden>` tag's required `name` and `value` attributes automatically get included in the submitted form's parameter list. These serve to label the form and can be invaluable when sorting out different forms or form versions from a collection of submitted and saved forms.

Another use for hidden fields is to manage user/server interactions. For instance, it helps the server to know that the current form has come from a person who made a similar request a few moments ago. Normally, the server does not retain this information and each transaction between the server and client is completely independent from all other transactions.

For example, the first form submitted by the user might have asked for some basic information, such as the user's name and where they live. Based on that initial contact, the server might create a second form asking more specific questions of the user. Since it is tedious for users to re-enter the same basic information from the first form, the server can be programmed to put those values in the second form in hidden fields. When the second form comes back, all the important information from both forms is there, and the second form can be matched to the first one, if necessary.

Hidden fields may also direct the server towards some specific action. For example, you might embed the hidden field:

```
<input type=hidden name=action value=change>
```

Therefore, if you have one server-side application that handles the processing of several forms, each form might contain a different action code to help that server application sort things out.

9.6 The <button> Tag

As we described earlier, you create an action button with traditional HTML or XHTML by including its type value in the standard `<input>` tag. For instance, the `<input type=submit>` form control creates a button that, when selected by the user, tells the browser to send the form's contents to the processing server or to an email address (mailto option). Display-wise, you don't have any direct control over what that submit button looks like, beyond changing the default label "Submit" to some other word or short phrase like "Hit me" or "Outta here!"

First introduced in the HTML 4.0 standard, the `<button>` tag acts the same as the `<input>` button, but gives you more control over how the element gets displayed by the browser. In particular, all of the attributes you might use with the `<input type=button>` element are acceptable with the `<button>` tag.

<button>

Function:

Create a button element within a form

Attributes:

ACCESSKEY	ONKEYDOWN
CLASS	ONKEYPRESS
DIR	ONKEYUP
DISABLED	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
NAME	ONMOUSEUP
NOTAB	STYLE
ONMOUSEOVER	TABINDEX
ONBLUR	TABORDER
ONCLICK	TITLE
ONDBLCLICK	TYPE
ONFOCUS	VALUE

End tag:

</button>; never omitted

Contains:

button_content

Used in:

form_content

9.6.1 The <button> button

Neither the HTML 4 nor the XHTML standard is overly clear as to what display enhancements to a form button control the `<button>` element should provide, other than to suggest that the contents should be 3D and visually appear to react like a push button when selected by the user; that is, go in and back out when pressed. Internet Explorer Version 5 and Netscape Version 6 support `<button>`.

The `<button>` control does provide for a greater variety and richer contents over its `<input>` analogues. Everything between the `<button>` and `</button>` tags becomes the content of the button, including any acceptable body content, such as text or multimedia. For instance, you could include an image and related text within a button, creating attractive labelled icons in your buttons. The only *verboten* element is an image map, since its mouse- and keyboard-sensitive actions interfere with the form button.

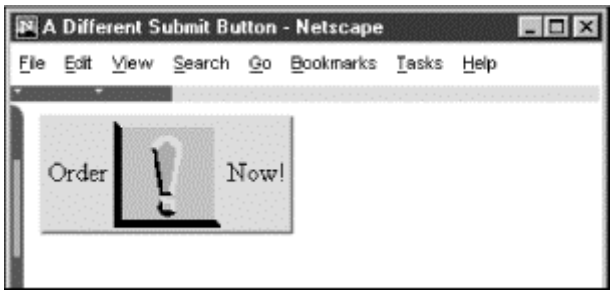
9.6.2 The type Attribute

Use the `type` attribute for the `<button>` tag to define the button's action. Its value should be set to `submit`, `reset`, or `button`. Like its `<input>` analog, a `<button type=submit>`, when selected by the user, tells the browser to package and send the contents of the form to the form-processing server or email it to the mailto recipient. Using `type=reset` creates a conventional reset button, and using `type=button` creates a conventional push button.

For example, Figure 9-5 shows how Netscape Navigator renders the following *exclaim.gif* icon inset on a 3D button that pushes in and pops back out when the user clicks it with the mouse. In doing so, the browser submits the form to the server:

```
<button type=submit>
Order <img src="icons/exclaim.gif" align=middle alt="Order Now"> Now!
</button>
```

Figure 9-5. A form-submit <button>



Notice that you can exploit the rich set of `<img>` tag attributes, including `align` and `alt`, for this `<button>` style of form control.

Since the `<button>` tag is so similar to the `<input type=button>` element, why have it at all? The only reason is to provide far richer content for buttons. If your buttons are conventional text buttons, the `<input>` tag will suffice. If you want to create fancy mixed-content buttons, you'll need to use the `<button>` tag.

9.7 Multiline Text Areas

The conventional and hidden-text types for forms restrict user input to a single line of characters. The `<textarea>` form tag sets users free.

9.7.1 The <textarea> Tag

As part of a form, the `<textarea>` tag creates a multiline text entry area in the user's browser display. In it, the user may type a nearly unlimited number of lines of text. Upon submission of the form, the browser collects all the lines of text, each separated by `"%0D%0A"` (carriage return/line feed), and sends them to the server as the value of this form element, using the name specified by the required `name` attribute.

You may include plain text inside the `<textarea>` tag and its end tag. That default text must be plain text - no tags or other special elements. The contents may be modified by the user, and the browser uses that text as the default value if the user presses a reset button for the form. Hence, the text content is most often included for instructions and examples:

```
Tell us about yourself:
<textarea name=address cols=40 rows=4>
  Your Name Here
  1234 My Street
  Anytown, State Zipcode
</textarea>
```

9.7.1.1 The rows and cols attributes

A multiline text input area stands alone onscreen: body content flows above and below, but not around it. You can control its dimensions, however, by defining the `cols` and `rows` attributes for the visible rectangular area set aside by the browser for multiline input. We suggest you do set these attributes. The common browsers have a habit of setting aside the smallest, least readable region possible for `<textarea>` input, and the user can't resize it. Both attributes require integer values for the respective dimension's size in characters. The browser automatically scrolls text that exceeds either dimension.

<textarea>

Function:

Create a multiline text input area

Attributes:

ACCESSKEY	ONKEYPRESS
CLASS	ONKEYUP
COLS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
DISABLED	ONMOUSEOUT
ID	ONMOUSEOVER
LANG	ONMOUSEUP
NAME	ONSELECT
ONBLUR	READONLY
ONCHANGE	ROWS
ONCLICK	STYLE
ONDBLCLICK	TABINDEX
ONFOCUS	TITLE
ONKEYDOWN	WRAP

End tag:

</textarea>; never omitted

Contains:

plain_text

Used in:

form_content

9.7.1.2 The wrap attribute

Normally, text typed in the text area by the user is transmitted to the server exactly as typed, with lines broken only where the user pressed the Enter key. Since this is often not the action desired by the user, you can enable word wrapping within the text area. When the user types a line that is longer than the width of the text area, the browser automatically moves the extra text down to the next line, breaking the line at the nearest point between words in the line.

With the wrap attribute set to virtual, the text is wrapped within the text area for presentation to the user, but the text is transmitted to the server as if no wrapping had occurred, except where the user pressed the Enter key.

With the wrap attribute set to physical, the text is wrapped within the text area and is transmitted to the server as if the user had actually typed it that way. This the most useful way to use word wrap, since the text is transmitted exactly as the user sees it in the text area.

To obtain the default action, set the wrap attribute to off.

As an example, consider the following 60 characters of text that are being typed into a 40-character-wide text area:

```
Word wrapping is a feature that makes life easier for users.
```

With wrap=off, the text area will contain one line and the user will have to scroll to the right to see all of the text. One line of text will be transmitted to the server.

With wrap=virtual, the text area will contain two lines of text, broken after the word "makes." Only one line of text will be transmitted to the server: the entire line with no embedded newline characters.

With wrap=physical, the text area will contain two lines of text, broken after the word "makes." Two lines of text will be sent to the server, separated by a newline character after the word "makes."

9.8 Multiple Choice Elements

Checkboxes and radio buttons give you powerful means for creating multiple-choice questions and answers, but they can lead to long forms that are tedious to write and put a fair amount of clutter onscreen. The <select> tag gives you two compact alternatives: pull-down menus and scrolling lists.

9.8.1 The <select> Tag

By placing a list of <option>-tagged items inside the <select> tag of a form, you magically create a pull-down menu of choices.

As with other form tags, the name attribute is required and used by the browser when submitting the <select> choices to the server. Unlike radio buttons, no item is preselected, so if none is selected, no values are sent to the server when the form is submitted. Otherwise, the browser submits the selected item or collects multiple selections, separated with commas, into a single parameter list and includes the name attribute when submitting <select> form data to the server.

9.8.1.1 The multiple attribute

To allow more than one option selection at a time, add the multiple attribute to the <select> tag. This causes the <select> element to behave like an <input type=checkbox> element. If multiple is not specified, exactly one option can be selected at a time, just like a group of radio buttons.

9.8.1.2 The size attribute

The size attribute determines how many options are visible to the user at one time. The value of size should be a positive integer. The default value is 1 when size isn't specified. At size=1, if multiple is not specified, the browser typically displays the <select> list as a pop-up menu. Size values greater than one or specification of the multiple attribute cause the <select> to be displayed as a scrolling list.

In the following XHTML example, we've converted our previous checkbox example into a scrolling, multiple-choice menu. Notice also that the size attribute tells the browser to display three options at a time:[4]

[4] Notice the </option> end tags. They are not usually included in standard HTML documents, but must appear in XHTML.

```
What pets do you have?
  <select name="pets" size="3" multiple="multiple">
    <option>Dog</option>
    <option>Cat</option>
    <option>Bird</option>
    <option>Fish</option>
  </select>
```

The result is shown in Figure 9-6.

<select>

Function:

Create single- and multiple-choice menus

Attributes:

CLASS	ONKEYPRESS
DIR	ONKEYUP
DISABLED	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
MULTIPLE	ONMOUSEOVER
NAME	ONMOUSEUP
ONBLUR	SIZE
ONCHANGE	STYLE
ONCLICK	TABINDEX
ONDBLCLICK	TITLE
ONFOCUS	
ONKEYDOWN	

End tag:

</select>; never omitted

Contains:

select_content

Used in:

form_content

Figure 9-6. A <select> element, formatted with size=3

9.8.2 The <option> Tag

Use the `<option>` tag to define each item within a `<select>` form control.

<option>

Function:

Define available options within a <select> menu

Attributes:

CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
DISABLED	ONMOUSEOUT
ID	ONMOUSEOVER
LABEL	ONMOUSEUP
LANG	SELECTED
ONCLICK	STYLE
ONDBLCLICK	TITLE
ONKEYDOWN	VALUE
ONKEYPRESS	
ONKEYUP	

End tag:

</option>; usually omitted in HTML

Contains:

plain_text

Used in:

select_content

The browser displays the `<option>` tag's contents as an element within the `<select>` tag's menu or scrolling list, so the content must be plain text only, without any other sort of markup.

9.8.2.1 The value attribute

Use the `value` attribute to set a value for each option the browser sends to the server if that option is selected by the user. If the `value` attribute has not been specified, the value of the option is set to the content of the `<option>` tag. As an example, consider these HTML options:

```
<option value=Dog>Dog
<option>Dog
```

Both have the same value. The first is explicitly set within the `<option>` tag; the second defaults to the content of the `<option>` tag itself.

9.8.2.2 The selected attribute

By default, all options within a multiple-choice `<select>` tag are unselected. Include the `selected` attribute (no value) inside the `<option>` tag to preselect one or more options, which the user may then deselect. Single-choice `<select>` tags will preselect the first option if no option is explicitly preselected.

9.8.2.3 The label attribute

Normally, the contents of the `<option>` tag are used to create the label for that element when it is displayed to the user. If the `label` attribute is supplied, its value is used as a label instead.

9.8.3 The <optgroup> Tag

Menus of choices in forms can be quite large, making them difficult to display and use. In these cases, it is helpful to group related choices, which can then be presented as a set of nested, cascading menus to the user. New in HTML 4.0, the `<optgroup>` tag brings this capability to HTML and XHTML forms, albeit limited.

<optgroup>

Function:

Group related <option> elements within a <select> menu

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
DISABLED	ONMOUSEMOVE
ID	ONMOUSEOUT
LABEL	ONMOUSEOVER
LANG	ONMOUSEUP
ONCLICK	STYLE
ONDBLCLICK	TITLE
ONKEYDOWN	
ONKEYPRESS	

End tag:

</optgroup>; may be omitted in HTML

Contains:

optgroup_contents

Used in:

select_content

Use the <optgroup> tag only within a <select> tag, and it may only contain <option> tags. The browser creates submenus for each <optgroup> tag within the main <select> menu. For example, with HTML you might use <optgroup> to present a form menu of states organized by region:

```
<select name=state>
   <optgroup label=Northeast>
      <option>Maine
      <option>New Hampshire
      ...
   </optgroup>
   <optgroup label=South>
      <option>Georgia
      <option>Florida
      ...
   </optgroup>
   ...
</select>
```

Since no browser yet fully supports the <optgroup> tag (the popular browsers simply display them as a scrolling menu), we can't show you what this menu might look like. However, it will probably look and feel much like the familiar pull-down menus that are a common feature of most graphical user interfaces. When selected with the mouse or keyboard, the optgroup opens into one or more menus. For instance, our "state" example probably will have submenus labeled "Northeast," "South," and so on, each of which can be pulled open to reveal a list of included states.

Regrettably, the biggest drawback to the <optgroup> tag is that it cannot be nested, limiting you to one level of submenus. Presumably this restriction will be lifted in a future version of XHTML.

9.8.3.1 The label attribute

Use the label attribute to define an optgroup's submenu title to the user. You should keep the label short and to the point to ensure that the menu can be displayed easily on a large variety of displays.

9.9 General Form Control Attributes

The many form control tags contain common attributes that, like most other tags, generally serve to label, set up the display, extend the text language, and make the tag extensible programmatically.

9.9.1 The id and title Attributes

The id attribute, like most other standard tags, lets you attach a unique string label to the form control and its contents for reference by programs (applets) and hyperlinks. This name is distinct from the name assigned to a control element with the name attribute. Names assigned with the id attribute are not passed to the server when the form is processed.

The title attribute is similar to id in that it uses a quote-enclosed string value to label the form control. However, it entitles only the form segment; its value cannot be used in an applet reference or hyperlink. Browsers may use the title as pop-up help for the user or in non-visual presentation of the form. Section 4.1.1.4 / Section 4.1.1.5

9.9.2 The event Attributes

Like most other elements, most of the form controls support a number of user mouse and keyboard event-related attributes that the HTML 4/XHTML-compliant browser will recognize and let you specially process with an applet, such as a JavaScript program. We describe the majority of these events in detail in Chapter 12.

9.9.3 The style, class, lang, and dir Attributes

The style attribute for the various form controls creates an inline style for the elements enclosed by the tag, overriding any other style rule in effect. The class attribute lets you format the content according to a predefined class of the <form> tag; its value is the name of that class. Section 8.1.1 / Section 8.3

The lang attribute specifies the language used within a control, accepting as its value any of the ISO standard two-character language abbreviations, including an optional language modifier. For example, adding lang=en-UK tells the browser that the list is in English ("en") as spoken and written in the United Kingdom (UK). Presumably, the browser may make layout or typographic decisions based upon your language choice. Section 3.6.1.2

Similarly, the dir attribute tells the browser which direction to display the control contents, from left to right (dir=ltr) like English or French, or from right to left (dir=rtl), such as with Hebrew or Chinese. Section 3.6.1.1

The dir and lang attributes are supported by the popular browsers, even though there are no behaviors defined for any specific language.

9.9.4 The tabindex, taborder, and notab Attributes

By default, all elements (except hidden elements) are part of the form's tab order. As the user presses the Tab key, the browser shifts the input focus from element to element in the form. For most browsers, the tabbing order of the elements matches the order of the elements within the <form> tag. With the tabindex attribute, you can change the order and the position of those elements within the tab order.

To reposition an element within the tab order, set the value of the attribute to the element's desired position in the tab order, with the first element in the order being number one. If you really want to change a form's tab order, we suggest you include the tabindex attribute with every element in the form, with an appropriate value for each element. In this way, you'll be sure to place every element explicitly in the tab order, and there will be no surprises when the user tabs through the form.

The value of the tabindex attribute is a positive integer indicating the position of the tagged contents in the overall tab sequence for the document. The tabbing order begins with elements with explicit tabindex values, starting from the lowest to the highest numbers. Same-valued tags get tab-selected in the order in which they appear in the document. All other table tags, such as the various form controls and hyperlinks, are the last to get tabbed, in the order in which they appear in the document. To exclude an element from the tab order, set the value of tabindex to zero. The element will be skipped when the user tabs around the form.

Internet Explorer introduced the concept of tab-order management with its proprietary taborder and notab attributes. The taborder attribute functions exactly like the tabindex attribute, while notab is equivalent to tabindex=0. Internet Explorer Version 5 now supports tabindex, as does Netscape Navigator. In general, we suggest that you use the tabindex attribute and not taborder.

9.9.5 The accesskey Attribute

Many user interfaces promote the idea of shortcut keys, short sequences of keystrokes that give you quick access to an element in the user interface. HTML 4 and XHTML provide support for this capability with the accesskey attribute. The value of the accesskey attribute is a single character that, when pressed in conjunction with some other special key, causes focus to shift immediately to the associated form element. This special key varies with each user interface: Windows users press the Alt key while Unix users press Meta.

For example, adding accesskey="T" to a <textarea> element would cause focus to shift to that text area when a Windows user pressed Alt-T. Note that the value of the accesskey attribute is a single character and is case-sensitive (capital "T" is not the same as its lower case cousin, for instance).

Currently, Internet Explorer Version 5 is the only browser that supports the accesskey attribute.

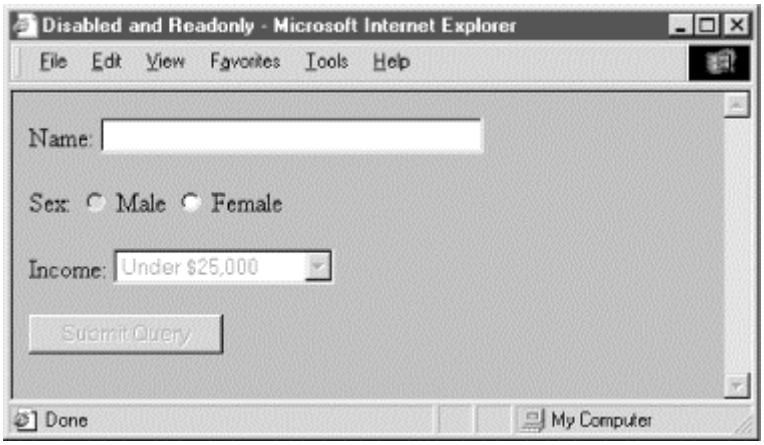
9.9.6 The disabled and readonly Attributes

The HTML 4 and XHTML standards let you define but otherwise disable a form control by simply inserting the disabled attribute within the tag. A disabled form control appears in the display, but cannot be accessed via the Tab key or otherwise selected with the mouse, nor are its parameters passed to the server when the form gets submitted by the user.

Browsers may change the appearance of disabled elements and may similarly alter any labels associated with disabled elements. Internet Explorer Version 5, for example, greys out disabled radio and submit buttons but does not change the appearance of disabled text input or menu select lists, as in Figure 9-7. You can just operate the menus or type anything into the text box:

```
<form>
  Name:
    <input type=text name=name size=32 maxlength=80 readonly>
  <p>
  Sex:
    <input type=radio name=sex value="M" disabled> Male
    <input type=radio name=sex value="F" accesskey="f"> Female
  <p>
  Income:
    <select name=income size=1 disabled>
      <option>Under $25,000
      <option>$25,001 to $50,000
      <option>$50,001 and higher
    </select>
            <p>
  <input type=submit disabled>
</form>
```

Figure 9-7. Internet Explorer Version 5 greys out some disabled form controls



Similarly, a text-related `<input>` or `<textarea>` form control that specifies the `readonly` attribute may not be altered by the user. These elements are still part of the tab order, and may be selected with the mouse, and the value of the control gets sent to the server when the user submits the form. The user can't just alter the value. So, in a sense, a form control rendered `readonly` is the visible analog of the `<input type=hidden>` control.

9.10 Labeling and Grouping Form Elements

The common text and other content you may use to label and otherwise explain a form are static. Other than by their visual relationship to the form's input areas, these labels and instructions otherwise are unassociated with the form control that they serve. Because of this, forms are not easily understood and navigable, particularly by people with impaired vision. Try it. Get a simple personal information form on screen, close your eyes, and find the place to enter your name.

The HTML 4.0 standard introduced three new tags that make navigation of forms easier for users, particularly those with disabilities. They include a way to group and caption regions of the form and a way to individually label form controls. All are supposed to get special treatment by the browser, such as being rendered by a speech-synthesizer as well as specially displayed, and can be easily accessed from the user keyboard. That is, when browsers become HTML 4/XHTML-compliant.

9.10.1 The <label> Tag

Use the `<label>` tag to define relationships between a form control, such as a text input field, and one or more text labels. According to the latest standards, the text in a label is to receive special treatment by the browser. Browsers may choose a special display style for the label (you can, too, with style sheets). And when selected by the user, the browser automatically transfers focus to a label's associated form control.

9.10.1.1 Implicit and explicit associations

One or more labels get associated with a form control in one of two ways: implicitly by including the form control as contents of the label tag, or explicitly by naming the ID of the target form control in the `<label>` tag's `for` attribute.

For example, in XHTML:

```
<label for="SSN">Social Security Number:</label>
<input type="text" name="SocSecNum" id="SSN" />
<label>Date of birth: <input type="text" name="DofB" /></label>
```

The first label explicitly relates the text "Social Security Number:" with the form's Social Security Number text-input control (SocSecNum) since its for attribute's value is identical to the control's id, "SSN." The second label ("Date of birth") does not require a `for` attribute, nor does its related control require an `id` attribute, since they are implicitly joined by placing the `<input>` tag within the `<label>` tag.

Be careful not to confuse the `name` and `id` attributes. The former creates a name for an element that is used by the server-side form processor; the latter creates a name that can be used by `<label>` tags and URLs. Note also that although a label may reference only a single form control, a single control may be referenced by several labels. This gives you the ability to steer users to a particular form input region from several places in a document.

<label>

Function:

Creates a label for a form element

Attributes:

ACCESSKEY	ONKEYDOWN
CLASS	ONKEYPRESS
DIR	ONKEYUP
FOR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONBLUR	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONFOCUS	TITLE

End tag:

</label>; never omitted

Contains:

label_contents

Used in:

form_content

9.10.1.2 Other label attributes

Labels also share many of the general display, access, and event-related tag attributes described in Section 9.9. In addition to the standard HTML 4 and XHTML event attributes, labels also support the `onfocus` and `onblur` attributes.

9.10.2 Forming a Group

Beyond individual labels, you may group a set of form controls and label the group with the `<fieldset>` and `<legend>` tags. Again, HTML 4 and XHTML standards attempt to make forms more readily accessible by users, particularly those with disabilities. Grouping form controls into explicit sections gives you the opportunity to specially display and otherwise manage the form contents.

9.10.2.1 The <fieldset> tag

The `<fieldset>` tag encapsulates a section of form contents, creating a group of related form fields. `<fieldset>` doesn't have any required or unique attributes.

When a group of form elements are placed within a `<fieldset>` tag, the browser may display them in a special manner. This might include a special border, 3D effects, or even creating a subform to handle the elements.

<fieldset>

Function:

Group related elements within a form

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

</fieldset>; never omitted

Contains:

form_content

Used in:

form_content

9.10.2.2 The <legend> tag

Use the `<legend>` tag to create a label for a fieldset in a form. The tag may appear only inside a `<fieldset>`. Like `<label>`, the `<legend>` contents are to be specially treated by the HTML 4/XHTML-compliant browser, transferring focus to associated form elements when selected and serving to improve accessibility of users to a `<fieldset>`.

In addition to supporting many of the form element attributes described in Section 9.9, the `<legend>` tag accepts the `accesskey` attribute and the `align` attribute. The value of `align` may be `top`, `bottom`, `left`, or `right`, instructing the browser where the legend should be placed with respect to the field set.

Bringing all these tags together, here is a field set and legend containing a few form elements, individually labelled. Notice in Figure 9-8 how Internet Explorer Version 5 neatly puts a frame around the fieldset and through the legend, but doesn't otherwise format the fieldset contents:

```
<fieldset>
   <legend>Personal information</legend>
   <label>Name:<input type="text" /></label>
   <label>Address:<input type="text" /></label>
   <label>Phone:<input type="text" /></label>
</fieldset>
```

<legend>

Function:

Create a legend for a field set within a form

Attributes:

ACCESSKEY	ONKEYPRESS
ALIGN	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE

End tag:

</legend>; may be omitted in HTML

Contains:

legend_content

Used in:

form_content

Figure 9-8. Internet Explorer Version 5 puts a frame around form fieldsets

9.11 Creating Effective Forms

Properly done, a form can provide an effective user interface for your readers. With some server-side programming tricks, you can use forms to personalize the documents that you present to readers and thereby significantly increase the value of your pages on the Web.

9.11.1 Browser Constraints

Unlike other graphical user interfaces, browser displays are static. They have little or no capability for real-time data validation, for example, or to update the values in a form based upon user input, giving users no help or guidance.[5] Hence, poorly designed web forms can be difficult to fill out.

> [5] This is not entirely true. While neither HTML nor XHTML provide for data validation and user guidance, it is possible to attach Java or JavaScript applets to your form elements that do a very nice job of validating form data, updating form fields based upon user input, and guiding users through your forms.

Make sure your forms assist users as much as possible in getting their input correct. Adjust the size of text input fields to give clues on acceptable input: five-character (or the new nine-character) zip code fields, for instance. Use checkboxes, radio buttons, and selection lists whenever possible to narrow the list of choices the user must make.

Make sure you also adequately document your forms. Explain how to fill them out, supplying examples for each field. Provide appropriate hyperlinks to documentation that describes each field, if necessary.

When the form is submitted, make sure that the server-side application exhaustively validates the user's data. If an error is discovered, present the user with intelligent error messages and possible corrections. One of the most frustrating aspects of filling out forms is having to start over from scratch whenever the server discovers an error. To alleviate this ugly redundancy and burden on your readers, consider spending extra time and resources on the server side that returns the user's completed form with the erroneous fields flagged for changes.

While these suggestions require significant effort on your part, they will pay off many times over by making life easier for your users. Remember, you'll create the form just once, but it may be used thousands or even millions of times by users.

9.11.2 Handling Limited Displays

Although most PCs have been upgraded to provide resolution significantly better than the 600 x 480 that was common when we wrote the first edition of this book, many devices (laptops with poor screens, WebTV, cell phones with built-in browsers) dictate that form design should be conservative. The best compromise is to assume a document-viewing window roughly 75 readable characters wide and 30 to 50 lines tall. You should design your forms (and all your documents) so that they are effective when viewed through a window of this size.

You should structure your form to scroll naturally into two or three logical sections. The user can fill out the first section, page down; fill out the second section, page down; and so forth.

You should also avoid wide input elements. It is difficult enough to deal with a scrolling-text field or text area without having to scroll the document itself horizontally to see additional portions of the input element.

9.11.3 User Interface Considerations

When you elect to create a form, you immediately assume another role: that of a user-interface designer. While a complete discussion of user interface design is beyond the scope of this book, it helps to understand a few basic design rules to create effective, attractive forms.

Any user interface is perceived at several levels simultaneously. Forms are no different. At the lowest level, your brain recognizes shapes within the document, attempting to categorize the elements of the form. At a higher level, you are reading the text guides and prompts, trying to determine what input is required of you. At the highest level, you are seeking to accomplish a goal with the interface as your tool.

A good form accommodates all three of these perceptive needs. Input elements should be organized in logical groups so that your brain can process the form layout in chunks of related fields. Consistent, well-written prompts and supporting text assist and lead the user to enter the correct information. Text prompts also remind users of the task at hand and reinforce the form's goal.

9.11.4 Creating Forms That Flow

Users process forms in a predictable order, one element after another, seeking to find the next element as they finish the previous one. To accommodate this searching process, you should design your forms so that one field leads naturally to another, and related fields are grouped together. Similarly, groups should lead naturally to one another and should be formatted in a consistent manner.

Simply stringing a number of fields together does not constitute an effective form. You must put yourself in the place of your users, who are using the form for the first time. Test your form on unsuspecting friends and colleagues before you release it on the general public. Is it easy to determine the purpose of the form? Where do you start filling things out? Can the user find a button to push to submit the form? Is there an opportunity to confirm decisions? Do readers understand what is expected of them for each field?

Your forms should lead the user naturally through the process of supplying the necessary data for the application. You wouldn't ask for a street address before asking for the user's name; other rules may dictate the ordering of other groups of input elements. To see if your form really works, make sure you view it on several browsers and have several people fill it out and comment on its effectiveness.

9.11.5 Good Form, Old Chap

At first glance, the basic rule of HTML and XHTML—content, not style—seems in direct opposition to the basic rule of good interface design—precise, consistent layout. Even so, it is possible to use some elements to greatly improve the layout and readability of most forms.

Traditional page layout uses a grid of columns to align common elements within a page. The resulting implied vertical and horizontal "edges" of adjacent elements give a sense of order and organization to the page and make it easy for the eye to scan and follow.

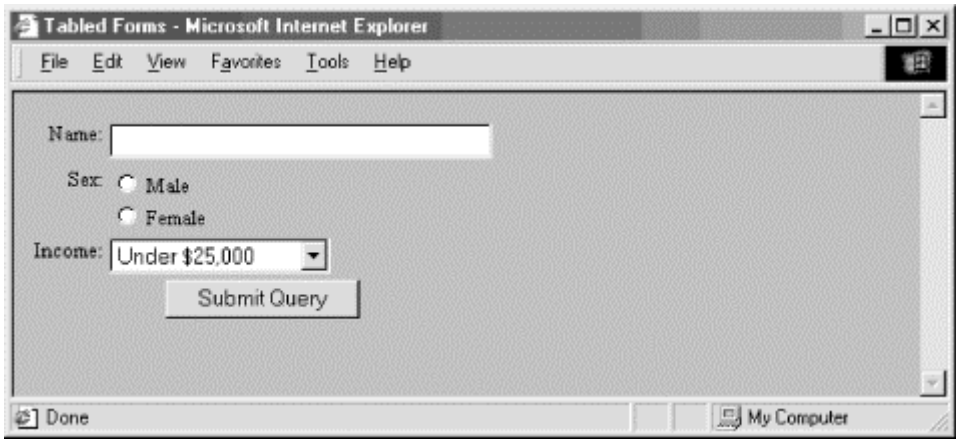
HTML and XHTML make it hard, but you can accomplish the same sort of layout for your forms. For example, you can group related elements and separate groups with empty paragraphs or horizontal rules.

Vertical alignment is more difficult, but not impossible. In general, forms are easier to use if you arrange the input elements vertically and aligned to a common margin. One popular form layout keeps the left edge of the input elements aligned, with the element labels immediately to the left of the elements. This is done by using tables to place and align each form element and its label. Here is our previous HTML form example, with the labels placed in the first column and the corresponding elements in the second:

```
<form method=POST action="http://www.kumquat.com/demo">
  <table border=0>
    <tr valign=top>
      <td align=right>Name:</td>
      <td align=left><input type=text name=name size=32 maxlength=80>
      </td>
    </tr>
    <tr valign=top >
      <td align=right>Sex:</td>
      <td align=left>
        <input type=radio name=sex value="M"> Male <br>
        <input type=radio name=sex value="F"> Female
      </td>
    </tr>

    <tr valign=top >
      <td align=right>Income:</td>
      <td align=left>
        <select name=income size=1>
          <option>Under $25,000
          <option>$25,001 to $50,000
          <option>$50,001 and higher
        </select>
      </td>
    </tr>

    <tr valign=top>
      <td colspan=2 align=center>
        <input type=submit value="Submit Query">
      </td>
    </tr>
  </table>
</form>
```

Notice in the resulting rendered form shown in Figure 9-9 that the table has placed each input element in its own row. The `align` attributes in the table cells force the labels to the right and the elements to the left, creating a vertical margin through the form. By spanning the cell in the last row, the submission button is centered with respect to the entire form. In general, using tables in this manner makes form layout much easier and consistent throughout your documents. If you find this example at all difficult, see Chapter 10, Tables, which explains all the glories of tables in detail.

Figure 9-9. Using a consistent vertical margin to align form elements



You may find other consistent ways to lay out your forms. The key is to find a useful layout style that works well across most browsers and stick with it. Even though HTML and XHTML have limited tools to control layout and positioning, take advantage of what is available in order to make your forms more attractive and easier to use.

9.12 Forms Programming

If you create forms, sooner or later you'll need to create the server-side application that processes your form. Don't panic. There is nothing magic about server-side programming, nor is it overly difficult. With a little practice and some perseverance, you'll be cranking out forms applications.

The most important advice we can give about forms programming is easy to remember: copy others' work. Writing a forms application from scratch is fairly hard; copying a functioning forms application and modifying it to support your form is far easier.

Fortunately, server vendors know this, and they usually supply sample forms applications with their server. Rummage about for a directory named *cgi-src*, and you'll discover a number of useful examples you can easily copy and reuse.

We can't hope to replicate all the useful stuff that came with your server, nor can we provide a complete treatise on forms programming. What we can do is offer a simple example of both GET and POST applications, giving you a feel for the work involved and hopefully getting you moving you in the right direction.

Before we begin, keep in mind that not all servers invoke these applications in the same manner. Our examples cover the broad class of servers derived from the original NCSA HTTP server. They also should work with the Netscape Communications family of server products and the public-domain Apache server. In all cases, consult your server documentation for complete details. You will find more detailed information in *CGI Programming with Perl*, by Scott Guelich, Shishir Gundavaram, and Gunther Birznieks, and *Webmaster in a Nutshell*, by Stephen Spainhour and Robert Eckstein, both published by O'Reilly & Associates.

An alternative to CGI programming is the Java servlet model, covered in *Java Servlet Programming, by Jason Hunter with William Crawford* (O'Reilly & Associates). Servlets can be used to process GET and POST form submissions, although they are actually more general objects. There are no examples of servlets in this book.

9.12.1 Returning Results

Before we begin, we need to discuss how server-side applications end. All server-side applications pass their results back to the server (and on to the user) by writing that result to the application's standard output as a MIME-encoded file. Hence, the first line of the application's output must be a MIME content-type descriptor. If your application returns an HTML document, the first line is:

```
Content-type: text/html
```

The second line must be completely empty. Your application can return some other content type, too—just include the correct MIME type. A GIF image, for example, is preceded with:

```
Content-type: image/gif
```

Generic text that is not to be interpreted as HTML can be returned with:

```
Content-type: text/plain
```

This is often useful for returning the output of other commands that generate plain text instead of HTML.

9.12.2 Handling GET Forms

One of two methods for passing form parameters from client to server is the GET method. In that way, parameters are passed as part of the URL that invokes the server-side forms application. A typical invocation of a GET-style application might use a URL like this:

```
http://www.kumquat.com/cgi-bin/dump_get?name=bob&phone=555-1212
```

When the server processes this URL, it invokes the application named *dump_get* stored in the directory named *cgi-bin*. Everything after the question mark is passed to the application as parameters.

Things diverge a bit at this point, due to the nature of the GET-style URL. While forms place name/value pairs in the URL, it is possible to invoke a GET-style application with only values in the URL. Thus:

```
http://www.kumquat.com/cgi-bin/dump_get?bob+555-1212
```

is a valid invocation as well, with parameters separated by a plus sign (+). This is a common invocation when the application is referenced by a searchable document with the `<isindex>` tag. The parameters typed by the user into the document's text entry field are passed to the server-side application as unnamed parameters separated by plus signs.

If you invoke your GET application with named parameters, your server will pass those parameters to the application in one way; unnamed parameters are passed differently.

9.12.2.1 Using named parameters with GET applications

Named parameters are passed to GET applications by creating an environment variable named `QUERY_STRING` and setting its value to the entire portion of the URL following the question mark. Using our previous example, the value of `QUERY_STRING` would be set to:

```
name=bob&phone=555-1212
```

Your application must retrieve this variable and extract from it the parameter name/value pairs. Fortunately, most servers come with a set of utility routines that performs this task for you, so a simple C program that just dumps the parameters might look like:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_ENTRIES 10000

typedef struct {char *name;
                char *val;
               }entry;

char *makeword(char *line, char stop);
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);

main(int argc, char *argv[])

{  entry entries[MAX_ENTRIES];
    int num_entries, i;
    char *query_string;

/* Get the value of the QUERY_STRING environment variable */
    query_string = getenv("QUERY_STRING");

/* Extract the parameters, building a table of entries */
    for (num_entries = 0; query_string[0]; num_entries++) {
       entries[num_entries].val = makeword(query_string, '&');

       plustospace(entries[num_entries].val);
       unescape_url(entries[num_entries].val);
       entries[num_entries].name =
          makeword(entries[num_entries].val, '=');
       }

/* Spit out the HTML boilerplate */
    printf("Content-type: text/html\n");
    printf("\n");

    printf("<html>");
    printf("<head>");
    printf("<title>Named Parameter Echo</title>\n");
    printf("</head>");
    printf("<body>");
    printf("You entered the following parameters:\n");
    printf("<ul>\n");
```

```
/* Echo the parameters back to the user */
    for(i = 0; i < num_entries; i++)
        printf("<li> %s = %s\n", entries[i].name,
                    entries[i].val);

/* And close out with more boilerplate */
    printf("</ul>\n");
    printf("</body>\n");
    printf("</html>\n");
}
```

The example program begins with a few declarations that define the utility routines that scan through a character string and extract the parameter names and values.[6] The body of the program obtains the value of the `QUERY_STRING` environment variable using the `getenv( )` system call, uses the utility routines to extract the parameters from that value, and then generates a simple HTML document that echoes those values back to the user.

[6] These routines are usually supplied by the server vendor. They are not part of the standard C or UNIX libraries.

For real applications, you should insert your actual processing code after the parameter extraction and before the HTML generation. Of course, you'll also need to change the HTML generation to match your application's functionality.

9.12.2.2 Using unnamed parameters with GET applications

Unnamed parameters are passed to the application as command-line parameters. This makes writing the server-side application almost trivial. Here is a simple shell script that dumps the parameter values back to the user:

```
#!/bin/csh -f
#
# Dump unnamed GET parameters back to the user

echo "Content-type: text/html"
echo
echo '<html>'
echo '<head>'
echo '<title>Unnamed Parameter Echo</title>'
echo '</head>'
echo '<body>'
echo 'You entered the following parameters:'
echo '<ul>'

foreach i ($*)
   echo '<li>' $i
end

echo '</ul>'
echo '</body>'

exit 0
```

Again, we follow the same general style: output a generic document header, including the MIME content type, followed by the parameters and some closing boilerplate. To convert this to a real application, replace the `foreach` loop with commands that actually do something.

9.12.3 Handling POST Forms

Applications that use POST-style parameters expect to read encoded parameters from their standard input. Like GET-style applications with named parameters, they can take advantage of the server's utility routines to parse these parameters.

Here is a program that echoes the POST-style parameters back to the user:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_ENTRIES 10000

typedef struct {char *name;
                char *val;
               } entry;

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);
```

```
main(int argc, char *argv[])

{  entry entries[MAX_ENTRIES];
    int num_entries, i;

/* Parse parameters from stdin, building a table of entries */
    for (num_entries = 0; !feof(stdin); num_entries++) {
       entries[num_entries].val = fmakeword(stdin, '&', &cl);
       plustospace(entries[num_entries].val);
       unescape_url(entries[num_entries].val);
       entries[num_entries].name =
          makeword(entries[num_entries].val, '=');
       }

/* Spit out the HTML boilerplate */
    printf("Content-type: text/html\n");
    printf("\n");
    printf("<html>");
    printf("<head>");
    printf("<title>Named Parameter Echo</title>\n");
    printf("</head>");
    printf("<body>");
    printf("You entered the following parameters:\n");
    printf("<ul>\n");

/* Echo the parameters back to the user */
    for(i = 0; i < num_entries; i++)
        printf("<li> %s = %s\n", entries[i].name,
                    entries[i].val);

/* And close out with more boilerplate */
    printf("</ul>\n");
    printf("</body>\n");
    printf("</html>\n");
}
```

Again, we follow the same general form. The program starts by declaring the various utility routines needed to parse the parameters, along with a data structure to hold the parameter list. The actual code begins by reading the parameter list from the standard input and building a list of parameter names and values in the array named `entries`. Once this is complete, a boilerplate document header is written to the standard output, followed by the parameters and some closing boilerplate.

Like the other examples, this program is handy for checking the parameters being passed to the server application while you are early in the forms and application debugging process. You can also use it as a skeleton for other applications by inserting appropriate processing code after the parameter list is built up and altering the output section to send back the appropriate results.

Chapter 10. Tables

Of all the extensions that found their way into HTML and XHTML, none is more welcome than tables. While tables are useful for the general display of tabular data, they also serve an important role in managing document layout. Creative use of tables, as we'll show in this chapter, can go a long way to enliven an otherwise dull document layout. And you may apply all the CSS styles to the various elements of a table to achieve a desktop-published look and feel.

10.1 The Standard Table Model

The standard model for tables is fairly straightforward: a table is a collection of numbers and words arranged in rows and columns of *cells*. Most cells contain the data values; others contain row and column headers that describe the data.

Define a table and include all of its elements between the `<table>` tag and its corresponding `</table>` end tag. Table elements, including data items, row and column headers, and captions, each have their own markup tag. Working from top to bottom and left to right, you define, in sequence, the header and data for each column cell across the table and progress down row by row.

The latest standards also provide an extended collection of tag attributes, which once were popular extensions to HTML as supported by the major browsers. They make your tables look good, including special alignment of the table values and headers, borders, and table rule lines, and automatic sizing of the data cells to accommodate their content. The various popular browsers have slightly different sets of table attributes; we'll point out those variations as we go.

10.1.1 Table Contents

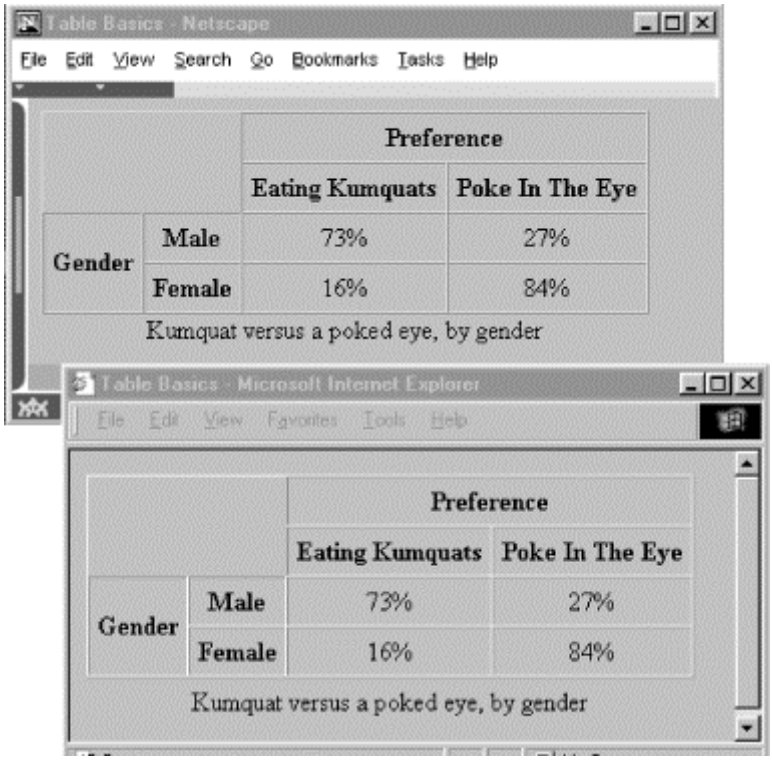
You can put nearly anything you might have within the body of an HTML or XHTML document inside a table cell, including images, forms, rules, headings, and even another table. The browser treats each cell as a window unto itself, flowing the cell's content to fill the space, but with some special formatting provisions and extensions.

10.1.2 An Example Table

Here's a quick example that should satisfy your itching curiosity to see what an HTML table looks like in source code and when finally rendered as in Figure 10-1. More importantly, it shows you the basic structure of a table from which you can infer many of the elements, tag syntax and order, attributes, and so on, and to which you may refer as you read the following various detailed descriptions:

```
<table border cellspacing=0 cellpadding=5>
  <caption align=bottom>
    Kumquat versus a poked eye, by gender</caption>

  <tr>
    <td colspan=2 rowspan=2></td>
    <th colspan=2 align=center>Preference</th>
  </tr>

  <tr>
    <th>Eating Kumquats</th>
    <th>Poke In The Eye</th>
  </tr>

  <tr align=center>
    <th rowspan=2>Gender</th>
    <th>Male</th>
    <td>73%</td>
    <td>27%</td>
  </tr>

 <tr align=center>
    <th>Female</th>
    <td>16%</td>
    <td>84%</td>
  </tr>
</table>
```

Figure 10-1. HTML table example rendered by Netscape (top) and by Internet Explorer (bottom)



10.1.3 Missing Features

At one time, standard HTML tables didn't have all the features of a full-fledged table-generation tool you might find in a popular word processor. Rather, the popular browsers, Internet Explorer and Netscape in particular, provided extensions to the language.

What was missing was support for running headers and footers, particularly useful when printing a lengthy table. Another was control over table rules and divisions.

Today, some browsers are behind; HTML 4 and XHTML standardize the many extensions and introduce new solutions.

10.2 Table Tags

A wide variety of tables can be created with only five tags: the `<table>` tag, which encapsulates a table and its elements in the document's body content; the `<tr>` tag, which defines a table row; the `<th>` and `<td>` tags, which define the table's headers and data cells; and the `<caption>` tag, which defines a title or caption for the table. Beyond these core tags, you may also define and control whole sections of tables, including adding running headers and footers, with the `<colgroup>`, `<col>`, `<tbody>`, `<thead>`, and `<tfoot>` tags. Each tag has one or more required and optional attributes, some of which affect not only the tag itself, but also related tags.

10.2.1 The <table> Tag

The `<table>` tag and its `</table>` end tag define and encapsulate a table within the body of your document. Unless otherwise placed within the browser window by style sheet, paragraph, division-level, or other alignment options, the browser stops the current text flow, breaks the line, inserts the table beginning on a new line, and then restarts the text flow on a new line below the table.

The only content allowed within the `<table>` is one or more `<tr>` tags, which define each row of table contents, along with the various table sectioning tags: `<thead>`, `<tfoot>`, `<tbody>`, `<col>`, and `<colgroup>`.

<table>

Function:

Define a table

Attributes:

ALIGN	ONCLICK
BACKGROUND N I	ONDBLCLICK
BGCOLOR	ONKEYDOWN
BORDER	ONKEYPRESS
BORDERCOLOR N I	ONKEYUP
BORDERCOLORDARK I	ONMOUSEMOVE
BORDERCOLORLIGHT I	ONMOUSEOUT
CELLPADDING	ONMOUSEOVER
CELLSPACING	ONMOUSEUP
CLASS	RULES
COLS N I	STYLE
DIR	SUMMARY
FRAME	TITLE
HEIGHT N I	VALIGN I
HSPACE N	VSPACE N
ID	WIDTH
LANG	
NOWRAP I	

End tag:

</table>; never omitted

Contains:

table_content

Used in:

block

10.2.1.1 The align attribute (deprecated)

The HTML 4 and XHTML standards have deprecated this attribute in favor of the `align` property provided by Cascading Style Sheets. Yet it remains popular and is currently well-supported by the popular browsers.

Like images, tables are rectangular objects that float in the browser display, aligned according to the current text flow. Normally, the browser left-justifies a table, abutting its left edge to the left margin of the display window. Or the table may be centered if under the influence of the `<center>` tag, centered paragraph, or centered division. Unlike images, however, tables are not inline objects. Text content normally flows above and below a table, not beside it. You can change that display behavior with the `align` attribute for the `<table>` tag.

The `align` attribute accepts a value of either `left`, `center`, or `right`, indicating that the table should be placed flush against the left or right margin of the text flow, with the text flowing around the table, or in the middle with text flowing above and below.

Note that the `align` attribute within the `<table>` tag is different from those used within a table's element tags `<tr>`, `<td>`, and `<th>`. In those tags, the attribute controls text alignment within the table's cells, not alignment of the table within the containing text flow.

10.2.1.2 The bgcolor and background attributes

You can make the background of a table a different color than the document's background with the `bgcolor` attribute for the `<table>` tag. The color value for the `bgcolor` attribute must be set to either an RGB color value or a standard color name. Both the syntax of color values and the acceptable color names are provided in Appendix G.

The popular browsers give every cell in the table (including the caption) this background color. You may also set individual row and cell colors by providing the `bgcolor` attribute or a style attribute for those rows or cells.

The `background` attribute, a nonstandard extension supported by the popular browsers, supplies the URL of an image that is tiled to fill the background of the table. The image will be clipped if the table is smaller than the image. By using this attribute with a borderless table, you can put text over an image contained within a document.

10.2.1.3 The border attribute

The optional `border` attribute for the `<table>` tag tells the browser to draw lines around the table and the rows and cells within it. The default is no borders at all. You may specify a value for `border`, but you don't have to with HTML. Alone, the attribute simply enables borders and a set of default characteristics different for each of the popular browsers (reexamine the Figure 10-1 table; it has borders). With XHTML, use `border="border"` to achieve the same default results. Otherwise, in HTML or with XHTML, supply an integer value for `border` equal to the pixel width of the 3D chiseled-edge lines that surround the outside of the table and make it appear to be embossed onto the page.

10.2.1.4 The frame and rules attributes

With Netscape Navigator 4, the `border` attribute is all or nothing, affecting the appearance and spacing both of the frame around the table and the rule lines between data cells. Internet Explorer Version 4 and later, as well as the latest Netscape Navigator Version 6, on the other hand, let you individually modify the various line segments that make up the borders around the table (`frame`) as well as around the data cells (`rules`).

The standard `frame` attribute modifies `border`'s effects for the lines that surround the table. The default value - what you get if you don't use `frame` at all - is `box`, which tells the browser to draw all four lines around the table. The value `border` does the same thing as `box`. The value `void` removes all four of the `frame` segments. The `frame` values `above`, `below`, `lhs`, and `rhs` draw the various border segments on the top, bottom, left, and right side, respectively, of the table. The value `hsides` draws borders on the top and bottom (horizontal) sides of the table; `vsides` draws borders on the left and right (vertical) sides of the table.

With standard tables (supported in Internet Explorer 4 and later, and in Netscape 6), you also may control the thickness of a table's internal cell borders via the `rules` attribute. The default behavior, represented by the value of `all`, is to draw borders around all cells. Specifying `groups` places thicker borders between row and column groups defined by the `<thead>`, `<tbody>`, `<tfoot>`, `<col>`, and `<colgroup>` tags. Using `rows` or `cols` places borders only between every row or column, respectively, while using `none` removes borders from every cell in the table.

10.2.1.5 The bordercolor, bordercolorlight, and bordercolordark attributes

The popular browsers normally draw a table border in three colors, using light and dark variations on the document's background color to achieve a 3D effect. The nonstandard `bordercolor` attribute lets you set the color of the table borders and rules to something other than the background (if borders are enabled, of course). The `bordercolor` attribute's value can be either an RGB hexadecimal color value or a standard color name, both of which are described fully in Appendix G.

Internet Explorer also lets you set the border edge colors individually with special extension attributes: the `bordercolorlight` and `bordercolordark` colors shade the lighter and darker edges of the border.

The effectiveness of the 3D beveled-border effect is tied to the relationship of these two colors. In general, the light color should be about 25 percent brighter than the border color, and the dark color should be about 25 percent darker.

10.2.1.6 The cellspacing attribute

The `cellspacing` attribute controls the amount of space placed between adjacent cells in a table and along the outer edges of cells along the edges of a table.

Browsers normally put two pixels of space between cells and along the outer edges of the table. If you include a `border` attribute in the `<table>` tag, the cell spacing between interior cells grows by two more pixels (four total) to make space for the chiseled edge on the interior border. The outer edges of edge cells grow by the value of the `border` attribute.

By including the `cellspacing` attribute you can widen or reduce the interior cell borders. For instance, to make the thinnest possible interior cell borders, include the `border` and `cellspacing=0` attributes in the table's tag.

10.2.1.7 The cellpadding attribute

The `cellpadding` attribute controls the amount of space between the edge of a cell and its contents, which by default is one pixel. You may make all the cell contents in a table touch their respective cell borders by including `cellpadding=0` in the table tag. You may increase the `cellpadding` space by setting its value greater than one.

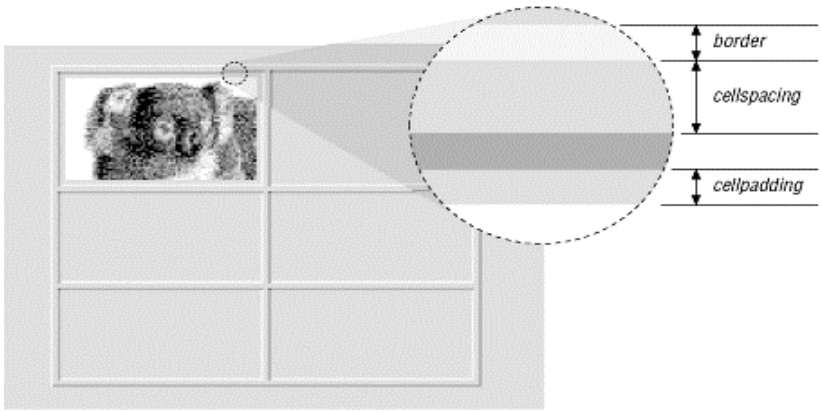
10.2.1.8 Combining border, cellspacing, and cellpadding attributes

The interactions between the `border`, `cellpadding`, and `cellspacing` attributes of the `<table>` tag combine in ways that can be confusing. Figure 10-2 summarizes how the attributes create interior and exterior borders of various widths.

While all sorts of combinations of the `border` and `cellspacing` attributes are possible, these are the most common:

- `border=1` and `cellspacing=0` produces the narrowest possible interior and exterior borders: two pixels wide.
- `border=`*n* and `cellspacing=0` makes the narrowest possible interior borders (two pixels wide), with an external border that is *n* plus one pixels wide.
- `border=1` and `cellspacing=`*n* tables have equal-width exterior and interior borders, all with chiseled edges just one pixel wide. All borders will be *n* plus two pixels wide.

Figure 10-2. The border, cellspacing, and cellpadding attributes of a table



10.2.1.9 The cols attribute

To format a table, the browser must first read the entire table contents, determining the number and width of each column in the table. This can be a lengthy process for long tables, forcing users to wait to see your pages. The nonstandard `cols` attribute tells the browser, in advance, how many columns to expect in the table. The value of this attribute is an integer value defining the number of columns in the table.

The `cols` attribute only advises the browser. If you define a different number of columns, the browser is free to ignore the `cols` attribute in order to render the table correctly. In general, it is good form to include this attribute with your `<table>` tag, if only to help the browser do a faster job of formatting your tables.

10.2.1.10 The valign and nowrap attributes

The `valign` attribute sets the default vertical alignment of data in cells for the entire table. Acceptable values for the `valign` attribute in `<table>` are `top`, `bottom`, `middle`, or `baseline`; the default vertical position is the center of the cell.

Browsers treat each table cell as though it's a browser window unto itself, flowing contents inside the cell as they would common body contents (although subject to special table-cell alignment properties). Accordingly, the browsers automatically wrap text lines to fill the allotted table cell space. The `nowrap` attribute, when included in the `<table>` tag, stops that normal word wrapping in all rows in the table. With `nowrap`, the browser assembles the contents of the cell onto a single line, unless you insert a `<br>` or `<p>` tag, which then forces a break so that the contents continue on a new line inside the table cell.

The `valign` and `nowrap` attributes for the `<table>` tag currently are supported only by Internet Explorer. You achieve similar effects in Netscape by including a `valign` or `nowrap` attribute within the individual `<tr>`, `<td>`, and `<th>` tags.

10.2.1.11 The width and height attributes

Browsers will automatically make a table only as wide as needed to correctly display all of the cell contents. If necessary, you can make a table wider with the `width` attribute.

The value of the `width` attribute is either an integer number of pixels or a relative percentage of the screen width, including values greater than 100 percent. For example:

```
<table width=400>
```

tells the extended browser to make the table 400 pixels wide, including any borders and cell spacing that extend into the outer edge of the table. If the table is wider than 400 pixels, the browser ignores the attribute. Alternatively:

```
<table width="50%">
```

tells the browser to make the table half as wide as the display window. Again, this width includes any borders or cell spacing that extend into the outer edge of the table and has no effect if the table normally is more than half the user's current screen width.

Use relative widths for tables you want to resize automatically to the user's window; for instance, tables you always want to extend across the entire window (`<table width="100%">`). Use an absolute width value for carefully formatted tables whose contents will become hard to read in wide display windows.

For Netscape Navigator and Internet Explorer, you can use the nonstandard `height` attribute to suggest a recommended height for the table. The browser will make the table no shorter than this height but may make the table taller if needed to contain the table's contents. This attribute is useful when trying to stretch tables to fit in a frame or some specific area of a document but is of little use otherwise, particularly since it is not a standard attribute.

10.2.1.12 The summary attribute

The `summary` attribute was introduced to HTML in the 4.0 standard. Its value is a quote-enclosed string which describes the purpose and summarizes the contents of the table. Its intended use, according to the standard, is to provide extended access to non-visual browsers, particularly for users with disabilities.

10.2.2 Common Table Attributes

The HTML and XHTML standards, combined with the Cascading Style Sheets (CSS) standard, provide a number of attributes common not only to the `<table>` tag and the other table creation tags, but for most other tags as well. Except for the CSS-related attributes `class` and `style` for controlling the table display, none of the other standard attributes are yet fully supported by any of the popular browsers.

10.2.2.1 The id and title attributes

Use the `id` attribute with a quote-enclosed string value to uniquely label a table tag for later reference by a hyperlink or an applet. Use the `title` attribute with a string value to optionally entitle the table or any of its segments for general reference. A title's value need not be unique, and it may or may not be used by the browser. Internet Explorer, for example, displays the `title` attribute's text value whenever the user passes the mouse pointer over the element's contents. (Section 4.1.1.4 / Section 4.1.1.5)

10.2.2.2 The dir and lang attributes

Although its contents are predominantly in English, the Web is world-wide. The HTML 4 and XHTML standards take pains to extend the language to all cultures. We support that effort wholeheartedly. The `dir` and `lang` attributes are just small parts of that process.

The `dir` attribute advises the browser as to the direction the text of the contents should flow, from left to right (`dir=ltr`), as for common Western languages like English and German, or right to left (`dir=rtl`), as for common Eastern language like Hebrew and Chinese.

The `lang` attribute lets you explicitly indicate the language used in the table or even individual cell contents. Its value should be an ISO standard two-letter primary code followed by an optional dialect subcode with a hyphen (`-`) between the two.

Currently, `dir` and `lang` are supported by Internet Explorer Version 5 and Netscape 6. (Section 3.6.1.1 / Section 3.6.1.2)

10.2.2.3 The class and style attributes

The Cascading Style Sheets (CSS) standard is the sanctioned way to define display attributes for HTML/XHTML elements, and it is rapidly becoming the only way. Use the `style` attribute to define display characteristics for the table and its elements that take immediate effect and override the display styles that may be currently in effect for the whole document. Use the `class` attribute to reference a style sheet that defines the unique display characteristics for the table and its elements.

We discuss the `class` and `style` attributes and the CSS standard in detail in Chapter 8. (Section 8.1.1 / Section 8.3)

10.2.2.4 The event attributes

The popular browsers have internal mechanisms that detect the various user-initiated mouse and keyboard events that can happen in and around your tables and their elements. For instance, the user might click the mouse pointer in one of the table cells or highlight the caption and then press the Return or Enter key.

With the various event attributes, you can react to these events, such as onClick and onKeyDown, by having the browser execute one or more JavaScript commands or applets that you reference as the value to the respective event attribute. See Chapter 12, for details.

10.2.3 The <tr> Tag

Make a new row in a table with the `<tr>` tag. Place within the `<tr>` tag one or more cells containing headers, defined with the `<th>` tag, and data, defined with the `<td>` tag (see Section 10.2.4). The `<tr>` tag accepts a number of special attributes that control its behavior, along with the common table attributes described in Section 10.2.2

Every row in a table has the same number of cells as the longest row; the browser automatically creates empty cells to pad rows with fewer defined cells.

10.2.3.1 The align and valign attributes

The align attribute for the `<table>` tag may be deprecated in the HTML and XHTML standards, but it is alive and kicking for `<tr>` and other table elements. The `align` attribute for the `<tr>` tag lets you change the default horizontal alignment of all the contents of the cells in a row. The attribute affects all the cells within the current row, but not subsequent rows.

An `align` attribute value of `left`, `right`, `center`, `justify`, or `char` causes the browser to align the contents of each cell in the row against the left or right edge, in the center of the cell, spread across the cell, or to a specified character in the cell, respectively.

<tr>

Function:

Define a row within a table

Attributes:

ALIGN	ONDBLCLICK
BGCOLOR	ONKEYDOWN
BORDERCOLOR O	ONKEYPRESS
BORDERCOLORDARK O	ONKEYUP
BORDERCOLORLIGHT O	ONMOUSEDOWN
CHAR	ONMOUSEMOVE
CHAROFF	ONMOUSEOUT
CLASS	ONMOUSEOVER
DIR	ONMOUSEUP
ID	STYLE
LANG	TITLE
NOWRAP NO	VALIGN
ONCLICK	

End tag:

</tr>; may be omitted in HTML

Contains:

tr_content

Used in:

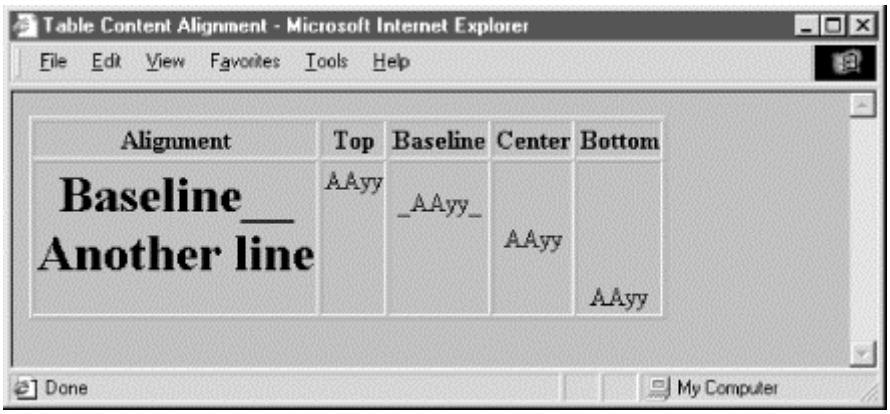
table_content

Similarly, change the default vertical alignment for the contents of data cells contained within a table row with the `valign` attribute. Normally, the browsers render cell contents centered vertically. By including the `valign` attribute in the `<tr>` tag with a value of `top`, `bottom`, or `baseline`, you tell the browser to place the table row's contents flush against the top or bottom of their cells or aligned to the baseline of the top line of text in other cells in the row (Figure 10-3).

The value middle, although acceptable, has no real effect since it simply reiterates the default vertical alignment:

```
<table border="border">
  <tr>
    <th>Alignment</th>
    <th>Top</th>
    <th>Baseline</th>
    <th>Center</th>
    <th>Bottom</th>
  </tr>
  <tr align="center">
    <th><h1>Baseline_  _<br />Another line</h1></th>
    <td valign="top">AAyy</td>
    <td valign="baseline">_AAyy_</td>
    <td valign="center">AAyy</td>
    <td valign="bottom">AAyy</td>
  </tr>
</table>
```

Figure 10-3. Effects of the valign attribute on table cell content alignment



You also may specify the horizontal and vertical alignments for individual cells within a row (Section 10.2.4.1). Use the alignment attributes in the <tr> tag to specify the most common cell content justifications for the row (if not the default), and use a different align or valign attribute for those individual cells that deviate from the common alignment.

Table 10-1 contains the horizontal (align) and vertical (valign) table cell- content attribute values and options. Values in parentheses are the defaults for the popular browsers.

Table 10-1, Table Cell-Content Alignment Attribute Values and Options

Attribute	Netscape and IE Headers	Netscape and IE Data
	Left	(Left)
align	(Center)	Center
	Right	Right
	Justify	Justify
	Char[a]	Char[a]
	Top	Top
valign	(Center)	(Center)
	Bottom	Bottom
	Baseline	Baseline

[a] Value not yet supported.

10.2.3.2 The char and charoff attributes

Even simple word processors let you line up decimal points for numbers in a table. Until the advent of the HTML 4.0 standard, the language was deficient in this feature. Now you may include the `char` attribute to indicate which letter in each of the table row's cells should be the axis for that alignment. You need not include a value with `char`. If you don't, the default character is language-based: it's a period in English, for example, and a comma in French. Include the `char` attribute and a single letter as its value to specify a different alignment character.

Use the `charoff` attribute and an integer value to specify the offset to the first occurrence of the alignment character on each line. If a line doesn't include the alignment character, it should be horizontally shifted to end at the alignment position.

The `char` and `charoff` attributes are new in HTML 4 and XHTML, but are not yet supported by any of the popular browsers.

10.2.3.3 The bgcolor attribute

Like its relative for the `<table>` tag, the `bgcolor` attribute for the `<tr>` tag sets the background color of the entire row.[2] Its value is either an RGB color value or a standard color name. Both the syntax of color values and the acceptable color names are provided in Appendix G.

[2] Unlike `<table>` with Internet Explorer though, `<tr>` does not support a background image.

Every cell in the row will be given this background color. Individual cell colors can be changed by providing the `bgcolor` attribute for those cells.

10.2.3.4 The bordercolor, bordercolorlight, and bordercolordark attributes

Like their nonstandard brethren for the `<table>` tag, Internet Explorer lets you use these attributes to set the color of the borders within the current row.

Their values override any values set by the corresponding attribute in the containing `<table>` tag. See the corresponding description of these extensions in Section 10.2.1.5 for details. Color values can be either an RGB color value or a standard color name, both of which are described fully in Appendix G.

10.2.3.5 The nowrap attribute

Browsers treat each table cell as though it were a browser window unto itself, flowing contents inside the cell as they would common body contents (although subject to special table-cell alignment properties). Accordingly, the browsers automatically wrap text lines to fill the allotted table cell space. The `nowrap` attribute, when included in a table row, stops that normal word wrapping in all cells in that row. With `nowrap`, the browser assembles the contents of the cell onto a single line, unless you insert a `<br>` or `<p>` tag, which then forces a break so that the contents continue on a new line inside the table cell.

10.2.4 The <th> and <td> Tags

The `<th>` and `<td>` tags go inside the `<tr>` tags of a table to create the cells and contents within the row. The tags operate similarly; the only real differences are that the browsers render header text - meant to entitle or otherwise describe table data - in boldface font style and that the default alignment of their respective contents may be different than for data. Data typically gets left-justified by default; headers get centered (Table 10-1).

Like those available for the table row (`<tr>`) tag, the table cell tags support a rich set of style and content-alignment attributes you may apply to a single data or header cell. These attributes override the default values for the current row. There are also special attributes that control the number of columns or rows a cell may span in the table. The `<th>` and `<td>` tags also accept the common table attributes described in Section 10.2.2.

The contents of the `<th>` and `<td>` tags can be anything you might put in the body of a document, including text, images, forms, and so on - even another table. And, as described earlier, the browser automatically creates a table large enough, both vertically and horizontally, to display all the contents of any and all the cells.

If a particular row has fewer header or data items than other rows, the browser adds empty cells at the end to fill the row. If you need to make an empty cell before the end of a row, for instance, to indicate a missing data point, create a header or data cell with no content.

Empty cells look different than those containing data or headers if the table has borders: the empty cell will not be seemingly embossed onto the window, but instead is simply left blank. If you want to create an empty cell that has incised borders like all the other cells in your table, be sure to place a minimal amount of content in the cell: a single `<br>` tag, for instance.

<th> and <td>

Function:

Define table data and header cells

Attributes:

ABBR	NOWRAP
ALIGN	ONCLICK
AXIS	ONDBLCLICK
BACKGROUND N O	ONKEYDOWN
BGCOLOR	ONKEYPRESS
BORDERCOLOR O	ONKEYUP
BORDERCOLORDARK O	ONMOUSEDOWN
BORDERCOLORLIGHT O	ONMOUSEMOVE
CHAR	ONMOUSEOUT
CHAROFF	ONMOUSEOVER
CLASS	ONMOUSEUP
COLSPAN	ROWSPAN
DIR	SCOPE
HEADERS	STYLE
HEIGHT	TITLE
ID	VALIGN
LANG	WIDTH

End tag:

</th> or </td>; may be omitted in HTML

Contains:

body_content

Used in:

tr_content

10.2.4.1 The align and valign attributes

The `align` and `valign` attributes are identical to those of the same name for the table row tag (`<tr>`; see Section 10.2.3), except that when used with a `<th>` or `<td>` tag, they control the horizontal or vertical alignment of content in just the current cell. Their value overrides any alignment established by the respective `align` or `valign` attribute of the `<tr>` tag, but does not affect the alignment of subsequent cells. See Table 10-1 for alignment details.

You may set the `align` attribute's value to `left`, `right`, or `center`, causing the browsers to align the cell contents against the left or right edge, or in the center of the cell, respectively. In addition, Internet Explorer supports a value of `justify` to fill each line of text so that it is flush to both sides of the cell. The `valign` attribute may have a value of `top`, `bottom`, `middle`, or `baseline`, telling the browser to align the cell's contents to the top or bottom edge, or in the center of the cell, or (Netscape only) to the baseline of the first line of text in other cells in the row.

10.2.4.2 The width attribute

Like its twin in the `<table>` tag that lets you widen a table, the `width` attribute for table cell tags lets you widen an individual cell and hence the entire column it occupies. You set the `width` to an integer number of pixels or a percentage indicating the cell's width as a fraction of the table as a whole.

For example:

```
<th width=400>
```

sets the current header cell's width, and hence the entire column of cells, to 400 pixels wide. Alternatively:

```
<td width="40%">
```

creates a data cell with a column occupying 40 percent of the entire table's width.

Since Netscape and Internet Explorer make all cells in a column the same width, you should place a `width` attribute in only one cell within a column, preferably the first instance of the cell in the first row, for source readability. If two or more cells in the same column happen to have `width` attributes, the widest one is honored. You can't make a column thinner than the minimum needed to display all of the cells in the column. So, if the browser determines that the column of cells needs to be at least 150 pixels wide to accommodate all the cells' contents, it will ignore a width attribute in one of the column's cell tags that attempts to make the cell only 100 pixels wide.

10.2.4.3 The height attribute

The `height` attribute lets you specify a minimum height, in pixels, for the current cell. Since all cells in a row have the same height, this attribute need only be specified on one cell in the row, preferably the first. If some other cell in the row needs to be taller to accommodate its contents, this attribute is ignored and all the cells in the row will be set to the larger size.

By default, all the cells in a row are the height of the largest cell in the row that just accommodates its contents.

10.2.4.4 The colspan attribute

It's common to have a table header that describes several columns beneath it, like the headers we use in Table 10-1. Use the `colspan` attribute in a table header or data tag to extend a table cell across two or more columns in its row. Set the value of the `colspan` attribute to an integer value equal to the number of columns you want the header or data cell to span. For example:

```
<td colspan="3">
```

tells the browser to make the cell occupy the same horizontal space as three cells in rows above or below it. The browser flows the contents of the cell to occupy the entire space.

What happens if there aren't enough extra cells on the right? The browser just extends the cell over as many columns as exist to the right; it doesn't add extra empty cells to each row to accommodate an over-extended `colspan` value. You may defeat that limitation by adding the needed extra, but content-less, cells to a single row. (Give them a single `<br>` tag as their contents if you want Netscape's embossed border around them.)

10.2.4.5 The rowspan attribute

Just as the `colspan` attribute layers a table cell across several columns, the `rowspan` attribute stretches a cell down two or more rows in the table.

Include the `rowspan` attribute in the `<th>` or `<td>` tag of the uppermost row of the table where you want the cell to begin and set its value equal to the number of rows you want it to span. The cell then occupies the same space as the current row and an appropriate number of cells below that row. The browser flows the contents of the cell to occupy the entire extended space.

For example:

```
<td rowspan="3">
```

creates a cell that occupies the current row plus two more rows below that.

Like the `colspan` attribute, the browser ignores over-extended `rowspan` attributes and will only extend the current cell down rows you've explicitly defined by other `<tr>` tags following the current row. The browsers will not add empty rows to a table to fill a rowspan below the last defined row in a table.

10.2.4.6 Combining colspan and rowspan

You may extend a single cell both across several columns and down several rows by including both the `colspan` and `rowspan` attributes in its table header or data tag. For example:

```
<th colspan="3" rowspan="4">
```

creates a header cell that, as you might expect, spans across three columns and down four rows, including the current cell and extending two more cells to the right and three more cells down. The browser flows the contents of the cell to occupy the entire space, aligned inside according to the current row's alignment specifications or to those you explicitly include in the same tag, as described earlier.

10.2.4.7 The nowrap attribute

Browsers treat each table cell as though it were a browser window unto itself, flowing contents inside the cell as they would common body contents (although subject to special table-cell alignment properties). Accordingly, the browsers automatically wrap text lines to fill the allotted table cell space. The `nowrap` attribute, when included in a table header or data tag, stops that normal word wrapping. With `nowrap`, the browser assembles the contents of the cell onto a single line, unless you insert a `<br>` or `<p>` tag, which then forces a break so that the contents continue on a new line inside the table cell.

10.2.4.8 The bgcolor and background attributes

Yet again, you can change the background color - this time for an individual data cell. This attribute's value is either an RGB hexadecimal color value or a standard color name. Both the syntax of color values and the acceptable color names are provided in Appendix G.

The `background` attribute, supported only by Internet Explorer, supplies the URL of an image that is tiled to fill the background of the cell. The image will be clipped if the cell is smaller than the image.

Neither `background` nor `bgcolor` will override a related style sheet property.

10.2.4.9 The bordercolor, bordercolorlight, and bordercolordark attributes

Internet Explorer lets you alter the colors that make up an individual cell's border - if table borders are turned on with the `border` attribute, of course. See the respective attributes' descriptions under the `<table>` tag in Section 10.2.1.5 for details.

The values for these three attributes override any values set for the containing `<table>` or `<tr>` tag. Their values can be either an RGB color value or a standard color name, both of which are described fully in Appendix G.

10.2.4.10 The char and charoff attributes

Just as for the `<tr>` tag, you may use the `char` attribute with `<th>` or `<td>` to indicate which letter in the table cell should be the axis for alignment, such as for decimal numbers. You need not include a value with `char`. If you don't, the default character is language-based: it's a period in English, for example, and a comma in French. Include the `char` attribute and a single letter as its value to specify a different alignment character.

Use the `charoff` attribute and an integer value to specify the offset to the first occurrence of the alignment character in the cell. If a cell doesn't include the alignment character, it should be horizontally shifted to end at the alignment position.

The `char` and `charoff` attributes are standard in HTML 4 and XHTML but are not yet supported by any of the popular browsers.

10.2.4.11 The headers and scope attributes

The `headers` attribute associates header cells with a data cell in the table. The value of this attribute is a quote-enclosed list of names that have been defined for various header cells using the `id` attribute. The `headers` attribute is especially useful for nonvisual browsers, which might speak the contents of a header cell before presenting the associated data cell contents.

Use the scope attribute to associate data cells with a header cell. With a value of row, all cells in the header's row are associated with the header cell. Specifying col binds all the cells in the current column to the cell. Using rowgroup or colgroup binds all the cells in the cell's row group (defined by a <thead>, <tbody>, or <tfoot> tag) or column group (defined by a <col> or <colgroup> tag) with the header cell.

10.2.4.12 The abbr attribute

The value of this attribute should be an abbreviated description of the cell's contents. When short on space, browsers might choose to render the abbreviation instead, or to use it in nonvisual contexts.

10.2.4.13 The axis attribute

Tables are usually chock-full of data, prompting the reader to ask questions. A tabular expense report, for example, naturally leads to queries like "How much did I spend on meals?" or "What did my cab fares total?" In the future, browsers may support such queries with the help of the axis attribute.

The value of this attribute is a quote-enclosed list of category names that might be used to form a query. As a result, if you used axis=meals on the cells containing meal purchases, the browser could locate those cells, extract their values, and produce a sum.

10.2.5 The <caption> Tag

<caption>

Function:

Define a table caption

Attributes:

ALIGN	ONKEYUP
CLASS	ONMOUSEDOWN
DIR	ONMOUSEMOVE
ID	ONMOUSEOUT
LANG	ONMOUSEOVER
ONCLICK	ONMOUSEUP
ONDBLCLICK	STYLE
ONKEYDOWN	TITLE
ONKEYPRESS	VALIGN

End tag:

</caption>; never omitted

Contains:

body_content

Used in:

table_content

A table commonly needs a caption to explain its contents, so the popular browsers provide a table-caption tag. Authors typically place the <caption> tag and its contents immediately after the <table> tag, but it can be placed nearly anywhere inside the table and between the row tags. The caption may contain any body content, much like a cell within a table.

10.2.5.1 The align and valign attributes

By default, browsers place the caption's contents centered above the table. You may place it below the table with the align attribute set to the value bottom (the value top, of course, is equivalent to the default).

With Internet Explorer, you may alternatively use the align attribute to control the horizontal position of the caption and use the valign attribute to change the caption's vertical position. Set the align attribute to left, center (the default), or right to position the caption at the respective location relative to the table. Use the valign attribute to place a caption at the top or bottom of the table. The other browsers ignore Internet Explorer's different caption-align values and attributes.

10.2.5.2 The many other attributes

Like the other table tags, <caption> supports the many and various language-, event-, and styles-related attributes, which are described in Section 10.2.2. Use them in good health. Just be sure to use the contextual selector TABLE CAPTION when referring to caption styles at the document level or in external style sheets.

10.3 Newest Table Tags

While it is possible to build a simple table quickly, complex tables with varying border styles, running headers and footers, and column-based layout were not easily constructed from the old HTML 3.2 table model. Microsoft had rectified this inadequacy somewhat by adding a number of table layout controls into Internet Explorer Version 3.0. These very useful extensions found their way into the HTML 4 standard and subsequently into XHTML 1.0. They provide row-based grouping and running headers and footers, along with column-based layout features.

There is good news and bad news about these new table features, of course. They provide a nice way to make your tables more attractive and presentable, but they currently work only within Internet Explorer and the very latest Netscape Navigator Version 6. If you choose to use them, make sure your tables stand up with the older browsers, too.

10.3.1 Defining Table Sections

Within tables, all rows are created equal. In real tables, some rows are more equal than others. And most tables have header and footer rows that repeat from page to page. In large tables, adjacent rows are grouped and delineated with different rules to make the table easier to read and understand. HTML 4 and XHTML support all of these features with the <thead>, <tfoot>, and <tbody> tags.

10.3.2 The <thead> Tag

Use the <thead> tag to define a set of table header rows. The <thead> tag may appear once within a <table> tag, at the beginning. Within the <thead> tag, you may place one or more <tr> tags, defining the rows within the table header. If given the opportunity, the HTML 4/XHTML-compliant browser will replicate these heading rows when the table is printed or displayed in multiple sections. Thereafter, it will repeat these headings on each printed page if the table appears on more than one page.

The ending </thead> tag is optional for HTML. Since the <thead> tag only appears in tables where, presumably, other rows will be designated as the table body or footer, the <thead> tag is automatically closed when the browser encounters a <tbody> or <tfoot> tag or when the table ends.

The many attributes of the <thead> tag operate identically, take the same values, and affect all the enclosed <tr> contents as if you had specified them individually for each <tr> entry. For example, the align attribute accepts values of left, right, center, or justify, controlling the horizontal alignment of text in all the heading's rows. Similarly, the valign attribute accepts values of top, middle, baseline, or bottom, dictating the vertical alignment of text in all of the heading rows.

If you don't specify any alignments or styles, the browser centers the heading text vertically and horizontally within the respective cells, equivalent to specifying align=center and valign=middle for each. Of course, individual row and cell or style sheet specifications may override these attributes.

<thead>

Function:

Define a table header

Attributes:

ALIGN	ONKEYPRESS
CHAR	ONKEYUP
CHAROFF	ONMOUSEDOWN
CLASS	ONMOUSEMOVE
DIR	ONMOUSEOUT
ID	ONMOUSEOVER
LANG	ONMOUSEUP
ONCLICK	STYLE
ONDBLCLICK	TITLE
ONKEYDOWN	VALIGN

End tag:

</thead>; may be omitted in HTML

Contains:

table_content

Used in:

table_content

10.3.3 The <tfoot> Tag

Use the <tfoot> tag to define a footer for a table. The <tfoot> tag may appear only once, just before the end of a table. Like <thead>, it may contain one or more <tr> tags that let you define those rows that Internet Explorer (Version 3 or later) or an HTML 4/XHTML-compliant browser uses as the table footer. There-after, the browser repeats these rows if the table is broken across multiple physical or virtual pages. Most often, the browser repeats the table footer at the bottom of each portion of a table printed on multiple pages.

The closing </tfoot> tag is optional in HTML, since the footer ends when the table ends.

<tfoot>

Function:

Define a table footer

Attributes:

ALIGN	ONKEYPRESS
CHAR	ONKEYUP
CHAROFF	ONMOUSEDOWN
CLASS	ONMOUSEMOVE
DIR	ONMOUSEOUT
ID	ONMOUSEOVER
LANG	ONMOUSEUP
ONCLICK	STYLE
ONDBLCLICK	TITLE
ONKEYDOWN	VALIGN

End tag:

</tfoot>; may be omitted in HTML

Contains:

table_content

Used in:

table_content

10.3.4 The <tbody> Tag

Use the <tbody> tag to divide your table into discrete sections. The <tbody> tag collects one or more rows into a group within a table. It is perfectly acceptable to have no <tbody> tags within a table, although where you might include one, you probably will have two or more <tbody> tags within a table. So identified, you can give each <tbody> group different rule line sizes above and below the section. Within a <tbody> tag, only table rows may be defined using the <tr> tag. And, by definition, a <tbody> section of a table stands alone. For example, you may not span from one <tbody> into another.

<tbody>

Function:

Define a section within a table

Attributes:

ALIGN	ONKEYPRESS
CHAR	ONKEYUP
CHAROFF	ONMOUSEDOWN
CLASS	ONMOUSEMOVE
DIR	ONMOUSEOUT
ID	ONMOUSEOVER
LANG	ONMOUSEUP
ONCLICK	STYLE
ONDBLCLICK	TITLE
ONKEYDOWN	VALIGN

End tag:

</tbody>; may be omitted in HTML

Contains:

table_content

Used in:

table_content

The closing </tbody> tag is optional in HTML, since the section ends at the next <tbody> or <tfoot> tag, or when the table ends. Like <tfoot>, there are many attributes for the <tbody> tag, but none supported, even by Internet Explorer. If you have special alignment attributes for this section, you'll need to specify them for each row within the <tbody> tag.

10.3.5 Using Table Sections

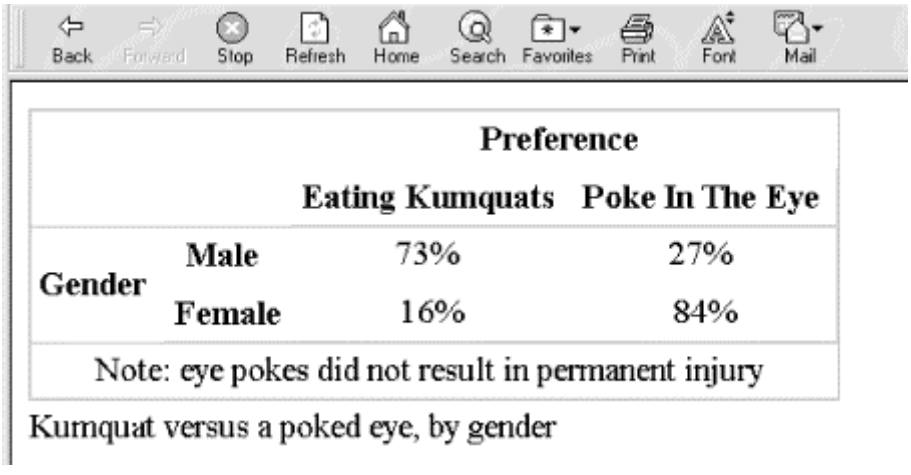
From a presentation standpoint, the most important thing you can do with the <thead>, <tfoot>, and <tbody> tags is divide your table into logical sections that are delimited by different borders. By default, Internet Explorer does not do anything special with the borders around the headers, footers, and sections within your table. By adding the rules attribute to the <table> tag, however, you can draw thicker rule lines between your <thead>, one or more <tbody>, and <tfoot> table sections, helping readers better understand your table's organization. Section 10.2.1.1

For example, here is the simple table you saw earlier in this chapter augmented with a header and footer. Notice that we've omitted many of the closing tags for brevity and readability of HTML, but that the tags must appear in an XHTML-compliant document:

```
<table border cellspacing=0 cellpadding=5 rules=groups>
  <caption align=bottom>Kumquat versus a poked eye, by gender</caption>
  <thead>
    <tr>
      <td colspan=2 rowspan=2>
      <th colspan=2 align=center>Preference
    </tr>
    <tr>
      <th>Eating Kumquats
      <th>Poke In The Eye
    </tr>
  </thead>
  <tbody>
    <tr align=center>
      <th rowspan=2>Gender
      <th>Male
      <td>73%
      <td>27%
    </tr>
    <tr align=center>
      <th>Female
      <td>16%
      <td>84%
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td colspan=4 align=center>
        Note: eye pokes did not result in permanent injury
</table>
```

The resulting table as rendered by Internet Explorer Version 4 is shown in Figure 10-4. Notice how the rules after the table header and before the footer are thinner than the borders around the other table rows? This happened because we included the special rules=groups attribute to the <table> tag. Similar effects may be obtained by specifying rules=rows or rules=all.

Figure 10-4. Use HTML 4/XHTML table tags to specially section your tables



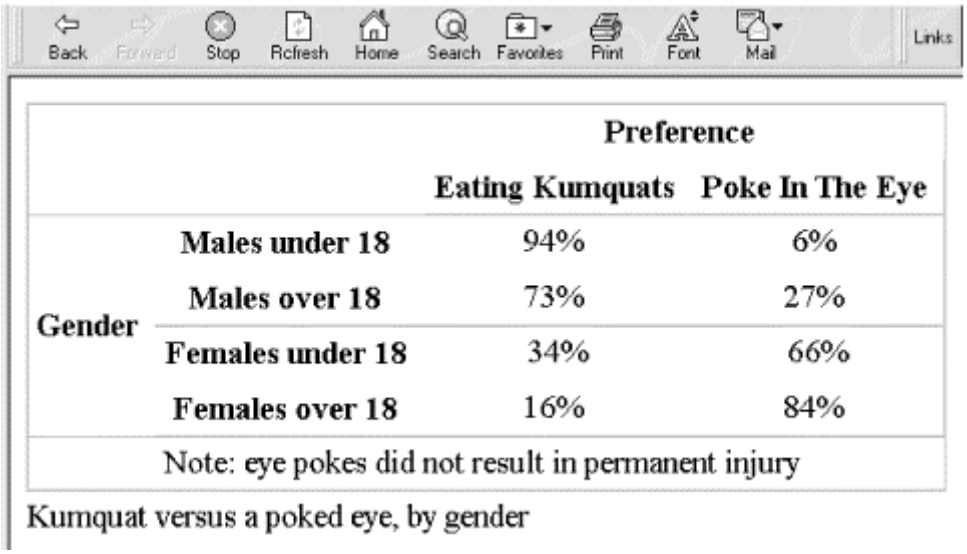
Long tables often benefit from thicker rules every few rows, making it easier to read the table. Do this by grouping the rules in your table with several <tbody> tags. Each set of rows contained in a single <tbody> tag will have thicker rules before and after it.

Here is an expanded version of our HTML table example, with additional sections set off as separate groups:

```
<table border cellspacing=0 cellpadding=5 rules=groups>
  <caption align=bottom>Kumquat versus a poked eye, by gender</caption>
  <thead>
    <tr>
      <td colspan=2 rowspan=2>
      <th colspan=2 align=center>Preference
    <tr>
      <th>Eating Kumquats
      <th>Poke In The Eye
  <tbody>
    <tr align=center>
      <th rowspan=4>Gender
      <th>Males under 18
      <td>94%
      <td>6%
    <tr align=center>
      <th>Males over 18
      <td>73%
      <td>27%
    <tr align=center>
      <th>Females under 18
      <td>34%
      <td>66%
    <tr align=center>
      <th>Females over 18
      <td>16%
      <td>84%
  <tfoot>
    <tr>
      <td colspan=4 align=center>
        Note: eye pokes did not result in permanent injury
</table>
```

The result is shown in Figure 10-5 as rendered by Internet Explorer Version 4. In this case, we wind up with four rows in the table, separated into two groups by a thicker rule between them. Any number of groups could be created within the table by adding more `<tbody>` tags.

Figure 10-5. Multiple <tbody> segments further divide a table

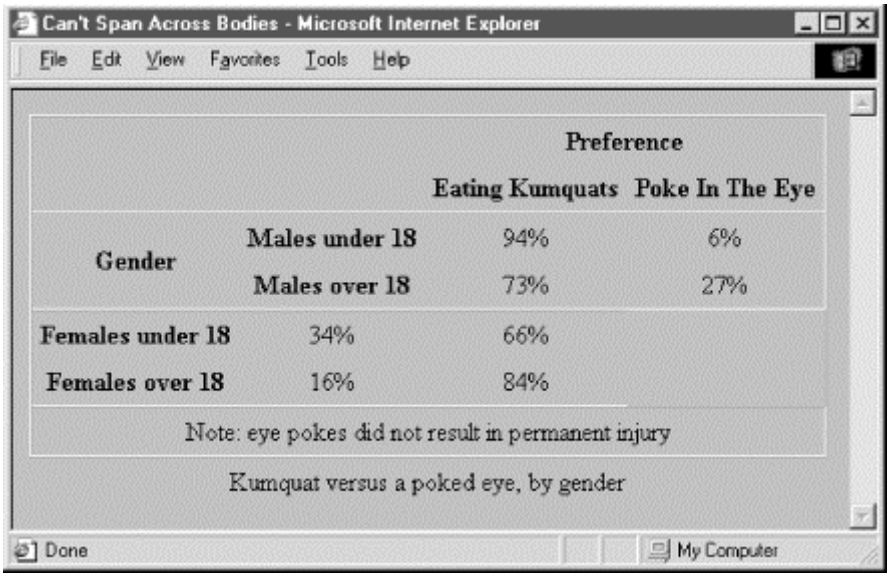


		Preference	
		Eating Kumquats	Poke In The Eye
Gender	Males under 18	94%	6%
	Males over 18	73%	27%
	Females under 18	34%	66%
	Females over 18	16%	84%
Note: eye pokes did not result in permanent injury			

Kumquat versus a poked eye, by gender

Note that Netscape Version 4 and earlier will properly display the example table, but not Internet Explorer Version 5 or Netscape 6. Why? Because, according to the latest standards, the Gender column may not span across `<tbody>` sections. The result is shown in Figure 10-6.

Figure 10-6. You cannot span across <tbody> sections of HTML 4 or XHTML tables



10.3.6 Defining Column Groups

The basic table model is row-centric. Sometimes, though, it is easier to deal with your table as a collection of columns. Using the `<colgroup>` and `<col>` tags, HTML 4 and XHTML, as originally implemented by Internet Explorer through table extensions, help you turn the tables and think in columns.

Unlike the sectioning tags described in the previous sections, which are interspersed with the rows of a table to define headers, footers, and sections within the table, the column-related tags cannot be intermingled with the content of a table. You must place them at the very beginning of a table, before the content. They define the model by which HTML 4/XHTML-compliant browsers render the columns.

10.3.7 The <colgroup> Tag

The `<colgroup>` tag defines a column group. You can use the `<colgroup>` tag in two ways: as a single definition of several identical columns or as a container for several dissimilar columns. The `<colgroup>` tag may appear only within a `<table>` tag, but you may define one or more column groups within a table. The ending `</colgroup>` tag is rarely used; instead, the `<colgroup>` ends at the next `<colgroup>`, `<thead>`, `<tbody>`, `<tfoot>`, or `<tr>` tag.

Internet Explorer and Netscape 6 support `<colgroup>`.

10.3.7.1 The span attribute

Use the `span` attribute with the `<colgroup>` tag to achieve the first type of column grouping. The value of the `span` attribute is the integer number of columns affected by the `<colgroup>` tag. For example, a table with six columns - four in the first group and two in the other - would appear in the source code as:

```
<colgroup span="4">
<colgroup span="2">
```

When an HTML 4/XHTML-compliant browser collects the table cells into columns by the example definition, it groups the first four cells in each row as the first column group and the next two cells into a second column group. Any other attributes of the individual `<colgroup>` tags then are applied to the columns contained within that group.

10.3.7.2 When to span and col

To use the `<colgroup>` tag as a container for dissimilar columns, leave out the `span` attribute, but include within each `<colgroup>` tag an individual `<col>` tag for each column within the group. For instance, in HTML:

```
<colgroup>
  <col>
  <col>
  <col>
  <col>
<colgroup>
  <col>
  <col>
```

This method creates the same number of columns in each group as we had with the `span` attribute, but lets you specify column attributes individually. You can still supply attributes for all the columns via the `<colgroup>` tag, but they will be overridden by the attributes in the `<col>` tags, as appropriate.

<colgroup>

Function:

Define a column group within a table

Attributes:

ALIGN	ONKEYUP
CHAR	ONMOUSEDOWN
CHAROFF	ONMOUSEMOVE
CLASS	ONMOUSEOUT
DIR	ONMOUSEOVER
ID	ONMOUSEUP
LANG	SPAN
ONCLICK	STYLE
ONDBLCLICK	TITLE
ONKEYDOWN	VALIGN
ONKEYPRESS	WIDTH

End tag:

</colgroup>; usually omitted in HTML

Contains:

column_content

Used in:

table_content

For instance, suppose we want our first example group of four columns to each occupy 20% of the table, with the remaining two columns taking up 10% each of the total table width. That's easy with the span attribute:

```
<colgroup span=4 width="20%">
<colgroup span=2 width="10%">
```

The structure also can be done with individually specified columns (in HTML):

```
<colgroup width="20%">
  <col>
  <col>
  <col>
  <col>
<colgroup width="10%">
  <col>
  <col>
```

There is no reason not to use both methods in the same table. For instance, we could specify our example column groupings, complete with `width` attributes:

```
<colgroup span=4 width="20%" align=right>
<colgroup width="10%">
  <col align=left>
  <col align=right>
```

Notice that this lets us align the contents of the two columns of the second group individually (the default alignment is centered).

10.3.7.3 The other <colgroup> attributes

The many attributes common to tables control the familiar aspects of each column in the `<colgroup>`-encapsulated column group. These attributes accept the same values and behave exactly like the equivalent attributes for the `<td>` tag.

10.3.8 The <col> tag

Use the `<col>` tag to control the appearance of one or more columns within a column group.

<col>

Function:

Define a column within a column group

Attributes:

ALIGN	ONKEYUP
CHAR	ONMOUSEDOWN
CHAROFF	ONMOUSEMOVE
CLASS	ONMOUSEOUT
DIR	ONMOUSEOVER
ID	ONMOUSEUP
LANG	SPAN
ONCLICK	STYLE
ONDBLCLICK	TITLE
ONKEYDOWN	VALIGN
ONKEYPRESS	WIDTH

End tag:

None in HTML; </col> or <col ... /> with XHTML

Contains:

Nothing

Used in:

column_content

The <col> tag may appear only within a <colgroup> tag within a table. It has no content, and thus has no ending tag with HTML. Use </col> or a lone forward slash at the end of the tag, such as <col /> for the required XHTML end tag. The <col> tag represents one or more columns within a <colgroup> to which an HTML 4/XHTML-compliant browser applies the <col> tag's attributes.

Internet Explorer and Netscape 6 support the <col> tag.

10.3.8.1 The span attribute

The span attribute for the <col> tag, like for the <colgroup> tag, lets you specify how many successive columns are affected by this <col> tag. By default, only one is affected. For example, let's create a <colgroup> that has five columns. We align the first and last columns to the left and right, respectively, while the middle three are centered:

```
<colgroup>
  <col align=left>
  <col align=center span=3>
  <col align=right>
```

The <col> tag should only be used within <colgroup> tags that do not themselves use the span attribute. Otherwise, Internet Explorer, Netscape Navigator Version 6, and future HTML 4/XHTML-compliant browsers ignore the individual <col> tags and their attributes.

10.3.8.2 The other <col> attributes

The many attributes common to tables control the familiar aspects of the column defined by the <col> tag. These attributes accept the same values and behave exactly like the equivalent attributes for the <td> tag.

10.3.9 Using Column Groups

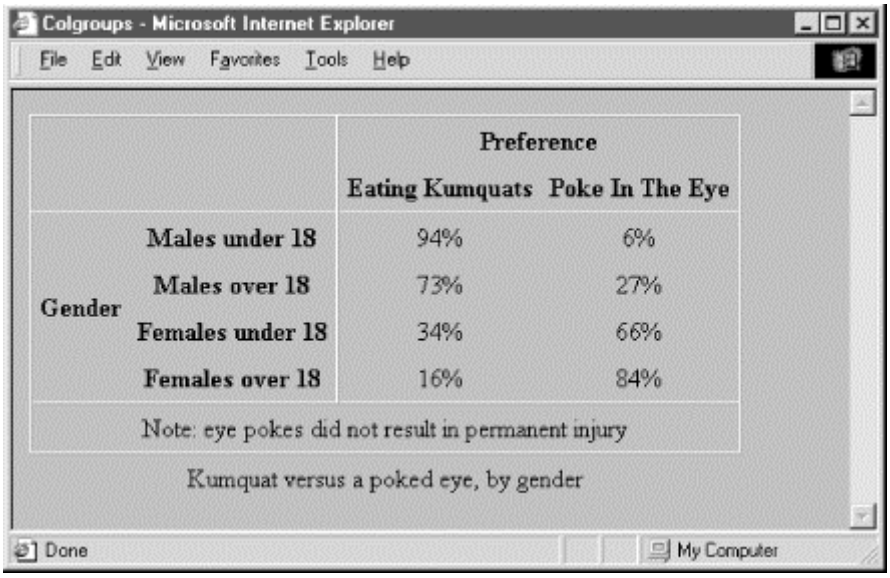
Column groups are easier to use than they first appear. Think of them as a template for how to format your table columns. Their main purpose is to create groups that can be separated by thicker rules within your table, and to streamline the process of applying formatting attributes to all the cells in one or more columns.

Returning to our original table example, we can place a thicker rule between the column labels and the data cells by placing the column labels in one column group and the data cells in another (in HTML):

```
<table border= cellspacing=0 cellpadding=5 rules=groups>
  <caption align=bottom>Kumquat versus a poked eye, by gender</caption>
  <colgroup span=2>
  <colgroup span=2>
  <thead>
    <tr>
      <td colspan=2 rowspan=2>
      <th colspan=2 align=center>Preference
    <tr>
      <th>Eating Kumquats
      <th>Poke In The Eye
  <tbody>
    <tr align=center>
      <th rowspan=4>Gender
      <th>Males under 18
      <td>94%
      <td>6%
    <tr align=center>
      <th>Males over 18
      <td>73%
      <td>27%
    <tr align=center>
      <th>Females under 18
      <td>34%</td>
      <td>66%</td>
    <tr align=center>
      <th>Females over 18
      <td>16%
      <td>84%
  <tfoot>
    <tr>
      <td colspan=4 align=center>
        Note: eye pokes did not result in permanent injury
</table>
```

The results are shown in Figure 10-7. All we added were the two <colgroup> tags; the additional borders were drawn by the rules=groups attribute in the <table> tag. For borders between column groups to be drawn, the rules attribute must be set to groups, cols, or all.

Figure 10-7. Example demonstrating the various HTML 4/XHTML new table features



10.4 Beyond Ordinary Tables

On the face of it, tables are pretty ordinary: just a way for academics and other like-minded data crunchers to format items into columns and rows for easy comparison. Scratch below the surface, though, and you will see that tables are really extraordinary. Besides `<pre>`, the `<table>` tag and related attributes provide the only way for you to easily control the *layout* of your document. The content inside a `<pre>` tag, of course, is very limited. Tables, on the other hand, may contain nearly anything allowed in normal body content, including multimedia and forms. And the table structure lets you explicitly control where those elements appear in the users' browser window. With the right combinations of attributes, tables provide a way for you to create multicolumn text, and side and straddle heads. They also enable you to make your forms easier to read, understand, and fill out. That's just for starters.

We don't know that we can recommend getting too caught up with page layout - tables or beyond. Remember, it ain't about looks, it's about content. But ...

It's easy to argue that at least tables of information benefit from some controlled layout, and that forms follow a close second. Tables provide the only way to create predictable, browser-independent layouts for your web pages. Used in moderation and filled with quality content, tables are a tool that every author should be able to wield.

And now that we've whetted your appetite for page layout with tables, don't despair that we've let you down by ending this chapter without examples - we have several in Chapter 17.

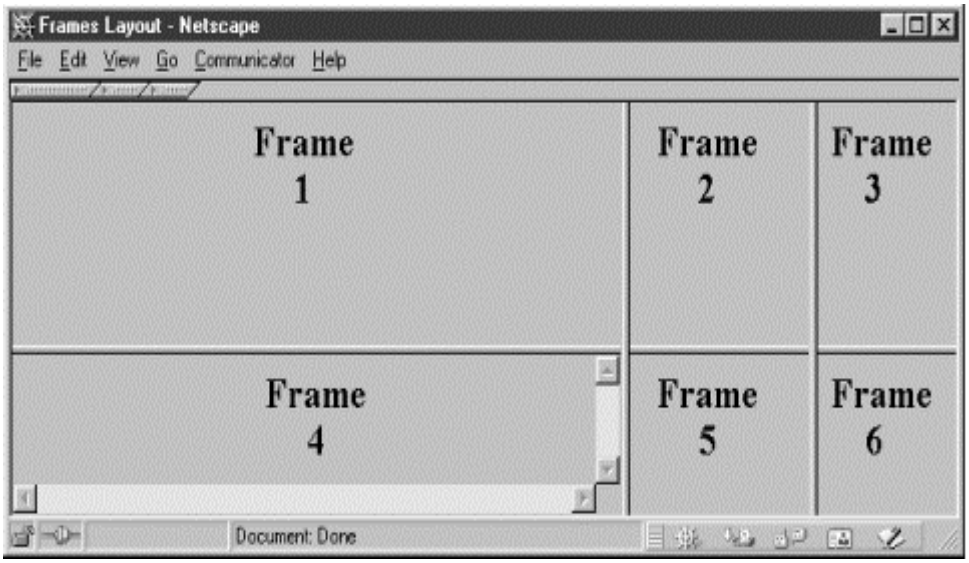
Chapter 11. Frames

Beginning with Netscape Navigator 2.0, HTML authors have been able to divide the browser's main display window into independent window *frames*, each simultaneously displaying a different document - something like a wall of monitors in a TV control room. Instantly popular, frames were adopted (and extended) by Microsoft for Internet Explorer and are standard features for HTML 4 and XHTML.

11.1 An Overview of Frames

Figure 11-1 is a simple example of a frame display. It shows how the document window may be divided into columns and rows of individual frames separated by rules and scroll bars. Although it is not immediately apparent in the example, each frame in the window is displaying an independent document. We use different HTML documents in the example, but the individual documents may contain any valid content the browser is capable of displaying, including XHTML documents and multimedia. If the frame's contents include a hyperlink that the user selects, the new document's contents - even another frame document - may replace that same frame, another frame's content, or the entire browser window.

Figure 11-1. A simple six-panel frame layout for Netscape



Frames are enabled with a special frame document. Its contents do not get displayed. Rather, the frame document contains extension tags that tell the browser how to divide its main display window into discrete frames and what documents go inside the frames.

The individual documents referenced and displayed in the frame document window act independently, to a degree; the frame document controls the entire window. You can, however, direct one frame's document to load new content into another frame. That's done by attaching a name to a frame and targeting the named frame with a special attribute for the hyperlink `<a>` tag.

11.2 Frame Tags

You need to know only three tags to create a frame document: `<frameset>`, `<frame>`, and `<noframes>`. In addition, the HTML 4 and XHTML standards provide the `<iframe>` tag, which you may use to create inline, or *floating*, frames.

A *frameset* is simply the collection of frames that make up a browser's window. Column- and row-definition attributes for the `<frameset>` tag let you define the number and initial sizes for the columns and rows of frames. The `<frame>` tag defines which document - HTML or otherwise - initially goes into the frames within those framesets and is where you may give the frame a name to use for document hypertext links.

Here is the HTML source that was used to generate Figure 11-1:

```
<html>
<head>
<title>Frames Layout</title>
</head>
<frameset rows="60%,*" cols="65%,20%,*">
  <frame src="frame1.html">
  <frame src="frame2.html">
  <frame src="frame3.html" name="fill_me">
  <frame scrolling=yes src="frame4.html">
  <frame src="frame5.html">
  <frame src="frame6.html">
  <noframes>
    Sorry, this document can be viewed only with a
    frames-capable browser.
    <a href = "frame1.html">Take this link</a>
    to the first HTML document in the set.
  </noframes>
</frameset>
</html>
```

Notice a few things in the simple frame example and its rendered image (Figure 11-1). First, the order in which the browser fills the frames in a frameset goes across each row. Second, Frame 4 sports a scrollbar because we told it to, even though the contents may otherwise fit without scrolling. (Scrollbars automatically appear if the contents overflow the frame's dimensions, unless explicitly disabled with the `scrolling` attribute in the `<frame>` tag.) Section 11.4.1

Another item of interest is the `name` attribute in one of the frame tags. Once named, you can reference a particular frame as the location in which to display a hypertext-linked document. To do that, you add a special `target` attribute to the anchor (`<a>`) tag of the source hypertext link. For instance, to link a document called `new.html` for display in Frame 3, which we've named "fill_me", the anchor looks like this:

```
<a href="new.html" target="fill_me">
```

If the user chooses the link, say in Frame 1, the *new.html* document will replace the original *frame3.html* contents in Frame 3. Section 11.7.1

Finally, although Netscape and Internet Explorer both support frames, it is possible that users with some other browser will try and view your frame documents. That's why each of your key frame documents should provide a back door to your document collection with the `<noframes>` tag. Frame-capable browsers display your frames; non-frame-capable browsers display the alternative `<noframes>` content.

11.2.1 What's in a Frame?

Anyone who has opened more than one window on their desktop display to compare contents or operate interrelated applications knows instinctively the power of frames.

One simple use for frames is to put content that is common in a collection, such as copyright notices, introductory material, and navigational aids, into one frame, with all other document content in an adjacent frame. As the user visits new pages, each loads into the scrolling frame, while the fixed-frame content persists.

A richer frame document-enabled environment provides navigational tools for your document collections. For instance, assign one frame to hold a table of contents and various searching tools for the collection. Have another frame hold the user-selected document contents. As users visit your pages in the content frame, they never lose sight of the navigational aids in the other frame.

Another beneficial use of frame documents is to compare a returned form with its original for verification of the content by the submitting user. By placing the form in one frame and its submitted result in another, you let the user quickly verify that the result corresponds to the data entered in the form. If the results are incorrect, the form is readily available to be filled out again.

11.3 Frame Layout

Frame layout is similar to table layout. Using the `<frameset>` tag, you can arrange frames into rows and columns while defining their relative or absolute sizes.

11.3.1 The <frameset> Tag

The `<frameset>` tag lets you define a collection of frames and control their spacing and borders. Use the `<frameset>` tag to define a collection of frames and other framesets. Framesets also may be nested, allowing for a richer set of layout capabilities.

<frameset>

Function:

Define a collection of frames

Attributes:

BORDER NO	ONLOAD
BORDERCOLOR NO	ONUNLOAD
CLASS	ROWS
COLS	STYLE
FRAMEBORDER NO	TITLE
FRAMESPACING O	
ID	

End tag:

</frameset>; never omitted

Contains:

frameset_content

Used in:

html_content

Use the `<frameset>` tag in lieu of a `<body>` tag in the frame document. You may not include any other content except valid `<head>` and `<frameset>` content in a frame document. Combining frames with a conventional document containing a `<body>` section may result in unpredictable browser behavior.

11.3.1.1 The rows and cols attributes

The `<frameset>` tag has one required attribute: either `cols` or `rows` - your choice. They define the size and number of columns (`cols`) or `rows` of either frames or nested framesets for the document window. Both attributes accept a quote-enclosed, comma-separated list of values that specify either the absolute (pixels) or relative (percentage or remaining space) width (for columns) or height (for rows) for the frames. The number of attribute values determines how many rows or columns of frames the browser will display in the document window.

As with tables, the browser will match the size you give a frameset as closely as possible. The browser will not, however, extend the boundaries of the main document window to accommodate framesets that would otherwise exceed those boundaries or fill the window with empty space if the specified frames don't fill the window. Rather, the browsers allocate space to a particular frame relative to all other frames in the row and column and resolutely fills the entire document window. (Did you notice that a frame document window does not have scrollbars?)

For example:

```
<frameset rows="150,300,150">
```

creates three rows of frames, each extending across the entire document window. The first and last frames are set to 150 pixels tall, the second to 300 pixels. In reality, unless the browser window is exactly 600 pixels tall, the browser automatically and proportionately stretches or compresses the first and last frames so that each occupies one quarter of the window space. The center row occupies the remaining half of the window space.

Frame row and column size values expressed as a percentage of the window dimensions are more sensible. For instance, the following example is effectively identical to the previous one:

```
<frameset rows="25%,50%,25%">
```

Of course, if the percentages don't add up to 100 percent, the browser automatically and proportionally resizes each row to make up the difference.

If you are like us, making things add up is not a strength. Perhaps some of the frame designers suffer the same difficulty, which would explain why they included the very nifty asterisk option for `<frameset>` `rows` and `cols` values. It tells the browser to size the respective column or row to whatever space is left over after putting adjacent frames into the frameset.

For example, when the browser encounters the following frame tag:

```
<frameset cols="100,*">
```

it makes a fixed-sized column 100 pixels wide, and then creates another frame column that occupies all of the remaining space in the frameset.

Here's a fancier layout example:

```
<frameset cols="10,*,10">
```

This one creates two very thin columns down the edges of the frameset and gives the remaining center portion to the middle column.

You may also use the asterisk for more than one row- or column-attribute value. In that case, the corresponding rows or columns equally divide the available space. For example:

```
<frameset rows="*,100,*">
```

creates a 100-pixel tall row in the middle of the frameset and equal-sized rows above and below it.

If you precede the asterisk with an integer value, the corresponding row or column gets proportionally more of the available space. For example:

```
<frameset cols="10%,3*,*,*">
```

creates four columns: the first column occupies 10 percent of the overall width of the frameset. The browser then gives the second frame 3/5 of the remaining space, and the third and the fourth are each given 1/5 of the remaining space.

Using asterisks (especially with the numeric prefix) makes it easy to divide up the remaining space in a frameset.

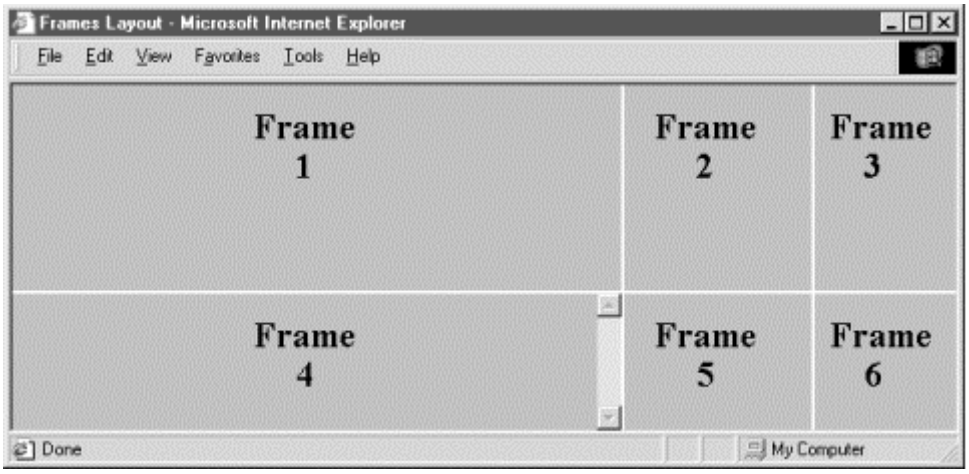
Be aware, too, that unless you explicitly tell it not to, the browser lets users manually resize the individual frame document's columns and rows, and hence change the relative proportions each frame occupies in their frames display. To prevent this, see the `noresize` attribute for the `<frame>` tag.

11.3.1.2 Controlling frame borders and spacing

The popular browsers provide attribute extensions that you may use to generally define and change the borders surrounding the frames in a frameset. The HTML 4 and XHTML standards prefer instead that you include these border-related display features via `<style>` tag attributes.

Both Internet Explorer and Netscape accept the `frameborder` attribute to disable or explicitly enable frame borders. (By default, every frame in a frameset as well as the frameset window itself is rendered with a 3D border; see Figure 11-1.) The two browsers' documentation disagree about the particular values for the `frameborder` attribute, but both acknowledge the other's conventions. Hence, setting the value of `frameborder` to `0` or `no` turns borders off (see Figure 11-2); `1` or `yes` turns borders on.

Figure 11-2. The frameborder attribute lets you remove the borders between frames

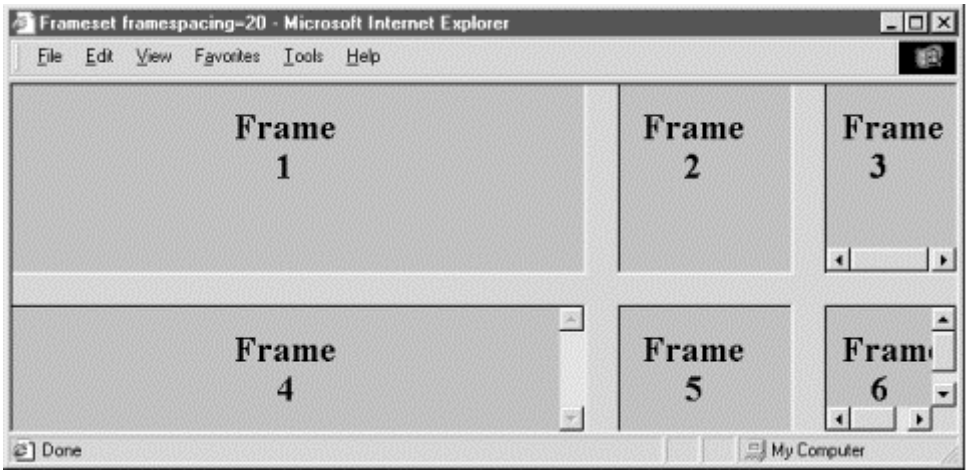


Internet Explorer and Netscape do disagree, however, as to how you may control the thickness of the borders. The latest version of Internet Explorer (Version 5) supports both the `framespacing` and `border` attributes, whose value is the number of pixels you want between frames (see Figure 11-3).

These attributes affect all frames and framesets nested within the current frameset as displayed by Internet Explorer. In practice, you should set it once on the outermost `<frameset>` to create a consistent border appearance for all of the frames in a single page.

Netscape accepts only the `border` attribute to define the border width, with an integer value in pixels. Like Internet Explorer, Netscape lets you include the `frameborder` attribute with any `<frameset>` tag, affecting all nested frames and framesets. Unlike Internet Explorer, Netscape lets you include the `border` attribute only in the outermost `<frameset>`, ensuring that all frame borders are the same width within that `<frameset>`.

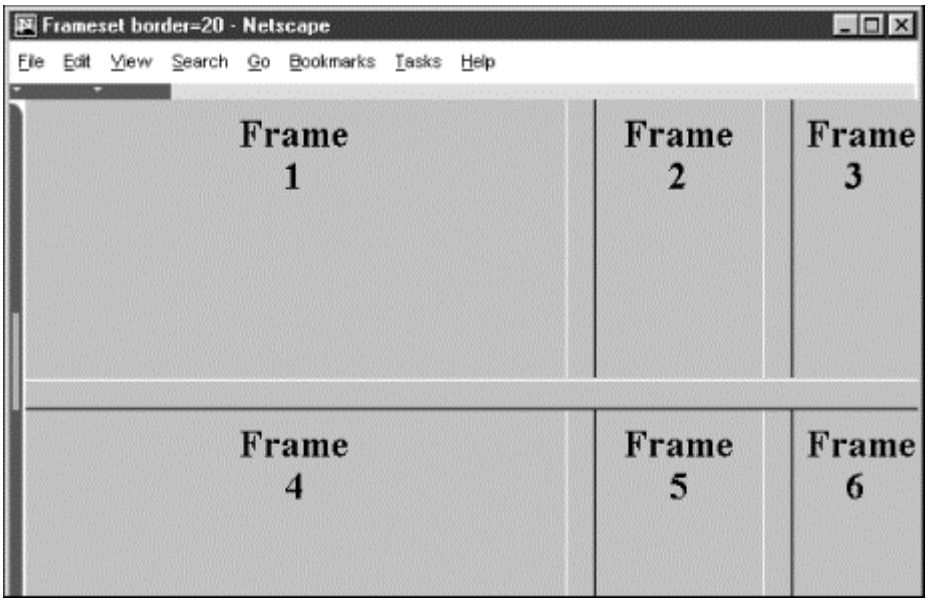
Figure 11-3. Internet Explorer takes the framespacing attribute to separate frames



Since browsers ignore unsupported attributes, it is possible to define frame borders so that both browsers do the right thing. Just make sure to use the same `framespacing` and `border` values.

Finally, both Netscape and the latest version of Internet Explorer (Version 5) let you control the color of the frame borders using the `bordercolor` attribute (Figure 11-4). It accepts a color name or hexadecimal triple as its value. A complete list of color names and values can be found in Appendix G.

Figure 11-4. Netscape accepts border and bordercolor attributes to control the color and spacing between frames



11.3.1.3 Frames and JavaScript

Internet Explorer and Netscape, as standardized in HTML 4 and XHTML, support JavaScript-related event handlers that let your frame documents react when they are first loaded and when the frame window gets resized (`onLoad`); unloaded from the browser by the user (`onUnload`); when the window containing the frameset loses focus, such as when the user selects another window (`onBlur`); or when the frameset becomes the active window (`onFocus`). Included as `<frameset>` attributes, these event handlers take quote-enclosed lists of JavaScript commands and function calls as their value. For example, you might notify the user when all the contents have been loaded into their respective frames of a lengthy frameset:

```
<frameset onLoad="window.alert('Everything is loaded. You may now continue.')">
```

These four attributes may also be used with the `<body>` tag. We cover JavaScript event handlers in more detail in Section 12.3.3.

11.3.1.4 Other <frameset> attributes

Like most of the other standard tags, the `<frameset>` tag honors four of the standard attributes: `class`, `style`, `title`, and `id`.

Use the `class` attribute to associate a predefined style class with this frame and, via style inheritance, its content. Alternatively, use the `style` attribute to define a style inline with the `<frameset>` tag. We cover styles more completely in Chapter 8.

The `id` attribute creates a unique identifier for the frame, while `title` creates a title for the frame that might be presented to the user or used by a nonvisual browser. Section 4.1.1.4 / Section 4.1.1.5

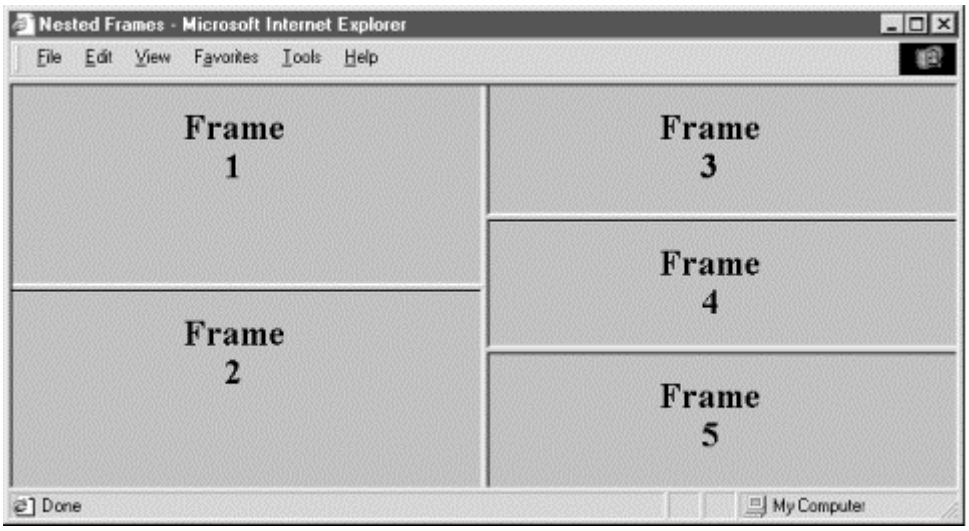
11.3.2 Nesting <frameset> Tags

You can create some elaborate browser displays with a single `<frameset>`, but the frame layout is unimaginative. Instead, create staggered frames and other more complex layouts with multiple `<frameset>` tags nested within a top-level `<frameset>` in the frame document (Figure 11-5).

For example, create a layout of two columns, the first with two rows and the second with three rows, by nesting two `<frameset>` tags with row specifications within a top-level `<frameset>` that specifies the columns:

```
<frameset cols="50%,*">
  <frameset rows="50%,*">
    <frame src="frame1.html">
    <frame src="frame2.html">
  </frameset>
  <frameset rows ="33%,33%,*">
    <frame src="frame3.html">
    <frame src="frame4.html">
    <frame src="frame5.html">
  </frameset>
</frameset>
```

Figure 11-5. Staggered frame layouts use nested <frameset> tags



11.4 Frame Contents

A frame document contains no displayable content, except perhaps a message for non-frame-enabled browsers. Instead, `<frame>` tags inside the one or more `<frameset>` tags (which encapsulate the contents of a frame document) provide URL references to the individual documents that are occupying each frame. Section 11.5

11.4.1 The <frame> Tag

The `<frame>` tag appears only within a `<frameset>`. Use it to set, via its associated `src` attribute, the URL of the document content that initially gets displayed inside the respective frame.

Frames are placed into a frameset column by column, from left to right, and then row by row, from top to bottom, so the sequence and number of `<frame>` tags inside the `<frameset>` tag are important.

The browser displays empty frames for `<frame>` tags that do not have a `src` attribute. It also displays empty frames if the <frameset> tag calls for more frames than the corresponding <frame> tags define. Such orphans remain empty; you cannot put content into them later, even if they have a target "name" for display redirection. Section 11.4.1.2

11.4.1.1 The src attribute

The value of the `src` attribute for the `<frame>` tag is a URL of the document that is to be displayed in the frame. There is no other way to provide content for a frame. You shouldn't, for instance, include any `<body>` content within the frame document; the browser will ignore the frame tags and display just the contents of a `<body>` tag if it comes first, or vice versa.

The document referenced by the `src` attribute may be any valid document or any displayable object, including images and multimedia. In particular, the referenced document may itself be composed of one or more frames. The frames are displayed within the referencing frame, providing yet another way of achieving complex layouts using nested frames.

Since the source may be a complete document, all the features of HTML/XHTML apply within a frame, including backgrounds and colors, tables, fonts, and the like. Unfortunately, this also means that multiple frames in a single browser window may conflict with each other. Specifically, if each nested frame document (not a regular HTML or XHTML document) has a different `<title>` tag, the title of the overall browser window will be the title of the most recently loaded frame document. The easiest way to avoid this problem is to ensure that all related frame documents use the same title.

11.4.1.2 The name attribute

The optional `name` attribute for the `<frame>` tag labels that frame for later reference by the `target` attribute for the hypertext link anchor `<a>` tag and the `<form>` tag. This way, you can alter the contents of a frame using a link in another frame. Otherwise, like normal browser windows, hypertext-linked documents replace the contents of the source frame. We discuss names and targets at greater length later in this chapter. Section 11.7.1

The value of the `name` attribute is a text string enclosed in quotation marks.

<frame>

Function:

Define a single frame in a <frameset>

Attributes:

BORDERCOLOR NO	NAME
CLASS	NORESIZE
FRAMEBORDER	SCROLLING
ID	SRC
LONGDESC	STYLE
MARGINHEIGHT	TITLE
MARGINWIDTH	

End tag:

</frame>; rarely included in HTML

Contains:

Nothing

Used in:

frameset_content

11.4.1.3 The noresize attribute

Even though you may explicitly set frame dimensions with attributes in the `<frameset>` tag, users can manually alter the size of a column or row of frames. To suppress this behavior, add the `noresize` attribute to the frame tags in the row or column whose relative dimensions you do not want users fiddling with. For a two-by-two frame document, a `noresize` attribute in any one of the four associated frame tags will effectively freeze the relative proportions of all the frames, for example.

The `noresize` attribute is especially useful for frames that contain fixed images serving as advertisements, a button bar, or a logo. By fixing the size of the frame to contain just the image and setting the `noresize` attribute, you guarantee that the image will be displayed in the intended manner and that the remainder of the browser window will always be given over to the other frames in the document.

11.4.1.4 The scrolling attribute

The browser will display vertical and horizontal scrollbars with frames whose contents are larger than the allotted window space. If there is sufficient room for the content, the scrollbars disappear. The `scrolling` attribute for the `<frame>` tag gives you explicit control over whether the scroll bars appear or disappear.

With `scrolling="yes"`, the browser adds scroll bars to the designated frame even if there is nothing to scroll. If you set the `scrolling` attribute value to `no`, scrollbars will never be added to the frame, even if the frame contents are larger than the frame itself. The value `auto`, supported only by Netscape, works as if you didn't include the `scrolling` attribute in the tag; Netscape adds scrollbars as needed. To achieve `auto` behavior in Internet Explorer, simply omit the `scrolling` attribute altogether.

11.4.1.5 The marginheight and marginwidth attributes

The browser normally places a small amount of space between the edge of a frame and its contents. You can change those margins with the `marginheight` and `marginwidth` attributes, each including a value for the exact number of pixels to place around the frame contents.

You cannot make a margin less than one pixel, or make it so large there is no room for the frame contents. That's because like most other HTML attributes, these advise: they do not dictate to the browser. If your desired margin values cannot be accommodated, the browser ignores them and renders the frame as best it can.

11.4.1.6 The frameborder and bordercolor attributes

You can add or remove borders from a single frame with the `frameborder` attribute. Values of `yes` or `1` and `no` or `0` respectively enable or disable borders for the frame and override the value of the `frameborder` attribute for any frameset containing the frame.

Note that the browsers do react somewhat differently to border specifications. Netscape, for instance, removes an individual border only if adjacent frames sharing that border have borders turned off. Internet Explorer, on the other hand, will remove those adjacent borders, but only if they are not explicitly turned on in those adjacent frames. Our advice is to explicitly control the borders for each frame if you want to consistently control the borders for all frames across both browsers.

With the popular browsers, you also can change the color of the individual frame's borders with the `bordercolor` attribute. Use a color name or hexadecimal triple as its value. If two adjacent frames have different `bordercolor` attributes, the resulting border color is undefined. A complete list of color names and values can be found in Appendix G.

11.4.1.7 The title and longdesc attributes

Like most other standard tags, you can provide a title for a frame with the `title` attribute. The value of the attribute is a quote-enclosed string that describes the contents of the frame. Browsers might display the title, for instance, when the mouse passes over the frame.

If the `title` attribute isn't quite enough for you, the `longdesc` attribute can be used. Its value is the URL of a document that describes the frame. Presumably, this long description might be in some alternative media, suitable for use by a nonvisual browser.

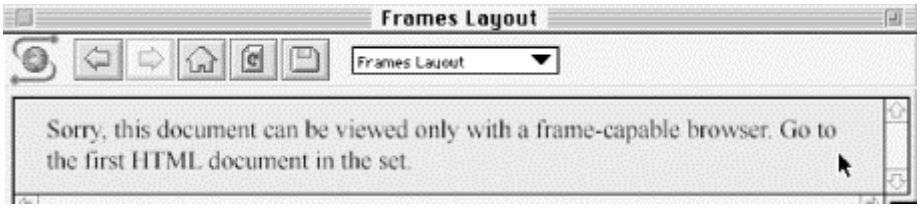
11.5 The <noframes> Tag

A frame document has no `<body>`. It must not, since the browser will ignore any frame tags if it finds any `<body>` content before it encounters the first `<frameset>` tag. A frame document, therefore, is all but invisible to any non-frame-capable browser. The `<noframes>` tag gives some relief to the frame-disabled.

You should use the `<noframes>` tag only within the outermost `<frameset>` tag of a frame document. The content inside the `<noframes>` tag and its required end tag (`</noframes>`) is not displayed by any frame-capable browser, but is displayed in lieu of other contents in the frame document by browsers that do not handle frames. The contents of the `<noframes>` tag can be any normal body content, including the `<body>` tag itself.

Although this tag is optional, experienced authors typically include the `<noframes>` tag in their frame documents with content that warns a non-frame-capable browser user that they're missing the show. And smart authors will give those users a way out, if not direct access to the individual documents that make up the frame document contents. Remember our first frame example in this chapter? Figure 11-6 shows what happens when that frame document gets loaded into an old version of Mosaic.

Figure 11-6. A <noframes> message in a non-frame-capable browser



<noframes>

Function:

Supply content for non-frame-capable browsers

Attributes:

CLASS	ONKEYUP
DIR	ONMOUSEDOWN
ID	ONMOUSEMOVE
LANG	ONMOUSEOUT
ONCLICK	ONMOUSEOVER
ONDBLCLICK	ONMOUSEUP
ONKEYDOWN	STYLE
ONKEYPRESS	TITLE

End tag:

</noframes>; sometimes omitted in HTML

Contains:

body_content

Used in:

frameset_content

```
<noframes>
  Sorry, this document can be viewed only with a
  frame-capable browser. Go to the <a href="frame1.html">
 first HTML document</a> in the set.
</noframes>
```

The reason `<noframes>` works is that most browsers are extremely tolerant of erroneous tags and incorrect documents. A nonframe browser simply ignores the frame tags. What's left, then, is the content of the `<noframes>` tag, which the browser dutifully displays.

If your browser strictly enforces some version of HTML or XHTML that does not support frames, it may simply display an error message and refuse to display the document, even if it contains a `<noframes>` tag.

11.5.1 <noframes> Attributes

There are no attributes specific to the `<noframes>` tag, but you can use any of the sixteen standard attributes: `class` and `style` for style management, `lang` and `dir` for language type and display direction, `title` and `id` for titling and naming the enclosed content, and any of the event attributes for user-activated JavaScript processing within the `<noframes>` tag. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.4 / Section 4.1.1.5 / Section 8.1.1 / Section 8.3 / Section 12.3.3

11.6 Inline Frames

To this point, our discussion has centered around frames that are defined as part of a frameset. A frameset, in turn, replaces the conventional <body> of a document and supplies content to the user via its contained frames.

The HTML 4 and XHTML standards let you do things a bit differently: you can also define a frame that exists within a conventional document, displayed as part of that document's text flow. These frames behave a lot like inline images, which is why they are known as inline frames.

All Internet Explorer Versions 4 and later, but only the latest version of Netscape Navigator (Version 6), support inline frames.

11.6.1 The <iframe> Tag

Define an inline frame with the <iframe> tag. The <iframe> tag is *not* used within a <frameset> tag. Instead, it appears anywhere in your document that an <img> tag might appear. The <iframe> tag defines a rectangular region within the document in which the browser displays a separate document, including scrollbars and borders.

<iframe>

Function:

Define an inline frame within a text flow

Attributes:

ALIGN	MARGINWIDTH
CLASS	NAME
FRAMEBORDER	SCROLLING
HEIGHT	SRC
ID	STYLE
LONGDESC	TITLE
MARGINHEIGHT	WIDTH

End tag:

</iframe>; never omitted

Contains:

body_content

Used in:

text

Use the src attribute with <iframe> to specify the URL of the document that occupies the inline frame. All of the other, optional attributes for the <iframe> tag, including class, frameborder, id, longdesc, marginheight, marginwidth, name, scrolling, style, and title, behave exactly like the corresponding attributes for the <frame> tag. Section 11.4.1

Use the content of the <iframe> tag to provide information to users of browsers that do not support inline frames. Compliant browsers will ignore these contents whereas all other browsers ignore the <iframe> tag and therefore display its contents as if it were regular body content.

For instance, use the <iframe> content to explain to users what they are missing:

```
...other document content
<iframe src="sidebar.html" width=75 height=200 align=right>
Your browser does not support inline frames.  To view this
<a href="sidebar.html">document</a> correctly, you'll need
a copy of Internet Explorer or the latest Netscape Navigator.
</iframe>
...subsequent document content
```

In this example, we let the user know that they were accessing an unsupported feature and provided a link to the missing content.

11.6.1.1 The align attribute

Like the align attribute for the <table> tag, this inline frame attribute lets you control where the frame gets placed inline with the adjacent text or moved to the edge of the document, allowing text to flow around the frame.

For inline alignment, use top, middle, or bottom as the value of this attribute. The frame will be aligned with the top, middle, or bottom of the adjacent text, respectively.

To allow text to flow around the inline frame, use the left or right values for this attribute. The frame will be moved to the left or right edge of the text flow, respectively, and the remaining content of the document will be flowed around the frame. A value of center places the inline frame in the middle of the display, with text flowing above and below.

11.6.1.2 The height and width attributes

Internet Explorer and Netscape 6 put the contents of an inline frame into a predefined, 150-pixel-tall, 300-pixel-wide box. Use the height and width attributes with values as the number of pixels to change those dimensions.

11.6.2 Using Inline Frames

Although you'll probably shy away from them for most of your web pages (at least until most browsers become fully standards-compliant), inline frames can be useful, particulary for providing information related to the current document being viewed, similar to the sidebar articles you find in a conventional printed publication.

Except for their location within conventional document content, inline frames are treated exactly like regular frames. You can load other documents into the inline frame using its name (see following section) and link to other documents from within the inline frame.

11.7 Named Frame or Window Targets

As we discussed in the <frame> tag description section, you can label a frame by adding the name attribute to its <frame> tag. The id attribute provides the same unique labeling. Once named or identified, the frame may become the destination display window for a hypertext-linked document selected within a document displayed in some other frame. You accomplish this redirection by adding the special target attribute to the anchor that references the document.

11.7.1 The target Attribute for the <a> Tag

If you include a target attribute within an <a> tag, the browser will load and display the document named in the tag's href attribute in a frame or window whose name matches the target. If the named or id'd frame or window doesn't exist, the browser will open a new window, give it the specified label, and load the new document into that window. Thereafter, hypertext-linked documents can target the new window.

Targeted hypertext links make it easy to create effective navigational tools. A simple table of contents document, for example, might redirect documents into a separate window:

```
<h3>Table of Contents</h3>
<ul>
  <li><a href="pref.html" target="view_window">Preface</a>
  <li><a href="chap1.html" target="view_window">Chapter 1</a>
  <li><a href="chap2.html" target="view_window">Chapter 2</a>
  <li><a href="chap3.html" target="view_window">Chapter 3</a>
</ul>
```

The first time the user selects one of the table of contents hypertext links, the browser will open a new window, label it "view_window," and display the desired document's contents inside it. If the user selects another link from the table of contents and the "view_window" is still open, the browser will again load the selected document into that window, replacing the previous document.

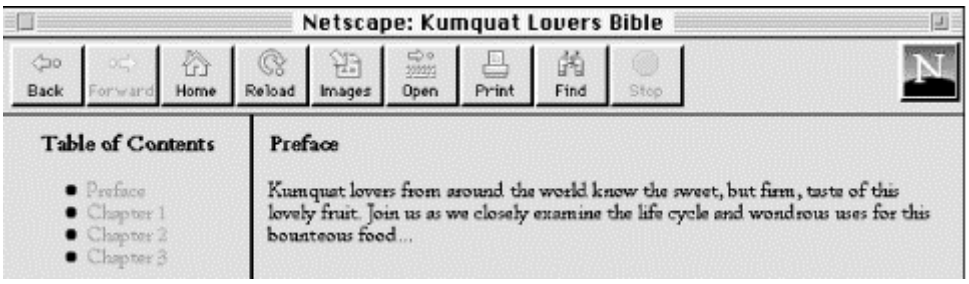
Throughout the whole process, the window containing the table of contents is accessible to the user. By clicking on a link in one window, the user causes the contents of the other window to change.

Rather than open an entirely new browser window, a more common use of `target` is to direct hyperlink contents to one or more frames in a `<frameset>` display. You might place the table of contents into one frame of a two-frame document and use the adjacent frame for display of the selected documents:

```
<frameset cols="150,*">
  <frame src="toc.html">
  <frame src="pref.html" id="view_frame">
</frameset>
```

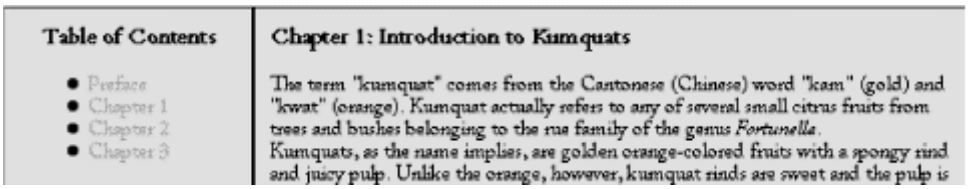
When the browser initially displays the two frames, the left frame contains the table of contents, and the right frame contains the preface (see Figure 11-7).

Figure 11-7. Table of contents frame controls content of adjacent frame



When a user selects a link from the table of contents in the left frame (for example, Chapter 1), the browser loads and displays the associated document into the "view_frame" frame on the right side (Figure 11-8). As other links are selected, the right frame's contents change, while the left frame continuously makes the table of contents available to the user.

Figure 11-8. The contents of Chapter 1 are displayed in the adjacent frame



11.7.2 Special Targets

There are four reserved target names for special document redirection actions:

`_blank`

The browser always loads a `target="_blank"` linked document into a newly opened, unnamed window.

`_self`

This target value is the default for all `<a>` tags that do not specify a target, causing the target document to be loaded and displayed in the same frame or window as the source document. This target is redundant and unnecessary unless used in combination with the `target` attribute in the `<base>` tag in a document's head (see Section 11.7.3).

`_parent`

The `_parent` target causes the document to be loaded into the parent window or frameset containing the frame containing the hypertext reference. If the reference is in a window or top-level frame, then it is equivalent to the target `_self`.

A brief example may help clarify how this link works. Consider a link in a frame that is part of a three-column frameset. This frameset, in turn, is a row in the top-level frameset being displayed in the browser window. This arrangement is shown in Figure 11-9.

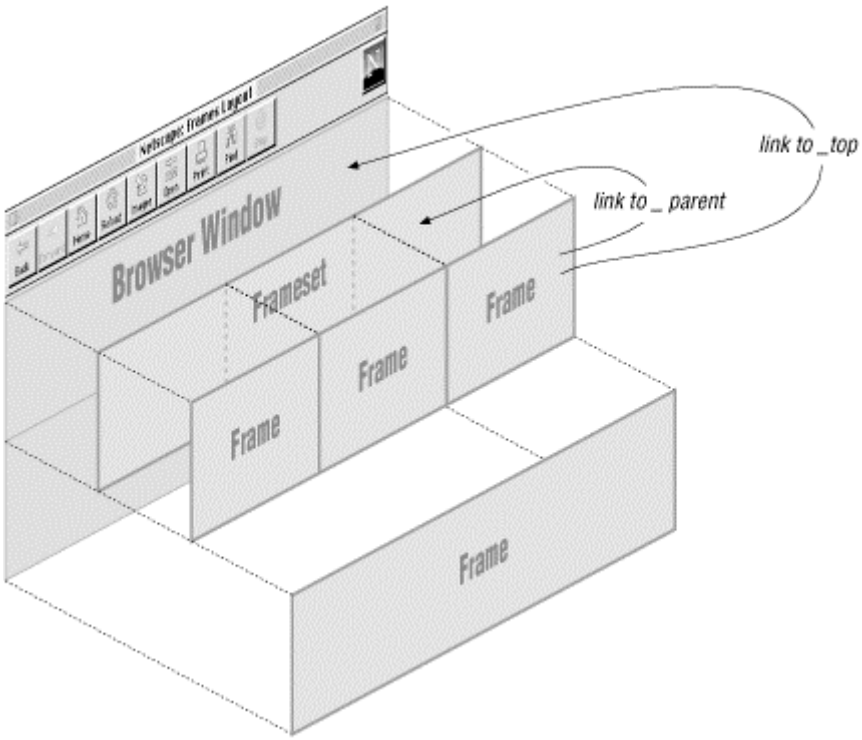
If no target is specified for the hypertext link, then it is loaded into the containing frame. If a target of `_parent` is specified, the document is loaded into the area occupied by the three-column frameset containing the frame that contains the link.

`_top`

This target causes the document to be loaded into the window containing the hypertext link, replacing any frames currently displayed in the window.

Continuing with the frame hierarchy, as shown in Figure 11-9, using a target of `_top` would remove all the contained frames and load the document into the entire browser window.

Figure 11-9. Using special hypertext targets in nested frames and framesets



All four of these `target` values begin with the underscore character. Any other window or target beginning with an underscore is ignored by the browser, so don't use the underscore as the first character of any frame `name` or `id` you define in your documents.

11.7.3 The <base> Default Target

It can be tedious to specify a target for every hypertext link in your documents, especially when most are targeted at the same window or frame. To alleviate this problem, you can add a `target` attribute to the `<base>` tag. Section 6.7.1

The `target` attribute in the `<base>` tag sets the default target for every hypertext link in the current document that does not contain an explicit `target` attribute. For example, in our example table of contents document, almost every link causes the document to be displayed in another window named "view_frame." Rather than include that target in each hypertext link, you should place the common target in the table of contents' `<base>` tag within its `<head>`:

```
<head>
<title>Table of Contents</title>
<base target="view_frame">
</head>
<body>
<h3>Table of Contents</h3>
<ul>
  <li><a href="pref.html">Preface</a></li>
  <li><a href="chap1.html">Chapter 1</a></li>
  <li><a href="chap2.html">Chapter 2</a></li>
  <li><a href="chap3.html">Chapter 3</a></li>
</ul>
</body>
```

Notice that we don't include any other target references in the list of hyperlinks, because the browser will load and display all the respective documents in the base target "view_frame."

11.7.4 Traditional Link Behavior

Before the onset of frames, each time you selected a hyperlink, the corresponding document replaced the contents of the browser window. With frames, this behavior is modified so that the corresponding document replaces the content of the referencing frame. This is often not the desired behavior and can be disconcerting to people browsing your documents.

For example, suppose you have arranged all of the documents on your site to present themselves in three frames: a navigational frame at the top of the browser window, a scrolling content frame in the middle, and a feedback form at the bottom. You named the content frame with the `name` attribute of the `<frame>` tag in the top-level document for your collection and used the `target` attribute of the `<base>` tag in every document on your site to ensure that all links will be loaded into the center content frame.

This arrangement works perfectly for all the documents on your site, but what happens when a user selects a link that takes them to a different site? The referenced document will still be loaded into the center content frame. Now the user is confronted by a document from some other site, surrounded by your navigation and feedback frames![1] Very impolite.

> [1] Check out Chapter 17 on how to step out into the forefront when your pages happen to be on the other end of a targetless hyperlink.

The solution is to make sure that every hypertext link that references a remote document has a target of `_top`. This way, when someone selects a link that takes them away from your site, the remote document will replace the contents of the entire browser window, including your navigation and feedback frames. If the majority of the links in your documents are to other sites, you might consider adding `target="_top"` to a `<base>` tag in your document and using explicit `target` attributes in the links to your local documents.

Chapter 12. Executable Content

One of the most exciting recent developments in web technologies is the ability to deliver applications directly to the user's browser. These typically small programs, which are known as *applets*, perform simple tasks on the client computer, from responding to user mouse- or keyboard-actions to spicing up your web page displays with multimedia-enabling software.

You may embed applets into your documents using a special programming language known as JavaScript. Or you can load and execute Java-based, platform-independent applets over the Internet. During execution, these programs may generate dynamic content, interact with the user, validate form data, or even create windows and run entire applications independent of your pages. The possibilities are endless, and they go far beyond the document model that was originally envisioned for HTML.

In this chapter, we show you, with simple examples, how to embed and include executable content - scripts and applets - in your documents. We won't, however, even begin to pretend to teach you how to write and debug your own applets. This is a book about HTML and XTHML, after all. Rather, get an expert opinion: turn to any of the many excellent texts from O'Reilly, including *JavaScript: The Definitive Guide*, by David Flanagan, *Java in a Nutshell*, by David Flanagan, and *Exploring Java,* Pat Niemeyer and Josh Peck.

12.1 Applets and Objects

Applets, like client-side image maps, represent a shift in the basic model of web communications. Until recently, servers performed most of the computational work on the Web, client browsers being not much more than glorified terminals. With applets, web technology is shifting toward the client, distributing some or all of the computational load from the server to the client and its browser.

Applets also represent a way of extending a browser's features without forcing users to purchase or otherwise acquire a new browser, as is the case when developers implement new tag and attribute extensions to HTML. Nor do users have to acquire and install a special application, as is required for helper or plug-in applications. This means that once users have a browser that supports applets (Netscape and Internet Explorer do), you can deliver applets immediately, including display and multimedia innovations.

12.1.1 The Object Model

Java-based applets - web page-referenced programs retrieved from a network server and executed on the user's client computer - actually are a subset of what the HTML 4 and XHTML standards call *inclusions*. As with images, the browser first loads the HTML document, then examines it for inclusions - additional, separate, and discrete content that is to be handled by the client browser. A GIF image is one type of inclusion. A *.wav* sound file is another; an MPEG movie is another; a Java-based clock program is another.

The HTML 4 and XHTML standards generally calls the inclusion contents *objects*. In fact, in your document you may identify and load nearly any object file over the network through a universal `<object>` tag, which we discuss in detail later in this chapter. Section 12.2.1

Once downloaded, the standards dictate that the browser somehow render the object, by internal or external mechanisms. The popular graphical browsers, for instance, have integrated software for displaying GIF and JPEG images. Otherwise, plug-ins and other helper applications may provide the necessary rendering mechanism.

12.1.1.1 The applet model

With applet-based objects, the browser sets aside a portion of the document display space. You may control the size and position of this display area; the applet controls what is presented inside.

The applet object is software, an executable program. Accordingly, besides providing a display space, the browser, in tandem with the client computer environment and resources, provides the applet runtime environment - typically Java.

During execution, the applet has access to a restricted environment within the user's computer. For instance, applets have access to the mouse and keyboard and may receive input from the user. They can initiate network connections and retrieve data from other servers on the Internet. In sum, applets are full-fledged programs, complete with a variety of input and output mechanisms, along with a full suite of network services.

Several applets may be placed in a single document; they all execute in parallel and may communicate with each other. While the browser may limit their access to its computer system, applets have complete control of their virtual environment within the browser.

12.1.1.2 The applet advantage

There are several advantages of applets, not the least of which is providing more compelling user interfaces within a web page. For instance, an applet might create a unique set of menus, choices, text fields, and similar user-input tools different from those available through the browser. When the user clicks a button within the applet's interaction/display region, the applet might respond by displaying results within the region, signaling another applet, or even by loading a completely new page into the browser.

We don't mean to imply that the only use of applets is to enhance the user interface. An applet is a full-fledged program that can perform any number of computational and user-interactive tasks on the client computer. An applet might implement a real-time video display, perform circuit simulation, engage the user in a game, provide a chat interface, and so on.

12.1.1.3 Using applets correctly

An applet is nothing more than another tool you may use to produce compelling and useful web pages. Keep in mind that an applet uses computational resources on the client to run and therefore places a load on the user's computer. It can degrade system performance.

Similarly, if an applet uses a lot of network bandwidth to accomplish its task (a real-time video feed, for example), it may make other network communication unbearably slow. While such applications are fun, they do little more than annoy your target audience.

To use an applet correctly, balance the load between the browser and the server. For each page, decide which tasks are best left to the server (forms processing, index searches, and the like) and which tasks are better suited for local processing (user interface enhancements, real-time data presentation, small animations, input validation, and so on). Divide the processing accordingly. Remember that many users have slower network connections and computers than you do and design your applets to satisfy the majority of your audience.

Used the right way, applets seamlessly enhance your pages and provide a satisfying experience for your audience. Used improperly, applets are just another annoying bandwidth waster, alienating your users and hurting your pages.

12.1.1.4 Writing applets

Creating applets is a programming task, not usually a job for the HTML or XHTML author, and certainly well beyond the scope of this book. For details, we recommend you consult any of the many applet programming texts that have recently appeared on bookshelves everywhere, including those from O'Reilly & Associates.

Today, one language dominates the applet programming world: Java. Developed by Sun Microsystems of Mountain View, California, Java supports an object-oriented programming style wherein classes of applets can be used and reused to build complex applications.

By invention, applets built from the same language should run with any browser that supports them. In reality, certain Microsoft implementation decisions had caused some valid Java applets to fail when running on earlier versions of Internet Explorer. Microsoft has fixed these problems with Internet Explorer Version 5 and Java will remain a universal programming language for the Web. In any case, the conscientious Java programmer should keep abreast of the latest technology and create applets that are certifiably 100% pure Java. Microsoft, in particular, is trying to get programmers to use proprietary extensions to Java that will work on only Microsoft platforms and is refusing to support key parts of the standard. We recommend avoiding any vendor extensions to Java that deviate from the standard Java 1.1 version.

You can shield yourself from platform dependencies by using the Java Plugin from Sun; see http://java.sun.com/products/plugin. There are versions for both Internet Explorer and Netscape Navigator. Currently, the plugin is the only way to get support for the latest version of Java (Java 2 SDK 1.3).[1]

> [1] The plugin has achieved some acceptance for running Java 2 applets in intranet (i.e., corporate network) environments, but we have yet to see an applet on the public Internet that required Java 2.

We should take this opportunity to also mention ActiveX, an alternative applet programming technology available from Microsoft. ActiveX is proprietary, closely coupled with various versions of Microsoft Windows, and works only when used with Internet Explorer. ActiveX applets will run on versions of Internet Explorer targeted to various versions of Windows, but a single ActiveX applet will not run on these different versions without recompilation. This is in contrast with Java applets, where a single Java applet can be written and compiled once and immediately run on a broad range of browsers and operating systems.

ActiveX also presents an unacceptably high security risk to any user whose browser supports ActiveX technology. It is ridiculously easy to penetrate and damage a computer running a browser that allows ActiveX applets to be executed. For this reason, we cannot recommend ActiveX as a viable applet implementation technology and we go so far as to recommend that users disable ActiveX capability within their browsers - specifically, Internet Explorer.

12.2 Embedded Content

In this section, we cover three tags that support embedded content. The `<object>` tag is in the HTML 4 and XHTML standards. It is a generalized hybrid of the deprecated `<applet>` tag for embedding applets, particularly Java applets, and the `<embed>` tag extension that lets you include an object whose MIME type references the plug-in needed to process and possibly display that object.

The latest standards strongly encourage you to use the `<object>` tag to include applets as well as a variety of discrete inclusions in your documents, even images (although the standards do not go so far as to deprecate the `<img>` tag). Use `<object>` with the `classid` attribute to insert Java and other applets into a document, along with their execution parameters as contents of the associated `<param>` tag. And with `<object>`, use the `data` attribute to download and display non-HTML/XHTML content, such as multimedia, in the user's computing environment. Object data may be processed and rendered by an included applet, by utilities that come with your browser, or by a plug-in ("helper") application that the user supplies.

For applets, the browser creates a display region in the containing text flow exactly like an inline image: without line breaks and as a single large entity. The browser then downloads and executes the applet's program code, if specified, and downloads and renders any included data just after download and display of the document. Execution of the applet continues until the code terminates itself or when the user stops viewing the page containing the applet.

With data, the browser decodes the object's data type and will either handle its rendering directly, such as with GIF and JPEG images, or invoke an associated plug-in application for the job.

12.2.1 The <object> Tag

The `<object>` tag was originally implemented by Microsoft to support its ActiveX applets and only later added Java support. In a similar manner, Netscape initially supported the alternative `<embed>` and `<applet>` tags for inclusion objects and later provided limited support for the `<object>` tag.

All that jostling for position by the browser giants made us nervous, and we were hesitant in previous editions of this book to even suggest that you use `<object>` at all. We now heartily endorse it, based on the strength of the HTML 4 and particularly the XHTML standards. Although not fully supported yet, expect `<object>` to be well supported soon by the popular browsers and expect the alternative `<embed>` and `<applet>` tags to be less supported, if not completely ignored, by future HTML 4/XHML-compliant browsers.

The contents of the `<object>` tag may be any valid HTML or XHTML content, along with `<param>` tags that pass parameters to an applet. If the browser can retrieve the requested object and successfully process it, either by executing the applet or by processing the object's data with a plug-in (helper) application, the contents of the `<object>` tag, except for the `<param>` tags, are ignored. If any problem occurs during the retrieval and processing of the object, the browser won't insert the object into the document, but instead will display the contents of the `<object>` tag, except for the `<param>` tags. In short, you should provide alternative content for browsers that cannot handle the `<object>` tag or if the object cannot be successfully loaded.

12.2.1.1 The classid attribute

Use the `classid` attribute to specify the location of the object, typically a Java class, that you want included by the browser. The value may be an absolute or relative URL of the desired object. Relative URLs are considered to be relative to the URL specified by the `codebase` attribute if it is provided; otherwise they are relative to the current document URL.

For example, to execute a clock Java applet contained in a file named *clock.class*, you might include in your HTML document the code:

```
<object classid="clock.class">
</object>
```

The browser will locate the code for the applet using the current document's base URL. Hence, if the current document's URL is:

```
http://www.kumquat.com/harvest_time.html
```

the browser will retrieve the applet code for our `clock` class example as:

```
http://www.kumquat.com/clock.class
```

<object>

Function:

Embed an object or applet in a document

Attributes:

ALIGN	ONKEYDOWN
ARCHIVE	ONKEYPRESS
BORDER	ONKEYUP
CLASS	ONMOUSEDOWN
CLASSID	ONMOUSEMOVE
CODEBASE	ONMOUSEOUT
CODETYPE	ONMOUSEOVER
DATA	ONMOUSEUP
DECLARE	SHAPES I
DIR	STANDBY
HEIGHT	STYLE
HSPACE	TABINDEX
ID	TITLE
LANG	TYPE
NAME	USEMAP
NOTAB I	VSPACE
ONCLICK	WIDTH
ONDBLCLICK	

End tag:

</object>; never omitted

Contains:

object_content

Used in:

text

12.2.1.2 The codebase attribute

Use the codebase attribute to provide an alternative base URL from which the browser will retrieve an object. The value of this attribute is a URL pointing to a directory containing the object referenced by the classid attribute. The codebase URL overrides, but does not permanently replace, the document's base URL, which is the default if you don't use codebase. Section 6.2

Now, continuing with our previous examples, suppose your document comes from *http://www.kumquat.com*, but the clock applet is kept in a separate directory named classes. You cannot retrieve the applet by specifying classid="classes/clock.class". Rather, include the codebase attribute and new base URL:

```
<object classid="clock.class" codebase="http://www.kumquat.com/classes/">
</object>
```

which resolves to the URL:

```
http://www.kumquat.com/classes/clock.class
```

Although we used an absolute URL in this example, you also can use a relative URL. For instance, applets typically get stored on the same server as the host documents, so we'd usually be better off, for relocation's sake, specifying a relative URL for the codebase, such as:

```
<object code="clock.class" codebase="/classes/">
</object>
```

The classid attribute is similar to the code attribute of the <applet> tag, providing the name of the file containing the object and is used in conjunction with the codebase attribute to determine the full URL of the object to be retrieved and placed in the document.

12.2.1.3 The archive attribute

For performance reasons, you may choose to preload collections of objects contained in one or more archives. This is particularly true of Java-based applications, where one Java class will rely on many other classes to get its work done. The value of the archive attribute is a quote-enclosed list of URLs, each pointing to an archive to be loaded by the browser before rendering or executing the object.

12.2.1.4 The codetype attribute

The codetype attribute is required only if the browser cannot determine an applet's MIME type from the classid attribute or if the server does not deliver the correct MIME type when downloading an object. This attribute is nearly identical to type (Section 12.2.1.6), except that it is used to identify program code type whereas type should be used to identify data file types.

The following example explicitly tells the browser that the object's code is Java:

```
<object code="clock.class" codetype="application/java">
</object>
```

12.2.1.5 The data attribute

Use the data attribute to specify the data file, if any, that is to be processed by the object. The data attribute's value is the URL of the file, either absolute or relative to the document's base URL or to that which you provide with the codebase attribute. The browser determines the data type by the type of object that is being inserted in the document.

This attribute is similar to the src attribute of the <img> tag in that it downloads data to be processed by the included object. The difference, of course, is that the data attribute lets you include just about any file type, not just an image file. In fact, the <object> tag expects, but doesn't require, that you explicitly name an enabling application for the object with the classid attribute, or indicate the MIME type of the file via the type attribute to help the browser decide how to process and render the data.

For example, here is an image included as an object, rather than as an <img> file:

```
<object data="pics/kumquat.gif" type="image/gif">
</object>
```

12.2.1.6 The type attribute

The type attribute lets you explicitly define the MIME type of the data that appears in the file you declare with the data attribute. (Use codetype to indicate an applet's MIME type.) If you don't provide data, or if the MIME type of the data is apparent from the URL or is provided by the server, you may omit this attribute. We recommend you include it anyway to ensure that your data is handled correctly by the browser and the included object.

For examples of data MIME types, look in your browser preferences for applications. There you'll find a list of the many file data types your browser will recognize and the application, if not the browser itself, that will process and render that file type.

12.2.1.7 The align, class, border, height, hspace, style, vspace, and width attributes

There are several attributes that let you control the appearance of the `<object>` display region exactly like the corresponding attributes for the `<img>` tag. The `height` and `width` attributes control the size of the viewing region. The `hspace` and `vspace` attributes define a margin around the viewing region. The value for each of these dimension attributes should be an actual number of pixels.

The `align` attribute (deprecated in the HTML 4 and XHTML standards here as well as for `<img>` and all other tags in lieu of style sheets, yet still popularly used) determines how the browser aligns the region in context with the surrounding text. Use `top`, `texttop`, `middle`, `absmiddle`, `baseline`, `bottom`, or `absbottom` to align the object display space with adjacent text, or `left` and `right` alignments for wraparound content.

The display region's dimensions often must match some other applet requirement, so be careful to check these values with the applet programmer. Sometimes, the applet may scale its display output to match your specified region.

For example, suppose our example clock applet should grow or shrink to fit nearly any size display region. We might create a square clock 100 x 100 pixels:

```
<object classid="clock.class" height="100" width="100">
</object>
```

As with `<img>`, use the `border` attribute to control the width of the frame that surrounds the object's display space when you include it as part of a hyperlink. The null value (`border=0`) removes the frame. Section 5.2.6

Use the `class` and `style` attributes, of course, to control the display style for the content enclosed by the tag and to format the content according to a predefined class of the `<object>` tag. Section 8.1.1 / Section 8.3

12.2.1.8 The declare attribute

The `declare` attribute lets you define an object but restrains the browser from downloading and processing it. Used in conjunction with the `name` attribute, this facility is similar to a forward declaration in a more conventional programming language that lets you defer download of an object until it actually gets used in the document.

12.2.1.9 The id, name, and title attributes

Use the `id` or `name` attributes to uniquely label an object. Use the `title` attribute to simply entitle the tag. Each attribute's value is a text string. The browser may choose to display a title to the user or may use it in some other manner while rendering the document. Use `id` or `name` to reference the object in other elements of your document, including hyperlinks and other objects.

For example, suppose you have two clock applets in your document, along with two applets the user operates to set those clocks. Provide unique labels for the clock applets using the `name` or `id` attribute, then pass those labels to the setting applets using the `<param>` tag, which we discuss later in this chapter:

```
<object classid="clock.class" name="clock1">
</object>
<object classid="clock.class" name="clock2">
</object>
<object classid="setter.class">
  <param name="clockToSet" value="clock1">
</object>
<object classid="setter.class">
  <param name="clockToSet" value="clock2">
</object>
```

Since we have no need to distinguish between the setter applets, we choose not to identify their instances.

The popular browsers support `name` and `id`, but earlier versions of Internet Explorer supported only `name`. For now, it may pay to use either `name` or both `name` and `id` in order to be compatible with the majority of browsers today.

12.2.1.10 The shapes and usemap attributes

Recall from our detailed discussion of hyperlinks in Chapter 6, that you can divide a picture into geometric regions and attach a hyperlink to each, creating a so-called image map. The `shapes` and `usemap` attributes for the `<object>` tag generalize that feature to include other object types.

The standard shapes attribute informs the browser that the <object> tag's contents are a series of hyperlinks and shape definitions. The usemap attribute and required URL value point to a <map> where you define the shapes and associated hyperlinks identical to the client-side image map discussed in Section 6.5.2.

For example, here is the image map we described in Chapter 6, rewritten in XHTML as a "shaped" object:

```
<object data="pics/map.gif" shapes="shapes">
  <a shape="rect" coords="0,0,49,49" href="main.html#link1"></a>
  <a shape="rect" coords="50,0,99,49" href="main.html#link2"></a>
  <a shape="rect" coords="0,50,49,99" href="main.html#link3"></a>
  <a shape="rect" coords="50,50,99,99" href="main.html#link4"></a>
</object>
```

and as the more familiar image map:

```
<object data="pics/map.gif" usemap="#map1">
</object>
...
<map name="map1">
  <area coords="0,0,49,49" href="main.html#link1" />
  <area coords="50,0,99,49" href="main.html#link2" />
  <area coords="0,50,49,99" href="main.html#link3" />
  <area coords="50,50,99,99" href="main.html#link4" />
</map>
```

You also may take advantage of all the attributes associated with the hyperlink, map, and <area> tags to define and arrange the image map regions. For instance, we recommend that you include alternative (alt attribute) text descriptions for each sensitive region of the image map.

12.2.1.11 The standby attribute

The standby attribute lets you display a message - the attribute's value text string - during the time the browser is downloading the object data. If your objects are large or if you expect slow network response, add this attribute as a courtesy to your users.

12.2.1.12 The tabindex and notab attributes

For Internet Explorer with ActiveX objects only, the notab attribute excludes the object from the document tabbing order.

As an alternative to the mouse, users also may press the Tab key to select and the Return or Enter key to activate a hyperlink or to access a form control. Normally, each time the user presses the Tab key, the browser steps to the next hyperlink or form control in the order that they appear in the document. Use the HTML 4/XHTML-standard tabindex and an integer value to modify the position the object occupies in the sequence of tab-selected elements on the page.

12.2.1.13 The dir and lang attributes

Use the dir and lang attributes, like their counterparts for most other tags, to specify the language and dialect of the <object>-enclosed contents as well as the direction by which the browser adds the text characters to the display. Section 3.6.1.1 / Section 3.6.1.2

12.2.1.14 Object event handling

As user-initiated mouse and keyboard events occur within the object, you may want to perform special actions. Accordingly, you can use the ten standard event attributes to catch these events and execute JavaScript code. We describe JavaScript event handlers more fully in Section 12.3.3.

12.2.1.15 Supporting incompatible browsers

Since some browsers may not support applets or the <object> tag, sometimes you may need to tell readers what they are missing. You do this by including body content between the <object> and </object> tags.

Browsers that support the <object> tags ignore the extraneous content inside. Of course, browsers that don't support objects don't recognize the <object> tags. Being generally tolerant of apparent mistakes, browsers usually ignore the unrecognized tag and blithely go on to display whatever content may appear inside. It's as simple as that. The following fragment tells object-incapable browser users they won't see our clock example:

```
<object classid=clock.class>
  If your browser were capable of handling applets, you'd see
  a nifty clock right here!
</object>
```

More importantly, object-capable browsers will display the contents of the <object> tag if they cannot load, execute, or render the object. If you have several objects of similar intent but with differing capabilities, you can nest their object tags. The browser will try each object in turn, stopping with the first one it can handle. Thus, the outmost object might be a full-motion video. Within that <object> tag, you might include a simpler MPEG video, and within that <object>, a simple GIF image. If the browser can handle full-motion video, your users get the full effect. If that level of video isn't available, the browser can try the simpler MPEG video stream. If that fails, the browser can just display the image. If images aren't possible, that innermost <object> might contain a textual description of the object.

12.2.2 The <param> Tag

The <param> tag supplies parameters for a containing <object> or <applet> tag. (We discuss the deprecated <applet> tag in Section 12.2.3.)

<param>

Function:

Supply a parameter to an embedded object

Attributes:

ID
NAME
TYPE
VALUE
VALUETYPE

End tag:

None in HTML; </param> or <param ... /> with XHTML

Contains:

Nothing

Used in:

applet_content

The <param> tag has no content and, with HTML, no end tag. It appears, perhaps with other <param> tags, only between an <object> or <applet> tag and its end tag. Use the <param> tag to pass parameters to the embedded object, such as a Java applet, as required for it to function correctly.

12.2.2.1 The name and value attributes

The <param> tag has two required attributes: name or id, and value. You've seen these before with forms. Together, they define a name/value pair that the browser passes to the applet.

For instance, our clock applet example might let users specify the time zone by which it sets its hour hand. To pass the parameter identified as "timezone" with the value "EST" to our example applet, specify the parameters as:

```
<object classid="clock.class">
  <param name="timezone" value="EST" />
</object>
```

The browser will pass the name/value pairs to the applet, but that is no guarantee that the applet is expecting the parameters, that the names and values are correct, or that the applet will even use the parameters. Correct parameter names, including capitalization and acceptable values, are determined by the applet author. The wise author will work closely with the applet programmer or have detailed documentation to ensure that the applet parameters are named correctly and assigned valid values.

12.2.2.2 The type and valuetype attributes

Use the `type` and `valuetype` attributes to define the type of the parameter the browser passes to the embedded object and how that object is to interpret the value. The `valuetype` attribute can have one of three values: `data`, `ref`, or `object`. The value `data` indicates that the parameter value is a simple string. This is the default value. The `valuetype` of `ref` indicates that the value is a URL of some other resource on the web. Finally, `object` indicates that the value is the name of another embedded object in the current document. This may be needed to support inter-object communication within a document.

The value of the `type` attribute is the MIME media type of the value of the parameter. This is usually of no significance when the parameter value is a simple string, but can be important when the value is actually a URL pointing to some other object on the Web. In those cases, the embedded object may need to know the MIME type of the object in order to use it correctly. For example, this parameter tells the embedded object that the parameter is actually the URL of a Microsoft Word document:

```
<param name="document" value="http://kumquats.com/quat.doc"
   type="application/msword" valuetype="ref" />
```

12.2.3 The <applet> Tag (Deprecated)

Use the `<applet>` tag within your document to download and execute an applet. Also, use the tag to define a region within the document display for the applet's display area. You may supply alternative content within the `<applet>` tag for display by browsers that do not support applets.

<applet>

Function:

Insert an application into the current text flow

Attributes:

ALIGN	ID
ALT	MAYSCRIPT N
ARCHIVE	NAME
CLASS	OBJECT
CODE	STYLE
CODEBASE	TITLE
HEIGHT	VSPACE
HSPACE	WIDTH

End tag:

</applet>; never omitted

Contains:

applet_content

Used in:

text

Most applets require one or more parameters that you supply in the document to control their execution. Put these parameters between the `<applet>` tag and its corresponding `</applet>` end tag using the `<param>` tag. The browser passes the document-specific parameters to the applet at time of execution. Section 12.2.2

The <applet> tag has been deprecated in the HTML 4 and XHTML standards in deference to the generalized <object> tag, which can do the same as <applet> and much more. Nonetheless, <applet> is a popular tag and remains supported by the popular browsers. Don't expect that it will go away any time soon, but do realize that <applet> *will* go away.

12.2.3.1 Applet rendering

The browser creates an applet's display region in the containing text flow exactly like an inline image: without line breaks and as a single large entity. The browser downloads and executes the applet just after download and display of the document, and continues execution until the code terminates itself or when the user stops viewing the page containing the applet.

12.2.3.2 The align attribute

Like an image, you may control the alignment of an applet's display region with respect to its surrounding text. As with the <img> tag, set the align attribute's value to top, texttop, middle, absmiddle, baseline, bottom, or absbottom, as well as left and right alignments for wraparound content. Section 5.2.6

12.2.3.3 The alt attribute

The alt attribute gives you a way to tell users gracefully that something is missing if, for some reason, the applet cannot or will not execute on their computer. Its value is a quote-enclosed message string that, like the alt attribute for images, gets displayed in lieu of the applet itself.

The alt message is only for browsers that support applets. See Section 12.2.1.5 in order to find out how to inform users of applet-incapable browsers why they can't view an applet.

12.2.3.4 The archive attribute

The archive attribute collects common Java classes into a single library that is cached on the user's local disk. Once cached, the browser doesn't need to use the network to access an applet; it retrieves the software from the local cache, thereby reducing the inherent delays of additional network activity to load the class.

The value of the archive attribute is a URL identifying the archive file. The suffix of the archive filename may be either *.zip* or *.jar*. Archived *.zip* files are in the familiar ZIP archive format, generated by PKZIP and many other utilities. Archived *.jar* files are in the Java archive format. Archived *.jar* files support compression and advanced features like digital signatures.

You may use the archive attribute with any <applet> tag, even if the class referenced by the tag's code attribute does not exist in the archive. If the class is not found in the archive, the browser simply attempts to retrieve the class relative to the document URL or the codebase URL, if specified.

12.2.3.5 The code and codebase attributes

The code attribute is required. Use code to specify the filename, *not* the URL, of the Java class to be executed by the browser. Like <object>, make the search relative to another storage location by using the codebase attribute described in Section 12.2.1.2 or an archive, as described in Section 12.2.3.4. Also, the extension suffix of the filename should be *.class*. If you don't include the suffix, some browsers will append *.class* automatically when searching for the applet.

Here is our infamous clock example rewritten as an <applet>:

```
<applet code="clock.class" codebase="http://www.kumquat.com/classes/">
</applet>
```

which the browser will retrieve and display from:

```
http://www.kumquat.com/classes/clock.class
```

12.2.3.6 The name attribute

The name attribute lets you supply a unique name for this instance of the code class - the copy of the applet that runs on the individual user's computer. Like other named elements in your document, providing a name for the applet lets other parts of your document, including other applets, reference and interact with this one, such as sharing computed results.

12.2.3.7 The height, hspace, vspace, and width attributes

Use the height and width attributes (identical to the counterparts for the <img> and <object> tags) to define the size of the applet's display region in the document. Use hspace and vspace to interpose some empty space around the applet region and thereby set it off from the text. They all accept values indicating the size of the region in pixels. Section 5.2.6.10

12.2.3.8 The mayscript attribute

The `mayscript` attribute, supported only by Netscape, indicates that the Java applet will be accessing JavaScript features within the browser. Normally, Java applets attempting to access JavaScript cause a browser error. If your applets access JavaScript, you must specify `mayscript` in the `<applet>` tag.

12.2.3.9 The title attribute

The value of this attribute is a quoted string, which is used by Internet Explorer to provide a title, if necessary, for the applet. This attribute is not supported by Netscape.

12.2.3.10 The object attribute

This unfortunately named attribute and its string value reference the name of the resource that contains a serialized version of the applet. How and what it does is an enigma; none of the popular browsers support it.

12.2.4 The <embed> Tag

Use the `<embed>` tag to include a reference in your document to some special plug-in application and perhaps data for that application. The standard analog for `<embed>` is the `<object>` tag with the `data` attribute.

<embed> N O

Function:

Embed an object in a document

Attributes:

ALIGN N	PLUGINSPAGE N
BORDER N	SRC
HEIGHT	TYPE N
HIDDEN	UNITS
HSPACE N	VSPACE N
NAME	WIDTH
PALETTE	

End tag:

None

Contains:

Nothing

Used in:

text

With `<embed>`, reference the data object via the `src` attribute and URL value for download by the browser. The browser uses the MIME type of the `src`'d object to determine the plug-in required to process the object. Alternatively, you may also use the `type` attribute to specify a MIME type without an object, and thereby initiate execution of a plug-in application, if it exists on the user's computer.

Like all other tags, the nonstandard <embed> tag extension has a set of predefined attributes that define parameters and modify the tag's behavior. Unlike most other tags, however, the browsers let you include plug-in-specific name/value attribute pairs in <embed> that, instead of altering the action of the tag itself, get passed to the plug-in application for further processing.

For example, this tag:

```
<embed src=movie.avi width=320 height=200 autostart=true loop=3>
```

has attributes that are processed by the <embed> tag (src, width, and height) and two that are not recognized, but rather passed to the plug-in associated with AVI video clips: autostart and loop.

It is not possible to document all the possible attributes that the many different plug-ins might need with their associated <embed> tag. Instead, you must turn to the plug-in developer to learn about all their required and optional attributes for each particular plug-in you plan to use in your pages.

12.2.4.1 The align, border, height, width, hspace, and vspace attributes

The browser displays embedded objects to the user in a region set aside within the document window. The <embed> tag's align, border, height, width, hspace, and vspace attributes let you control the appearance of that region exactly as they do for the <img> tag, so we won't belabor them. Section 5.2.6

Briefly, the height and width attributes control the size of the viewing region. Normally, you should specify the height and width in pixels, but you may also use some other units of measure if you also specify the units attribute (see Section 12.2.4.8). The hspace and vspace attributes define a margin, in pixels, around the viewing region. The align attribute determines how the browser aligns the region within surrounding text, while the border attribute determines the width of the border surrounding the viewing region.

Only Netscape supports the align, border, hspace, and vspace attributes for the <embed> tag.

12.2.4.2 The hidden attribute

The hidden attribute makes an object invisible to the user, forcing it to have a height and width of zero. Note that setting hidden does not cause the browser to display an empty region within the document, but rather completely removes the object from the containing text flow.

The attribute's useful for audio streams placed within documents. The HTML entry:

```
<embed src=music.wav hidden autostart=true loop=true>
```

embeds an audio object in the page. The browser does not show anything to the user but rather plays background music for the page. By contrast, the plug-in associated with:

```
<embed src=music.wav>
```

might present an audio control panel to users so that they might start and stop the audio playback, adjust the volume, and so forth.

12.2.4.3 The name attribute

Like other name attributes, this one also lets you label the embedded object for later reference by other elements in your document, including other objects. The value of the attribute is a character string.

12.2.4.4 The palette attribute

The palette attribute is supported by both Netscape and Internet Explorer, but in completely different ways. With Netscape, the value of the palette attribute is either foreground or background, indicating which palette of window system colors the plug-in uses for its display.

With Internet Explorer, the value of palette is instead a pair of hexadecimal color values, separated by a vertical bar. The first value determines the foreground color used by the plug-in; the second sets the background color. Thus, specifying this palette:

```
palette=#ff0000|#00ff00
```

causes the plug-in to use red as its foreground color and green as its background color. For a complete description of hexadecimal color values, see Appendix G.

12.2.4.5 The pluginspage attribute

The pluginspage attribute, supported only by Netscape, specifies the URL of a web page that provides instruction on where to obtain and how to install the plug-in associated with the embedded object.

12.2.4.6 The src attribute

Like its document-referencing counterparts for a myriad of other tags, the `src` attribute supplies the URL of the data object that you embed in the HTML document. The server providing the object must be configured so that it notifies the browser of the correct MIME type of the object. If not, the browser will use the suffix of the last element of the `src` value - the object's filename in the URL path - to determine the type of the object. The browser uses this MIME type to determine which plug-in it will execute to process the object.

If you don't include a `src` attribute with the `<embed>` tag, then you've got to include a `type` attribute to explicitly reference the MIME type and as a result the plug-in application.

12.2.4.7 The type attribute

Use the `type` attribute in addition to or in lieu of the `src` attribute. Its value explicitly indicates the MIME type of the embedded object, which in turn determines which plug-in the browser will invoke to process the object. This attribute is not required if you include the `src` attribute and the browser can determine the object type from the object's URL or server. You must supply a `type` attribute if you don't include the `src` attribute.

It may seem odd to use an `<embed>` tag without a `src` attribute reference to some object, but this is common if the plug-in requires no data or retrieves its data dynamically after it is started. In these cases, the `type` attribute is required so that the browser knows which plug-in to invoke.

12.2.4.8 The units attribute

Pixels are the default unit of measure for the `height` and `width` attributes that control the `<embed>` display space. The `units` attribute lets you explicitly state that the absolute measure is `pixels`, or change it to the relative `en`, which is one-half the current point size of text in the document. With the `en` units, you tailor the object's viewing area (viewport) to be proportional to its immediately surrounding content, the size of which is varied by the user.

For example:

```
<embed src=movie.avi height=200 width=320 units=pixels>
```

creates a viewport for the window 200 x 320 pixels. By changing `units` to `en`, that same viewport, when included within a flow of 12-point text, will become 1200 x 1920 pixels.

12.2.5 The <noembed> Tag

Some browsers do not support the `<embed>` tag. The `<noembed>` tag makes it easy to supply alternative content that tells users what they are missing.

<noembed> 

Function:

Supply content to <embed>-incompatible browsers

Attributes:

None

End tag:

</noembed>; never omitted

Contains:

Nothing

Used in:

text

The popular browsers ignore the contents of the `<noembed>` tag, whereas browsers that do not support the `<embed>` tag will display the contents of the `<noembed>` tag. Normally, use the contents of the `<noembed>` tag to display some sort of message placating users of inadequate browsers:

```
<embed src=cool.mov autostart=true loop=true>
<noembed>To view the cool movie, you need to upgrade to a browser
that supports the &lt;embed&gt; tag!</noembed>
```

We recommend using a <noembed> message only in those cases where the object is crucial for the user to comprehend and use your document. And, in those cases, provide a link to a document that can stand alone without the embedded object, or nicely explain the difficulty.

12.3 JavaScript

All the executable content elements we've talked about so far have had one common trait: they are separate from the browser and the HTML/XHTML document - separate data, separate execution engine.

JavaScript is different. It is a scripting language that taps the native functionality of the browser. You may sprinkle JavaScript statements throughout your documents, either as blocks of code or single statements attached to individual tags. The JavaScript-enabled browsers, including both Netscape Navigator and Internet Explorer, interpret and act upon the JavaScript statements you provide to do such things as alter the appearance of the document, control the display, validate and manipulate form elements, and perform general computational tasks.

As with Java, we will not pretend to teach JavaScript programming in this book. We'll show you how to embed and execute JavaScript within your documents and ask that you turn to books like *JavaScript: The Definitive Guide* (O'Reilly & Associates) for a complete definition of the JavaScript language.

12.3.1 The <script> Tag

One way to place JavaScript code in your document is via the HTML and XHTML standard <script> tag.

Everything between <script> and </script> is processed by the browser as executable JavaScript statements and data. You cannot place HTML or XHTML within this tag; it will be flagged as an error by the browser.

Browsers that do not support <script> will process contents of the tag as regular HTML, to the confusion of the user. For this reason, we recommend that you include the contents of the <script> tag inside HTML comments:

```
<script language="JavaScript">
<!--
   JavaScript statements go here
// -->
</script>
```

For browsers that ignore the <script> tag, the contents are masked by the comment delimiters <!-- and -->. JavaScript-enabled browsers, on the other hand, automatically recognize and interpret the JavaScript statements delimited by the comment tags. By using this skeleton for all your <script> tags, you can be sure that all browsers will handle your document gracefully, if not completely.

Unfortunately, as we discuss in Chapter 16, as with document-level stylesheets, script content for XHTML documents must be within a special CDATA declaration rather than within comments. Hence, HTML browsers won't honor XHTML scripts, and vice versa. Our only recommendation at this point is to follow the popular browsers: write in HTML, but use as many of the features of XHTML as you can in preparation for the future.

You may include more than one <script> tag in a document, located in either the <head> or the <body>. The JavaScript-enabled browser executes the statements in order. Variables and functions defined within one <script> tag may be referenced by JavaScript statements in other <script> tags. In fact, one common JavaScript programming style is to use a single <script> in the document <head> to define common functions and global variables for the document and then to call those functions and reference their variables in other JavaScript statements sprinkled throughout the document.

12.3.1.1 The language and type attributes

Use the language or type attribute in the <script> tag to declare the scripting language that you used to compose the contents of the tag. The language attribute is deprecated by the HTML 4 and XHTML standards in favor of the type attribute. Regrettably, the value for each attribute is different.

If you are using JavaScript - by far the most common scripting language on the Web - use language=JavaScript or use type=text/javascript. You may occasionally see the language value VBScript (text/vbscript for type), indicating that the enclosed code is written in Microsoft's Visual Basic Script.

With JavaScript, you may also use the language value "JavaScript 1.1", indicating that the enclosed script is to be processed only by Netscape 3.0 or later. Netscape 2.0, which supports JavaScript 1.0, will not process scripts identified as "JavaScript 1.1".

<script>

Function:

Define an executable script within a document

Attributes:

CHARSET
DEFER
LANGUAGE
SRC
TYPE

End tag:

</script>; never omitted

Contains:

scripts

Used in:

head_content, body_content

12.3.1.2 The src and charset attributes

For particularly large JavaScript programs or ones you reuse often, you might want to store the code in a separate file. In these cases, have the browser load that separate file through the `src` attribute. The value of the `src` attribute is the URL of the file containing the JavaScript program. The stored file should have a MIME type of `application/x-javascript`, but will also be properly handled automatically by a properly configured server if the filename suffix is *.js*.

For example:

```
<script language="JavaScript" src="http://www.kumquat.com/quatscript.js">
</script>
```

tells the `<script>`-able browser to load a JavaScript program named *quatscript.js* from the server. Although there are no `<script>` contents, the ending `</script>` is still required.

Used in conjunction with the `src` attribute, the `charset` attribute tells the browser the character set used to encode the JavaScript program. Its value is the name of any ISO standard character set encoding.

12.3.1.3 The defer attribute

Some JavaScript scripts are used to create actual document content using the `document.write` method; others are not. If your scripts do not alter the contents of the document, add the `defer` attribute to the `<script>` tag to speed processing of your document. Since the browser knows that it can safely read the remainder of the document without executing your scripts, it will defer interpretation of the script until after the document has been rendered for the user.

12.3.2 The <noscript> Tag

Tell users of browsers that do not support the `<script>` tag that they are missing something via the `<noscript>` tag.

Unfortunately, only Netscape 3.0 and Internet Explorer 4.0 and later versions ignore the contents of the `<noscript>` tag. So even `<script>`-able browsers like Netscape 2 and Internet Explorer 3 will display the contents of the `<noscript>` tag, to the confusion of their users. There are other ways to detect and handle `<script>`-challenged browsers, detailed in any good JavaScript book.

<noscript>

Function:

Supply content to <script>-challenged browsers

Attributes:

None

Contains:

body_content

Used in:

text

The `<noscript>` tag supports the sixteen standard HTML 4/XHTML attributes: `class` and `style` for style management, `lang` and `dir` for language type and display direction, `title` and `id` for titling and naming the enclosed content, and the event attributes for user-initiated processing. Section 3.6.1.1 / Section 3.6.1.2 / Section 4.1.1.4 / Section 4.1.1.5 / Section 8.1.1 / Section 8.3 / Section 12.3.3

12.3.3 JavaScript Event Handlers

One of the most important features provided by JavaScript is the ability to detect and react to events that occur while a document is loading, rendering, and being browsed by the user. The JavaScript code that handles these events may be placed within the `<script>` tag, but more commonly, it is associated with a specific tag via one or more special tag attributes.

For example, you might want to invoke a JavaScript function when the user passes the mouse over a hyperlink in a document. The JavaScript-aware browsers support a special "mouse over" event-handler attribute for the `<a>` tag called `onMouseOver` to do just that:

```
<a href="doc.html" onMouseOver="status='Click me!';
return true">
```

When the mouse passes over this example link, the browser executes the JavaScript statements. (Notice that the two JavaScript statements are enclosed in quotes and separated by a semicolon, and that single quotes surround the text-message portion of the first statement.)

While a complete explanation of this code is beyond our scope, the net result is that the browser places the message "Click me!" in the status bar of the browser window. Commonly, authors use this simple JavaScript function to display a more descriptive explanation of a hyperlink, in place of the often cryptic URL that the browser traditionally displays in the status window.

HTML and XHTML both support a rich set of event handlers through related "on" event tag attributes. The value of any of the JavaScript event handler attributes is a quoted string containing one or more JavaScript statements separated by semicolons. Extremely long statements can be broken across several lines, if needed. Care should also be taken in using entities for embedded double quotes in the statements to avoid a syntax error when processing the attribute value.

12.3.3.1 Standard event handler attributes

Table 12-1 presents the current set of event handlers as tag attributes. Most are supported by the popular browsers, which also support a variety of nonstandard event handlers as well, tagged with asterisks in the table.

We put the event handlers into two categories: user- and document-related. The user-related ones are the mouse and keyboard events that occur when the user handles either device on the computer. User-related events are quite ubiquitous, appearing as standard attributes in nearly all the standard tags (even though they may not yet be supported by any browser), so we don't list their associated tags in Table 12-1. Instead, we'll tell you which tags *do not* accept these event attributes: `<applet>`, `<base>`, `<basefont>`, `<bdo>`, `<br>`, `<font>`, `<frame>`, `<frameset>`, `<head>`, `<html>`, `<iframe>`, `<isindex>`, `<meta>`, `<param>`, `<script>`, `<style>`, and `<title>`.

Table 12-1, Event Handlers	
Event Handler	**HTML/XHTML Tags**
onAbort*	<img>
onBlur	<a> <area> <body> <button> <frameset> <input> <label> <select> <textarea>
onChange	<input> <select> <textarea>
onClick	Most tags
onDblClick	Most tags
onError*	<img>
onFocus	<a> <area> <body> <button> <frameset> <input> <label> <select> <textarea>
onKeyDown	Most tags
onKeyPress	Most tags
onKeyUp	Most tags
onLoad	<body> <frameset> <img>*
onMouseDown	Most tags
onMouseMove	Most tags
onMouseOut	Most tags
onMouseOver	Most tags
onMouseUp	Most tags
onReset	<form>
onSelect	<input> <textarea>
onSubmit	<form>
onUnload	<body> <frameset>

Some events, however, occur rarely and with special tags. These relate to the special events and states that occur during the display and management of a document and its elements by the browser.

12.3.3.2 The mouse-related events

The `onClick` , `onDblClick`, `onMouseDown` and `onMouseUp` attributes refer to the mouse button. The `onClick` event happens when the user presses down and then quickly releases the mouse button, unless the user then quickly clicks the mouse button for a second time. In that latter case, the `onDblClick` event gets triggered in the browser.

If you need to detect both halves of a mouse click as separate events, use `onMouseDown` and `onMouseUp`. When the user presses the mouse button, the `onMouseDown` event occurs. The `onMouseUp` event happens when the user releases the mouse button.

The `onMouseMove` , `onMouseOut`, and `onMouseOver` events happen when the user drags the mouse pointer. The `onMouseOver` event occurs when the mouse first enters the display region occupied by the associated HTML element. After entry, `onMouseMove` events are generated as the mouse moves about within the element. Finally, when the mouse exits the element, `onMouseOut` occurs.

For some elements, the `onFocus` event corresponds to `onMouseOver`, and `onBlur` corresponds to `onMouseOut`.

12.3.3.3 The keyboard events

Only three events currently are supported by the HTML 4 and XHTML standards relating to user keyboard actions: `onKeyDown` , `onKeyUp` and `onKeyPress`,. The `onKeyDown` event occurs when the user depresses a key on the keyboard; `onKeyUp` happens when the key is released. The `onKeyPress` attribute is triggered when a key is pressed and released. Usually, you'll have handlers for either the up and down events, or the composite keypress event, but not for both.

12.3.3.4 Document events

Most of the document-related event handlers relate to the actions and states of form controls. For instance, `onReset` and `onSubmit` happen when the user activates the respective `reset` or `submit` button. Similarly, `onSelect` and `onChange` occur as users interact with certain form elements. Please consult Chapter 9 for a detailed discussion of these forms-related events.

There also are some document-related event handlers that occur when various document elements get handled by the browser. For instance, the `onLoad` event may happen when a frameset is complete, or when the body of an HTML or XHTML document gets loaded and displayed by the browser. Similarly, `onUnload` occurs when a document is removed from a frame or window.

12.3.4 JavaScript URLs

You can replace any conventional URL reference in a document with one or more JavaScript statements. The browser then executes the JavaScript code, rather than downloading another document, whenever the browser references the URL. The result of the last statement is taken to be the "document" referenced by the URL and is displayed by the browser accordingly. The result of the last statement is *not* the URL of a document; it is the actual content to be displayed by the browser.

To create a JavaScript URL, use `javascript` as the URL's protocol:

```
<a href="javascript:generate_document(  )">
```

In the example, the JavaScript function `generate_document( )` gets executed whenever the hyperlink gets selected by the user. The value returned by the function, presumably a valid HTML or XHTML document, is rendered and displayed by the browser.

It may be that the executed statement returns no value. In these cases, the current document is left unchanged. For example, this JavaScript URL:

```
<a href="javascript:alert('Error!')">
```

pops up an alert dialog box and does nothing else. The document containing the hyperlink would still be visible after the dialog box was displayed and dismissed by the user.

12.3.5 JavaScript Entities

Character entities in HTML and XHTML consist of an ampersand (`&`), an entity name or number, and a closing semicolon. For instance, to insert the ampersand character itself in a document text flow, use the character sequence `&`. Similarly, JavaScript entities consist of an ampersand, one or more JavaScript statements enclosed in curly braces, and a closing semicolon.

For example:

```
&{document.fgColor};
```

More than one statement must be separated by semicolons within the curly braces. The value of the last (or only) statement is converted to a string and replaces the entity in the document.

Normally, entities can appear anywhere in an document. JavaScript entities are restricted to values of tag attributes. This lets you write "dynamic tags" whose attributes are not known until the document is loaded and the JavaScript executed. For example:

```
<body text=&{favorite_color(  )};>
```

will set the text color of the document to the color value returned by the individual's `favorite_color( )` function.

12.3.6 The <server> Tag

The `<server>` tag is a strange beast. It is processed by the web server and never seen by the browser. So what you can do with this tag depends on the server you are using, not the reader's browser.

Netscape's server (not to be confused with the Netscape Navigator browser) uses the `<server>` tag to let you to place JavaScript statements within a document that get processed by the server. The results of the executed JavaScript then gets inserted into the document, replacing the `<server>` tag. A complete discussion of this so-called "server-side" JavaScript is completely beyond this book; we include this brief reference only to document the `<server>` tag.

<server>

Function:

Define server-side JavaScript

Attributes:

None

End tag:

</server>; never omitted

Contains:

JavaScript

Used in:

head_content

Like the `<script>` tag, the `<server>` tag contains JavaScript code. However, the latter tag and content code must appear inside the document `<head>`. It is extracted from the document and executed by the server when the document is requested for download.

Obviously, server-side JavaScript is tightly coupled to the server, not to the browser. To fully exploit this tag and the benefits of server-side JavaScript or other server-side programming languages, consult your web server's documentation.

12.4 JavaScript Style Sheets

Much of a browser's work is manipulating the display, and much of its display code already has been exposed for JavaScripting. So it seemed only natural, perhaps even relatively easy, for the developers at Netscape to implement JavaScript Style Sheets. Based on the W3C recommended Cascading Style Sheet model (CSS; see Chapter 8), this alternative document style technology lets you prescribe display properties for all the various HTML elements, either inline as tag attributes, at the document level, or for an entire document collection.

JavaScript Style Sheets (JSS) are a Netscape invention. In fact, for a short time in the fall of 1996, Netscape appeared ready to eschew the CSS methodology, which Internet Explorer had already implemented, and use JSS exclusively for HTML document designers with its then-current browser, Netscape Navigator 4. In the end, Netscape Navigator 4 supported both JSS and CSS technologies while Netscape 6 eschews support for JSS in favor of the standard CSS2. At this point, CSS should be seen as Netscape's long-term direction.

We are strong proponents of reasonable standards, and now that the CSS2 model is fully supported in HTML 4 and XHTML, we can't recommend that you use anything but CSS-standard style sheets. Evidently, Netscape now agrees with us on this point.

The CSS model is a good one, and it is good that Netscape has decided to support it. Whether Internet Explorer will someday support JSS is not known, but we doubt it. It is clear that Microsoft intends continued support for the CSS2 standard, as well as the HTML 4 and XHTML standards (they haven't had good results bucking web standards in the past).

But standards aren't the whole story. We can't imagine that the HTML author, let alone the page layout designer, is going to abide the rigid programming syntax of JavaScript. Nonetheless, there are some advantages to JSS that some authors will find useful, even though it restricts their document's full potential to the select Netscape 4 user.

We thoroughly discuss the concepts and ideas behind style sheets - specifically, Cascading Style Sheets - in Chapter 8, so we won't repeat ourselves here. Rather, we address only how to create and manipulate styles with JavaScript. Before forging ahead in this section, we recommend that you first absorb the information in Chapter 8.

12.4.1 JavaScript Style Sheet Syntax

Netscape implements JSS by extending several existing HTML tags and defining a few new objects that store your document's styles.

12.4.1.1 External, document-level, and inline JSS

As with CSS, you can reference and load external JSS files with the `<link>` tag. For example:

```
<link href="styles.js" rel=stylesheet type=text/JavaScript>
```

The only real difference between this tag and the one for a CSS external style sheet is that the `type` attribute of the `<link>` tag is set to `text/JavaScript` instead of `text/CSS`. The referenced file, *styles.js*, contains JavaScript statements that define styles and classes that Netscape will then use to control display of the current document.

Document-level JSS is defined within a `<style>` tag in the `<head>` of the document, just like CSS. Again, there is only one real difference in that the `type` attribute of the `<style>` tag is set to `text/JavaScript` instead of `text/CSS`.

The contents of the `<style>` tag for JSS are quite different from those for CSS, however. For example:

```
<style type=text/JavaScript>
<!--
    tags.BODY.marginLeft = "20px";
    tags.P.fontWeight = "bold";
  // -->
</style>
```

First, notice that we use the standard JavaScript and HTML comments to surround our JSS definitions, preventing noncompliant browsers from processing them as HTML content. Also notice that the syntax of the style definition is that of JavaScript, where letter case *does* make a difference, among other things.

You associate inline JavaScript-based style rules with a specific tag using the `style` attribute, just like CSS inline styles. The value of the attribute is a list of JSS assignments, separated by semicolons. For example:

```
<p style="color = 'green'; fontWeight = 'bold'">
```

creates a green, bold-faced text paragraph. Notice first that you need to enclose inline style values within single quotation marks, not double quotation marks, as you might use for document-level and in external JSS styles. This is reasonable, since the style attribute value itself must be enclosed in double quotation marks.

Also note that inline JSS definitions use only the property name, not the containing tag object that owns the property. This makes sense, since inline JSS styles affect only the current tag, not all instances of the tag.

12.4.1.2 JSS values

In general, all of the values you may use for CSS may also be used in JSS definitions. For keyword, length, and percentage values, simply enclose the value in quotes and use it as you would any string value in JavaScript. Thus, the CSS value `bold` becomes `"bold"` or `'bold'` for JSS document-level or inline styles, respectively; `12pt` in CSS becomes `'12pt'` or `"12pt"` in JSS.

Specify color values as the color name or a hexadecimal color value, enclosed in single or double quotes. The CSS decimal rgb notation is not supported in JSS.

JSS URL values are strings containing the desired URL. Thus, the CSS URL value `url(http://www.kumquat.com)` becomes `'http://www.kumquat.com'` for a JSS inline style; or `"http://www.kumquat.com"` at the document level.

One unique power of JSS is that any value can be computed dynamically when the document is processed by the browser. Instead of statically specifying the font size, for example, you can compute it on the fly:

```
tags.P.fontSize = favorite_font_size(  );
```

We assume that the JavaScript function `favorite_font_size( )` somehow determines the desired font size and returns a string value containing that size. This, in turn, is assigned to the `fontSize` property for the `<p>` tag, defining the font size for all paragraphs in the document.

12.4.1.3 Defining styles for tags

JavaScript defines a new document property, `tags`, that contains the style properties for all HTML tags. To define a style for a tag, simply set the appropriate property of the desired style property within the `tag` property of the `document` object. For example:

```
document.tags.P.fontSize = '12pt';
document.tags.H2.color = 'blue';
```

These two JSS definitions set the font size for the `<p>` tag to 12 points and render all `<h2>` tags in blue. The equivalent CSS definitions are:

```
p {font-size : 12pt}
h2 {color : blue}
```

Since the `tags` property always refers to the current document, you may omit `document` from any JSS tag style definition. We could have written the previous two styles as:

```
tags.P.fontSize = '12pt';
tags.H2.color = 'blue';
```

Moreover, as we mentioned previously, you may omit the tag name, as well as the `document` and `tags` properties for inline JSS using the `style` attribute.

Capitalization and case are significant in JSS. The tag names within the `tags` property must always be fully capitalized. The embedded capital letters within the tag properties are significant: any deviation from the exact lettering produces an error, and Netscape won't honor your JSS declaration. All of the following JSS definitions are invalid, though the reason is not overly apparent:

```
tags.p.fontsize = '12pt';
tags.Body.Color = 'blue';
tags.P.COLOR = 'red';
```

The correct versions are:

```
tags.P.fontSize = '12pt';
tags.BODY.color = 'blue';
tags.P.color = 'red';
```

It can be very tedious to specify a number of properties for a single tag, so you can take advantage of the JavaScript `with` statement to reduce your typing burden. These styles:

```
tags.P.fontSize = '14pt';
tags.P.color = 'blue';
tags.P.fontWeight = 'bold';
tags.P.leftMargin = '20%';
```

can be more easily written as:

```
with (tags.P) {
  fontSize = '14pt';
  color = 'blue';
  fontWeight = 'bold';
  leftMargin = '20%';
  }
```

You can apply similar styles to diverse tags just as easily:

```
with (tags.P, tags.LI, tags.H1) {
  fontSize = '14pt';
  color = 'blue';
  fontWeight = 'bold';
  leftMargin = '20%';
  }
```

12.4.1.4 Defining style classes

Like CSS, JSS lets you target styles for specific ways in which a tag may be used in your document. JSS uses the `classes`property to define separate styles for the same tag. There are no predefined properties within the `classes` property; instead, any property you reference is defined as a class to be used by the current document.

For example:

```
classes.bold.P.fontWeight = 'bold';
with (classes.abstract.P) {
   leftMargin = '20pt';
   rightMargin = '20pt';
   fontStyle = 'italic';
   textAlign = 'justify';
   }
```

The first style defines a class of the <p> tag named bold whose font weight is set to bold. The next style uses the with statement to create a class of the <p> tag named abstract with the specified properties. The equivalent CSS rules would be the following:

```
P.bold {font-weight : bold}
P.abstract {left-margin : 20pt;
            right-margin : 20pt;
            font-style : italic;
            text-align : justify
           }
```

Once defined, use a JSS class just like any CSS class: with the class attribute and the class name.

Like CSS, JSS also lets you define a class without defining the tag that will use the class. This lets you define generic classes that you can later apply to any tag. To create a generic style class in JSS, use the special tag property all:

```
classes.green.all.color = "green";
```

You can then add class="green" to any tag to have Netscape render its contents in green. The equivalent CSS is:

```
.green {color : green}
```

12.4.1.5 Using contextual styles

One of the most powerful aspects of CSS is its contextual style capability, wherein the browser applies a style to tags only if they appear in the document in a certain nesting. JSS supports contextual styles as well, through the special contextual( ) method within the tags property. The parameters to this method are the tags and classes that define the context in which Netscape will apply the style. For example:

```
tags.contextual(tags.UL, tags.UL, tags.LI).listStyleType = 'disc';
```

defines a context wherein the elements (tags.LI) of an unordered list nested within another unordered list (tags.UL, tags.UL) use the disc as their bullet symbol. The CSS equivalent is:

```
ul ul li {list-style-type : disc}
```

You can mix tags and classes in the contextual( ) method. For instance:

```
tags.contextual(classes.abstract.P, tags.EM).color = 'red';
```

tells the browser to display in red tags that appear within paragraphs that are of the abstract class. The CSS equivalent is:

```
p.abstract em {color : red}
```

Since the tags object is unambiguously included within the contextual( ) method, you may omit them from the definition. Hence, our nested list example may be rewritten as:

```
tags.contextual(UL, UL, LI).listStyleType = 'disc';
```

12.4.2 JavaScript Style Sheet Properties

A subset of the CSS style properties are supported in JSS. The JSS style properties, their CSS equivalents, and the sections in which those properties are fully documented are shown in Table 12-2.

JSS also defines three methods that allow you to define margins, padding, and border widths within a single style property. The three methods, margins( ), paddings( ), and borderWidths( ), accept four parameters, corresponding to the top, right, bottom, and left margin, padding or border width, respectively. Unlike their CSS counterparts (margin, Section 8.4.6.10; padding, Section 8.4.6.11; and border-width, Section 8.4.6.4), these JSS methods require that you always specify all four parameters. There is no shorthand way in JSS to set multiple margins, paddings, or border widths with a single value.

Table 12-2, JSS Properties and Cascading Style Sheet Equivalents		
JSS Property	**CSS Property**	**See Section**
align	float	Section 8.4.6.8
backgroundImage	background-image	Section 8.4.4.3
backgroundColor	background-color	Section 8.4.4.2
borderBottomWidth	border-bottom-width	Section 8.4.6.4
borderLeftWidth	border-left-width	Section 8.4.6.4
borderRightWidth	border-right-width	Section 8.4.6.4
borderStyle	border-style	Section 8.4.6.5
borderTopWidth	border-top-width	Section 8.4.6.4
clear	clear	Section 8.4.6.7
display	display	Section 8.4.8.1
fontSize	font-size	Section 8.4.3.2
fontStyle	font-style	Section 8.4.3.5
height	height	Section 8.4.6.9
lineHeight	line-height	Section 8.4.5.2
listStyleType	list-style-type	Section 8.4.7.3
marginBottom	margin-bottom	Section 8.4.6.10
marginLeft	margin-left	Section 8.4.6.10
marginRight	margin-right	Section 8.4.6.10
marginTop	margin-top	Section 8.4.6.10
paddingBottom	padding-bottom	Section 8.4.6.11
paddingLeft	padding-left	Section 8.4.6.11
paddingRight	padding-right	Section 8.4.6.11
paddingTop	padding-top	Section 8.4.6.11

JSS Property	CSS Property	See Section
textDecoration	text-decoration	Section 8.4.5.4
textTransform	text-transform	Section 8.4.5.9
textAlign	text-align	Section 8.4.5.3
textIndent	text-indent	Section 8.4.5.5
verticalAlign	vertical-align	Section 8.4.5.7
whiteSpace	white-space	Section 8.4.8.2
width	width	Section 8.4.6.12

Chapter 13. Dynamic Documents

The standard HTML/XHTML document model is static. Once displayed on the browser, a document does not change until the user initiates some activity like selecting a hyperlink with the mouse.[1] The Netscape developers found that limitation unacceptable and built in some special features to their browser that let you change HTML document content dynamically. In fact, they provide two different mechanisms for dynamic documents, which we describe in detail in this chapter. Internet Explorer supports some of these mechanisms, which we'll discuss as well.

[1] Of course you could embed animated GIFs or applets that dynamically update the display, but the underlying HTML document itself doesn't change.

We should mention that many of the features of dynamic documents have been displaced by plug-in browser accessories and in particular applets. Nonetheless, Netscape and Internet Explorer continue to support dynamic documents, and we believe the technology has virtues you should be aware of, if not take advantage of, in your HTML documents. Section 12.1

13.1 An Overview of Dynamic Documents

If you remember from our discussion in Chapter 1, the client browser initiates data flow on the Web by contacting a server with a document request. The server honors the request by downloading the document. The client subsequently displays the document's contents to the user. For normal web documents, a single transaction initiated from the client side is all that is needed to collect and display the document. Once displayed, however, it does not change.

Dynamic documents, on the other hand, are the result of multiple transactions initiated from either or both the server side and the client side. A *client-pull* document is one that initiates multiple transactions from the client side. When the server is the instigator, the dynamic document is known as a *server-push* document.

In a client-pull document, special HTML codes tell the client to periodically request and download another document from one or more servers on the network, dynamically updating the display.

Server-push documents also advance the way servers communicate with clients. Normally, over the Web, the client stays connected with a server for only as long as it takes to retrieve a single document. With server-push documents, the connection remains open and the server continues to send data periodically to the client, adding to or replacing the previous contents.

Netscape is currently the only browser able to handle server-push dynamic documents correctly; both Internet Explorer and Netscape support client-pull documents. With other browsers, you might see only part of the dynamic document at best. At worst, the browser will completely reject the document. Unfortunately, because dynamic documents are client-server processes, they don't work without an HTTP server. That means you can't develop and test your dynamic documents stored as local files unless you have a server running locally as well.

13.1.1 Another Word of Caution

As always, we tell you exactly how to use these exciting but nonstandard features, and we admonish you not to use them unless you have a compelling and overriding reason to do so. We are particularly strident with that admonition for dynamic documents, not only because they aren't part of the HTML standard, but because dynamic documents can hog the network. They require larger, longer downloads than their static counterparts. And they require many more (in the case of client-pull) or longer-term (for server-push) client-server connections. Multiple connections on a single server are limited to a few of the vast millions of web users at a time. We'd hate to see your readers miss out because you've created a jiggling image in a dynamic document that would otherwise have been an effective and readily accessible static document more people could enjoy.

13.2 Client-Pull Documents

Client-pull documents are relatively easy to prepare. All you need to do is embed a `<meta>` tag in the header of your HTML or XHTML document. The special tag tells the client Netscape browser to display the current document for a specified period of time, and then load and display an entirely new one just as if the user had selected the new document from a hyperlink. (Note that currently there is no way to change just a portion of a document dynamically using client-pull.) Section 6.8.1

13.2.1 Uniquely Refreshing

Client-pull dynamic documents work with Netscape and Internet Explorer because the browsers respond to a special HTTP header field called "Refresh."

You may recall from previous discussions that whenever an HTTP server sends a document to the client browser, it precedes the document's data with one or more header fields. One header field, for instance, contains a description of the document's content type, used by the browser to decide how to display the document's contents. For example, the server precedes HTML documents with the header "Content-type: text/html," whose meaning should be fairly obvious.

As we discussed at length in Chapter 6, you may add your own special fields to an HTML document's HTTP header by inserting a `<meta>` tag into its `<head>`. Section 6.8.1

The HTTP Refresh field implements client-pull dynamic HTML documents, enabled by the `<meta>` tag format:

```
<meta http-equiv="Refresh" content="field value">
```

The tag's `http-equiv` attribute tells the HTTP server to include the `Refresh` field, with a value specified by the `content` attribute (if any, carefully enclosed in quotation marks), in the string of headers it sends to the client browser just before it sends the rest of the document's content. The browser recognizes the Refresh header as the mark of a dynamic HTML document and responds accordingly, as we discuss next.

13.2.2 The Refresh Header Contents

The value of the `content` attribute in the special `<meta>` Refresh tag determines when and how the browser updates the current document. Set it to an integer, and the browser will delay that many seconds before automatically loading another document. You may set the content field value to zero, meaning no delay at all. In that case, the browser loads the next document immediately after it finishes rendering the current one, by which you may achieve some very crude animation effects. Section 6.8.1.2

13.2.2.1 Refreshing the same document

If the Refresh field's content value is the number of seconds alone, the browser reloads that same document over and over again, delaying the specified time between each cycle, until the user goes to another document or shuts down the browser.

For example, the browser will reload the following client-pull document every 15 seconds:

```
<html>
<head>
<meta http-equiv="Refresh" content="15">
<title>Kumquat Market Prices</title>
</head>
<body>
<h3> Kumquat Market Prices</h3>
Kumquats are currently trading at $1.96 per pound.
</body>
</html>
```

The financial wizards among you may have noticed that with some special software tricks on the server side, you can update the price of kumquats in the document so that it acts like a ticker-tape machine: the latest kumquat commodity price updated every 15 seconds.

13.2.2.2 Refreshing with a different document

Rather than reload the same document repeatedly, you can tell the browser to load a different document dynamically. You do so by adding that document's absolute URL after the delay time and an intervening semicolon in the `<meta>` tag's `content` attribute. For example:

```
<meta http-equiv="Refresh"
 content="15; URL=http://www.kumquat.com/next.html">
```

would cause the browser to retrieve the *next.html* document from the *www.kumquat.com* web server after having displayed the current document for 15 seconds.

The URL must be an absolute one, including server type and full pathname; relative URLs don't work.

13.2.2.3 Cycling among documents

Keep in mind that the effects of the Refresh `<meta>` tag apply only to the document in which it appears. Hence, to cycle among several documents, you must include a Refresh `<meta>` tag in each one. The `content` value for each document in the cycle must contain an absolute URL that points to the next document, with the last document pointing back to the first one to complete the cycle.

For example, the following are the <meta> tags for the headers of each in a three HTML-document cycle.

The document *first.html* contains:

```
<meta http-equiv="Refresh"
 content="30; URL=http://www.kumquat.com/second.html">
```

The document *second.html* contains:

```
<meta http-equiv="Refresh"
 content="30; URL=http://www.kumquat.com/third.html">
```

And the *third.html* document has in its <head> (besides other crazy ideas):

```
<meta http-equiv="Refresh"
 content="30; URL=http://www.kumquat.com/first.html">
```

If it is left alone, the browser will endlessly loop among the three documents at 30-second intervals.

Cycling documents make excellent attractors, catching the attention of passers-by to a web-driven kiosk, for example. Users may then navigate through the wider collection of kiosk documents by clicking hyperlinks in one of the kiosk's attractor pages and subsequent ones.[2]

> [2] This brings up a good point: the user may override the Refresh dynamic action at any time; for instance, by clicking a hyperlink before the client-pull timeout expires. The browser always ignores the Refresh action in lieu of user interaction.

To return to the cycling set of attractors, each document in the rest of the collection should have its own Refresh fields that eventually point back to the attractor. You should specify a fairly long delay period for the nonattractor pages - 120 to 300 seconds or more - so that the kiosk doesn't automatically reset while a user is reading the current document. However, the delay period should be short enough so that the kiosk resets to the attractor mode in a reasonable period of time after the user finishes.

13.2.3 Pulling Non-HTML Content

Netscape's and Internet Explorer's client-pull feature is not restricted to HTML documents, although it is certainly easiest to create dynamic documents with HTML. With a bit of server-side programming, you can add a Refresh field to the HTTP header of any sort of document from audio files to images to video clips.

For example, create a real-time video feed by adding a Refresh header field in each of a sequence of images grabbed and digitized from a camera. Include a delay of zero with the URL that points to the next image, so that as quickly as the browser displays one image, it retrieves the next. Assuming that the network keeps up, the result is a crude (really crude) TV.

Since the browser clears the window before presenting each subsequent image, the resulting flicker and flash make it almost impossible to present a coherent sequence of images. This technique is more effective when presenting a series of images designed to be viewed as a slide show, where the user expects some sort of display activity between each of the images.

Perhaps a better use of the client-pull feature is with long-playing multimedia documents for which Netscape and Internet Explorer use special helper applications to display. On a multitasking computer, such as one running Unix or Windows 98, the browser downloads one document, while a helper application plays another. Combine the client-pull capabilities with that multitasking to improve multimedia document performance. Rather than wait for a single, large document like a movie or audio file to download before playing, break it into smaller segments, each automatically downloaded by the previous segment via the Refresh header. The browser will play the first segment while downloading the second, then third, then fourth, and so on.

13.2.4 Combining Refresh with Other HTTP Header Fields

You can have your client-pull dynamic documents perform some neat tricks by combining the effects of the Refresh field with other HTTP header fields. One combination in particular is most useful: Refresh with a "Redirect" field.

The Redirect field lets the server tell the browser to retrieve the requested document elsewhere at the field's accompanying URL value. The client browser automatically redirects its request to the new URL and gets the document from the new location, usually without telling the user. We retrieve redirected documents all the time and may never notice.

The most common cause for redirection is when someone moves their HTML document collection to a new directory or to a new server. As a courtesy, the webmaster programs the original host server to send an HTTP header field containing the Redirect field and new URL (without a document body) to any and all browsers that request the document from the original location. That way, the new document location is transparent to users, and they won't have to reset their browser bookmarks.

But sometimes you want the user to reset their bookmarks to the new location because the old one won't be redirecting browsers forever, perhaps because it's being taken out of service. One way to notify users of the new location is to have the redirection URL point to some HTML document other than the home page of the new collection that contains a message about the new location. Once noted, users then take a "Continue" hyperlink to the new home page location and set their bookmarks accordingly.

By combining the Redirect and Refresh fields, you can make that notification screen automatically move to the new home page. If the browser receives an HTTP header with both fields, it will honor both; it immediately fetches the redirected URL and displays it, and it sets the refresh timer and replacement URL, if specified. When the time expires, the browser retrieves the next URL - your new home page location - automatically.

13.2.4.1 A random URL generator

Another application for the combination of Redirect and Refresh HTTP header fields is a perpetual, random URL generator. You'll need some programming skills to create a server-side application that selects a random URL from a prepared list and outputs a Redirect field that references that URL along with a Refresh field that reinvokes the random-URL application after some delay.

When Netscape or Internet Explorer receives the complete header, it immediately loads and displays the randomly selected document specified in the Redirect field's URL. After the delay specified in the Refresh field, the browser reruns the random-URL generator on the server (as specified in the Refresh URL), and the cycle starts over. The result is an endless cycle of random URLs displayed at regular intervals.

13.2.5 Performance Considerations

Client-pull documents consume extra network resources, especially when the refresh delay is small, since each refresh may involve a completely new connection to a server. It may take a browser several seconds to contact the server and begin retrieving the document. As a result, rapid updates generally are not feasible, especially over slow network connections.

Use client-pull dynamic documents for low-frequency updates of entire documents, or for cycling among documents without user intervention.

13.3 Server -Push Documents

Server-push dynamic documents are driven from the server side. The client-server connection remains open after an initial transfer of data, and the server periodically sends new data to the client, updating the document's display. Server-push is made possible by some special programming on the server side, and is enabled by the multipart mixed-media type feature of Multipurpose Internet Mail Extensions (MIME), the computer industry's standard for multimedia document transmission over the Internet.

Server-push documents currently are not supported by Internet Explorer.

13.3.1 The Multipart/Mixed Media Type

As we mentioned earlier in this chapter in the discussion of client-pull dynamic documents, the HTTP server sends a two-part transmission to the client browser: a header describing the document followed by the document itself. The document's MIME type is part of the HTTP header field. Normally, the server includes `Content-type: text/html` in an HTML document's header before sending its actual contents. By changing that content type to multipart/mixed-media, you can send an HTML document or several documents in several pieces, rather than in a single chunk. Only Netscape, though, understands and responds to the multipart header field; the other browsers either ignore additional parts or refuse the document altogether.

The general form of the MIME multipart mixed-media content type header looks like this:

```
Content-type: multipart/mixed;boundary="SomeRandomString"
```

This HTTP header component tells the Netscape client to expect the document to follow in several parts and to look for `SomeRandomString`, which separates the parts. That boundary string should be unique and not appear anywhere in any of the individual parts. The content of the server-to-client transmission looks like this:

```
--SomeRandomString
Content-type: text/plain

Data for the first part
--SomeRandomString
Content-type: text/plain

Data for the second part

--SomeRandomString--
```

The above example has two document parts, both plain text. The server sends each part preceded by our `SomeRandomString` document-boundary delimiter preceded by two dashes, followed by the `Content-type` field, and then the data for each part. The last transmission from server to client is a single reference to the boundary string followed by two more dashes indicating that this was the last part of the document.

Upon receipt of each part, the Netscape browser automatically adds the incoming data to the current document display.

You've got to write a special HTTP server application to enable this type of server-push dynamic document, one that creates the special HTTP MIME multipart/mixed header and sends the various documents separated by the boundary delimiter.

13.3.2 Multipart Mixed-Replace Media Type

Server-push dynamic document authors may use an experimental variant of the MIME multipart mixed-media content known as *multipart mixed-replace-media*. The difference between this special content-type and its predecessor is that rather than simply adding content to the current display, the "replace" version has each subsequent part replace the preceding one.

The format of the mixed-replace HTTP header is very similar to its multipart mixed counterpart; the only difference is in the `Content-type`:

```
multipart/x-mixed-replace;boundary=SomeRandomString
```

All other rules regarding the format of the multipart content are the same, including the boundary string used to separate the parts and the individual `Content-type` fields for each part of the content.

13.3.3 Exploiting Multipart Documents

It is easy to see how you can use the two special MIME multipart content types to create server-push dynamic documents. By delaying the time between parts, you might create an automatically scrolling message in the Netscape browser window. Or by replacing portions of the document through the x-mixed-replace MIME type, you might include a dynamic billboard in your document, perhaps even animation.

Note in particular that server-push multipart documents need not apply only to HTML or other plain text documents. Images, too, are a MIME-encoded content type, so you can have the HTTP server transmit several images in sequence as parts of a multipart transmission. Since you may also have each new image replace the previous one, the result is crude animation. Done correctly over a network of sufficient bandwidth, the effect can be quite satisfying.

13.3.3.1 Efficiency considerations

Server-push documents keep a connection open between the client and server for the duration of the dynamic document's activity. For some servers, this may consume extra network resources and may also require that several processes remain active, servicing the open connection. Make sure the server-push process (and, hence, the client-server connection) expire upon completion or after some idle period. Otherwise, someone will inadvertently camp on an endlessly cycling server-push document and choke off other users' access to the server.

Before choosing to implement server-push documents, make sure that your server can support the added processing and networking load. Keep in mind that many simultaneous server-push documents may be active, multiplying the impact on the server and seriously affecting overall server performance.

13.3.4 Creating a Server-Push Document

Create a special application that runs with the HTTP server to enable server-push dynamic documents. The application must create the special MIME `Content-type` header field that notifies the Netscape browser that the following document comes in several parts - added to or replacing a portion of the current document. The application must also create the appropriate boundary delimiter and send the `Content-type` header and data for each part, perhaps also delaying transmission of each part by some period of time. You will need to consult your server's documentation to learn how to create a server-side application that can be invoked by accessing a specific URL on the server. With some servers this may be as simple as placing the application in a certain directory on the server. With others, you may have to bend over backwards and howl at the moon on certain days.

13.3.4.1 Server-push example application for NCSA and Apache httpd

The NCSA and Apache *httpd* servers run on most Unix systems. Administrators usually configure the server to run server-side applications stored in a directory named *cgi-bin*.

The following is a simple Unix shell script that illustrates how to send a multipart document to a Netscape client via NCSA or Apache *httpd*:[3]

> [3] It is an idiosyncrasy of NCSA *httpd* that no spaces are allowed in the `Content-type` field that precedes your multipart document. Some authors like to place a space after the semicolon and before the boundary keyword. Don't do this with NCSA *httpd*; run the whole `Content-type` together without spaces to get the server to recognize the correct multipart content type.

```
#!/bin/sh
#
# Let the client know we are sending a multipart document
# with a boundary string of "NEXT"
#
echo "HTTP/1.0 200"
echo "Content-type: multipart/x-mixed-replace;boundary=NEXT"
echo ""
echo "--NEXT"
while true
do
#
# Send the next part, followed by a boundary string
# Then sleep five seconds before repeating
#
 echo "Content-type: text/html"
 echo ""
 echo "<html>"
 echo "<head>"
 echo "<title>Processes On This Server</title>"
 echo "</head>"
 echo "<body>"
 echo "<h3> Processes On This Server</h3>"
 echo "Date:"
 date
 echo "<p>"
 echo "<pre>"
 ps -el
 echo "</pre>"
 echo "</body>"
 echo "</html>"
 echo "--NEXT"
 sleep 5
done
```

In a nutshell, this example script updates a list of the processes running on the server machine every five seconds. The update continues until the browser breaks the connection by moving on to another document.

We offer this shell script example to illustrate the basic logic behind any server-push document generator. In reality, you should try to create your server-side applications using a more conventional programming language like Perl or C, for instance. The applications run more efficiently and can better detect when the client has severed the connection to the server.

Chapter 14. Netscape Layout Extensions

HTML was conceived in academia, not on Madison Avenue. It was originally intended to be an easy-to-use markup language to help people make their documents more readable through text embellishments like headers as well as more extensible to their own and others' work through hyperlinks, and include other media besides text - not for pizzazz, but to better explain and illustrate their work. HTML is not a page layout tool. Well, at least not at first.

As we discussed in Chapter 1, the language has evolved with the Internet itself. In particular, chances are probably a million to one that the web user is a college professor doing research, but a youngster surfing for a cool site or a buyer shopping for product information and the best deal. Commercial interests - the driving force of the Web today - demand increasingly complex page formats and visual displays to attract the ever-growing population of users, emphasizing look and feel over content.

From the start of their enterprise, the developers at Netscape have been at the forefront of browser design that addresses the needs of commercial interests. Throughout the years, they have consistently extended HTML to provide authors with far more sophisticated page layout capabilities than previous versions.

In this chapter, we cover three features that are unique to Netscape Navigator Versions 4[1] and earlier: spacers, multiple columns, and layers. These tags are seductive in the extreme, luring the designer away from the HTML standard with the promise of exciting new page layout capabilities. As always, we warn you to use these extensions with eyes wide open, since you run the risk of alienating a portion of your audience each time you elect to include these tags in your documents. Most importantly, these extensions won't become part of the XHTML standard and won't be supported by the forthcoming Version 6 of Netscape Navigator.

[1] As of this writing, Netscape insiders have suggested that none of these extensions will appear in Version 6.

More importantly, please keep in mind that you may achieve some of these same effects with standards. We describe Cascading Style Sheet standards in Chapter 8.

14.1 Creating Whitespace

One of the simplest elements in any page design is the empty space surrounding content. Empty space is often just as important to the look and feel of a page as the areas filled with text and images. Commonly known as *whitespace*, these empty areas shape and contain the content of your page.

Native HTML has no way to create empty space on your page, short of using a `<pre>` tag filled with blank lines or an empty image. In fact, browsers - acting according to the HTML standard - remove leading, trailing, and any other extra spaces in text and ignore extra linefeeds. Netscape fills this void with the `<spacer>` tag. Section 4.7.1

14.1.1 The <spacer> Tag

Use the `<spacer>` tag to create horizontal, vertical, and rectangular whitespace in your documents.

14.1.1.1 Creating horizontal space

The most common use of the `<spacer>` tag is to indent a line of text. To achieve this effect, set the value of the `type` attribute to `horizontal` and use the `size` attribute to define the width, in pixels (not text characters), of the horizontal area. For example:

```
<spacer type=horizontal size=100>
```

inserts 100 pixels of space in line with the current line of text. Netscape appends subsequent content at the end of the spacer if sufficient space remains on the current line. Otherwise, it places the next element onto the next line, following the normal word-wrap behavior used by Netscape.

If there is not enough room to place the entire `<spacer>` tag's whitespace on the current line, the browser shortens the space to fit on the current line. In a sense, the size of the spacer is soft, telling the browser to insert up to the specified number of pixels until the end of the current line is reached.

For example, if a spacer is 100 pixels wide, and there are only 75 pixels of space remaining on the current line within the browser's display window, Netscape will insert 75 pixels of space into the line and place the next element at the beginning of the next line in the display. Accordingly, a horizontal spacer will never be broken across a line, creating space at the end of one line and the beginning of the next.

<spacer>

Function:

Define a blank area in a document

Attributes:

ALIGN
HEIGHT
SIZE
TYPE
WIDTH

End tag:

None in HTML

Contains:

Nothing

Used in:

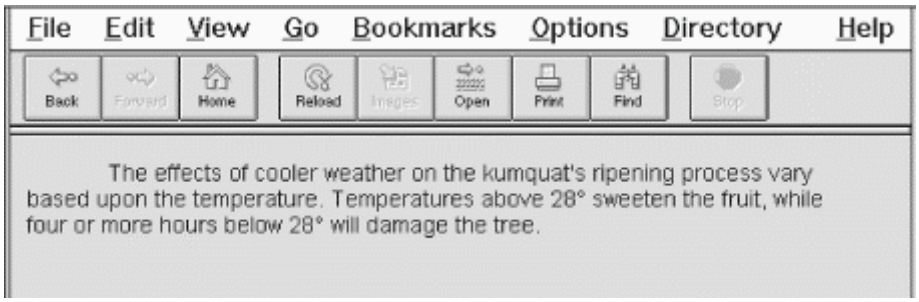
text

By far, the most common application of the horizontal spacer is to indent the first line of a paragraph. Simply place a horizontal spacer at the start of a paragraph to get the desired result:

```
<spacer type=horizontal size=50>
The effects of cooler weather on the kumquat's ripening process
vary based upon the temperature.  Temperatures above 28&deg;
sweeten the fruit, while four or more hours below 28&deg; will
damage the tree.
```

The results can be seen in Figure 14-1.

Figure 14-1. Indenting a paragraph with a horizontal spacer



Of course, you also can use horizontal spacers to insert additional space between letters or words in a line of text. This might be useful for displaying poetry or specialized ad copy. But don't use a spacer to create an indented block of text - you cannot predict the size of the user's browser window, font sizes, and so forth, and, hence, where it will break a particular line of text. Instead, use the `<blockquote>` tag or adjust the paragraph's left margin with an appropriate style.

14.1.1.2 Creating vertical space

You may insert extra whitespace between lines of text and paragraphs in your documents by setting the `type` attribute in the `<spacer>` tag to `vertical`. The `size` attribute must also be included. Make its value a positive integer equal to the amount of whitespace, in pixels.

The vertical spacer acts just like the
 tag. Both tags cause an immediate line break. The difference is, of course, that with the vertical spacer you control how far below the current line of text Netscape should start the subsequent line. The white space is added to - and therefore is never less than - the normal amount of space that would appear below the current line of text as a result of the paragraph's line spacing.

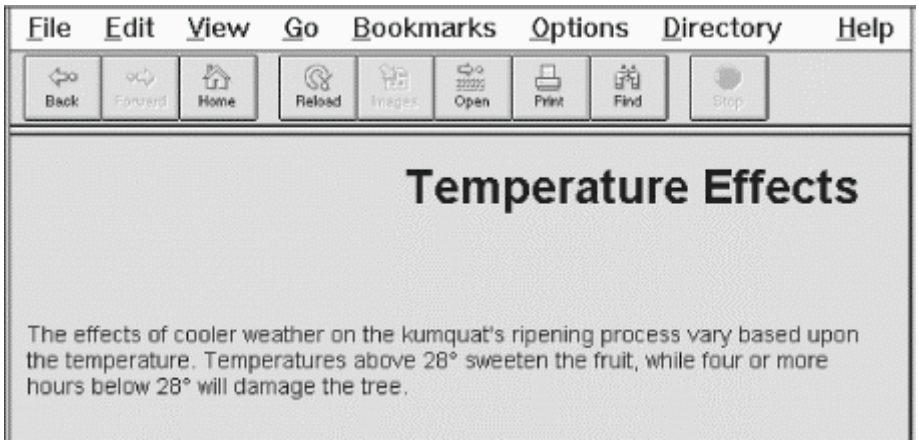
Since HTML pages are infinitely tall, the vertical space may be any number of pixels high. Of course, it'd be sophomoric to be excessive (oh, okay, try size=100000000). Most of today's monitors have a vertical scan of no more than 1024 lines. So a vertical pixel size value of 1025 ensures that the next line of text will be placed off the user's screen, if that is the effect you desire.

Vertical spacers aren't quite as common as horizontal spacers, but they can still be useful. In the following text, we've used a vertical spacer to provide a bit more separation between the document's header and the regular text:

```
<h1 align=right>Temperature Effects</h1>
<spacer type=vertical size=50>
The effects of cooler weather on the kumquat's ripening process
vary based upon the temperature.  Temperatures above 28&deg;
sweeten the fruit, while four or more hours below 28&deg; will
damage the tree.
```

The results can be seen in Figure 14-2.

Figure 14-2. Using a vertical spacer to separate a header from the text



14.1.1.3 Creating blocks of space

The third spacer type creates a rectangular block of blank space, much like a blank image. Set the type attribute to block and include three other attributes to fully define the space: width, height, and align.

The width and height attributes specify the size of the spacer in pixels or as a percentage of the element containing the spacer. These attributes are used only when the type attribute is set to block and are otherwise ignored. Similarly, the size attribute is ignored when the <spacer> type is block. If specifying a size in pixels, you must give a positive integer value to both the width and height attributes; their default value is zero.

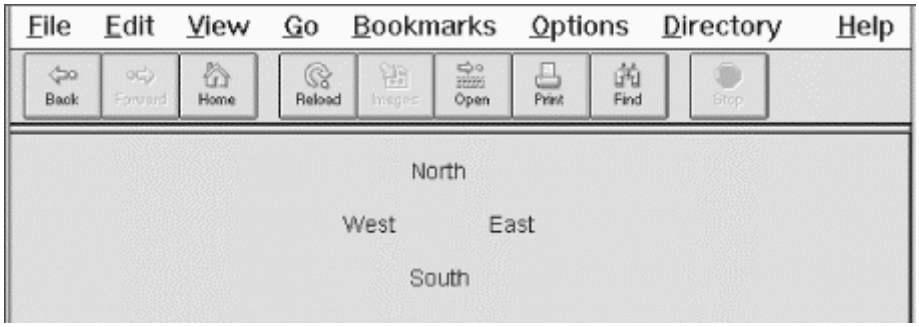
The third required spacer block attribute, align , controls how Netscape places the empty block relative to the surrounding text. The values for this attribute are identical to those for the align attribute in the <img> tag. Use the top, texttop, middle, absmiddle, baseline, bottom, and absbottom values to obtain the desired vertical alignment of the block spacer. Use the left and right values to force the block spacer to the indicated margin and cause the following text to flow up and around the spacer. The default value is bottom. For a complete description of the align attribute and its values, see Section 5.2.6.4.

This HTML fragment places the compass points around an empty area:

```
<center>
North
<br>
West
<spacer type=block width=50 height=50 align=absmiddle>
East
<br>
South
</center>
```

The resulting document is shown in Figure 14-3.

Figure 14-3. Using a block spacer to create space in a document



14.1.2 Mimicking the <spacer> Tag

Since only Netscape prior to Version 6 supports the `<spacer>` tag, other browsers ignore it, ruining your carefully contrived layout. But it is possible to completely emulate the `<spacer>` tag using the `<img>` tag and a special, small image. And, as we mentioned earlier, standard style sheet properties also let you do much of what `<spacer>` does, in a more orderly and comprehensive fashion, too.

For an image to emulate `<spacer>`, you'll need a GIF image that is completely transparent. Since no part of the image will ever be seen, you can make it as small as you'd like; we recommend a 1 x 1 pixel GIF image. In the following examples, our tiny 1 x 1 pixel transparent image is named *small.gif*.

To emulate a horizontal spacer of the form:

```
<spacer type=horizontal size=n>
```

use this `<img>` tag:

```
<img src=small.gif width=n height=1>
```

Replace *n* with the desired pixel width, of course. Keep in mind, however, that the width of the `<img>` tag is fixed and may not integrate into the text flow exactly like the `<spacer>` tag would, especially if the `<img>` tag falls at or near the end of a line of text.

To emulate a vertical spacer of the form:

```
<spacer type=vertical size=n>
```

use this HTML fragment:

```
<br>
<img src=small.gif width=1 height=n>
<br>
```

The `<br>` tags are needed in the example to emulate the line-breaking behavior of the vertical spacer. Again, replace *n* with the desired height.

To emulate a block spacer of the form:

```
<spacer type=block width=w height=h align=a>
```

use this `<img>` tag:

```
<img src=small.gif width=w height=h align=a>
```

Replace *w*, *h*, and *a* with the desired width, height, and alignment values.

Given that simple replacements exist for all the variants of the `<spacer>` tag, you might wonder why it is needed at all. If nothing else, the `<spacer>` tag will render faster, since the equivalent `<img>` tag will require that an image be retrieved from a server and scaled before it is inserted into your text. While this is a small issue for most users, there may be cases where using the `<spacer>` tag in lieu of an `<img>` tag makes sense.

14.2 Multicolumn Layout

Multicolumn text formatting is one of the most common features of desktop publishing. In addition to creating attractive pages in a variety of formats, multiple columns let you present your text using shorter, easier-to-read lines. HTML page designers have longed for the ability to easily create multiple text columns in a single page, but have been forced to use various tricks, such as multicolumn tables (see Chapter 17).

Netscape has neatly solved this problem with the unique <multicol> tag. While fancy unbalanced columns and straddling are not possible with this tag, as they are with tables, conventionally balanced text columns are easy to create with <multicol>. And while this capability is available only with Netscape, the <multicol> tag degrades nicely in other browsers.

14.2.1 The <multicol> Tag

The <multicol> tag creates multiple columns of text and lets you control the size and number of columns.

<multicol>

Function:

Format text with multiple columns

Attributes:

CLASS
COLS
GUTTER
STYLE
WIDTH

End tag:

</multicol>; always used

Contains:

body_content

Used in:

block

The <multicol> tag can contain any other HTML content, much like the tag. All of the content within the <multicol> tag is displayed just like conventional content, except that Netscape places the contents into multiple columns instead of just one.

The <multicol> tag creates a break in the text flow and inserts a blank line before rendering its content into multiple columns. After the tag, another blank line is added and the text flow resumes using the previous layout and formatting.

Netscape automatically balances the columns, making each approximately the same length. Where possible, the browser moves text between columns to accomplish the balancing. In some cases, the columns cannot be perfectly balanced because of embedded images, tables, or other large elements.

You can nest <multicol> tags, embedding one set of columns within another set of columns. While infinite nesting is supported, more than two levels of nesting is generally impractical and results in unattractive text flows.

14.2.1.1 The cols attribute

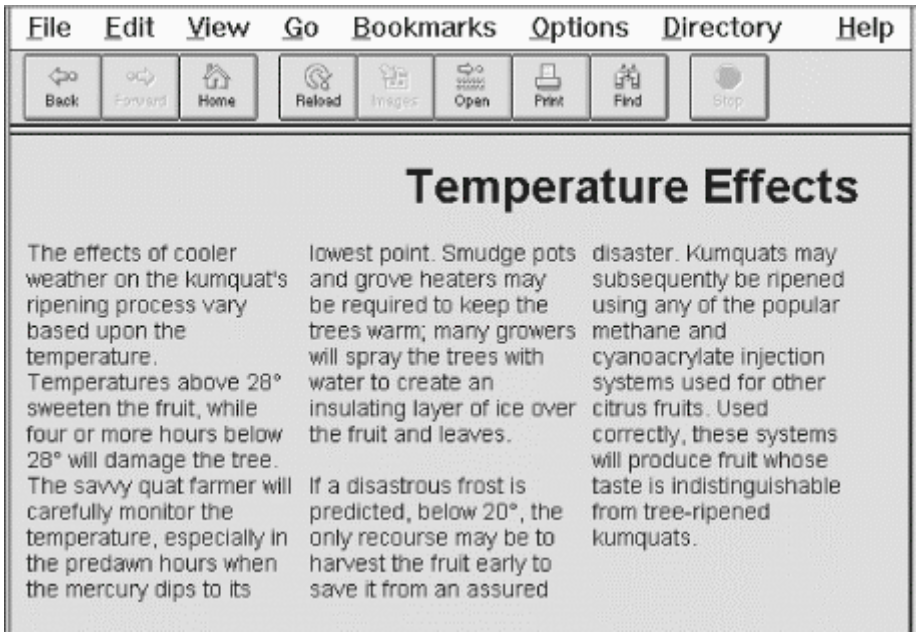
The cols attribute is required by the <multicol> tag to define the number of columns. If omitted, Netscape creates just one column, as if the <multicol> tag isn't there at all. You may create any number of columns, but in practice, more than three or four columns make text unreadable on most displays.

The following example creates a three-column layout:

```
<h1 align=right>Temperature Effects</h1>
<multicol cols=3>
The effects of cooler weather on the kumquat's ripening process
vary based upon the temperature.  Temperatures above 28&deg;
sweeten the fruit, while four or more hours below 28&deg; will
damage the tree.  The savvy quat farmer will carefully monitor
the temperature, especially in the predawn hours when the
mercury dips to its lowest point.  Smudge pots and grove heaters
may be required to keep the trees warm; many growers will spray
the trees with water to create an insulating layer of ice over
the fruit and leaves.
<p>
If a disastrous frost is predicted, below 20&deg;, the only
recourse may be to harvest the fruit early to save it from an
assured disaster.  Kumquats may subsequently be ripened using
any of the popular methane and cyanoacrylate injection systems
used for other citrus fruits.  Used correctly, these systems will
produce fruit whose taste is indistinguishable from tree-ripened
kumquats.
</multicol>
```

The results are shown in Figure 14-4.

Figure 14-4. A three-column <multicol> document segment



You can see in Figure 14-4 how Netscape has balanced the columns to approximately equal lengths. You also can see how several lines within the columns appear shorter, since longer words were wrapped to the next line of text. These overly ragged right margins within the columns are unavoidable and serve to emphasize that you shouldn't create more than four or five columns in a flow. Our example is still barely readable if displayed as five columns; it breaks down completely and even induces rendering errors if `cols` is set to 7, as shown in Figure 14-5.

14.2.1.2 The gutter attribute

The space between columns is known as the *gutter*. By default, Netscape creates a gutter ten pixels wide between each of your columns. To change this, set the `gutter` attribute's value to the desired width in pixels. Netscape will reserve this much space between your columns; the remaining space will be used for the columns themselves.

Figure 14-6 shows the effect this can have on your columns. In this figure, we've reformatted our sample text using `<multicol cols=3 gutter=50>`. Contrast this with Figure 14-4, which uses the default ten-pixel gutters.

Figure 14-5. Too many columns create unreadable pages

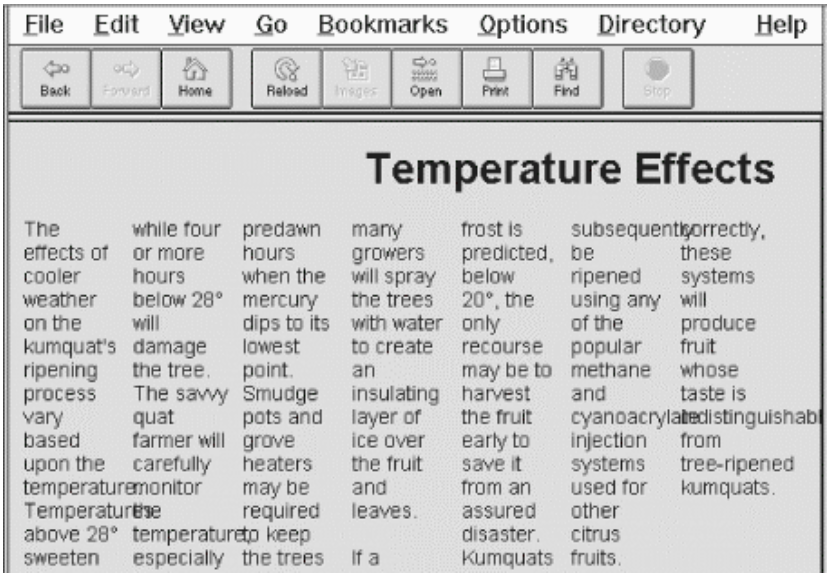
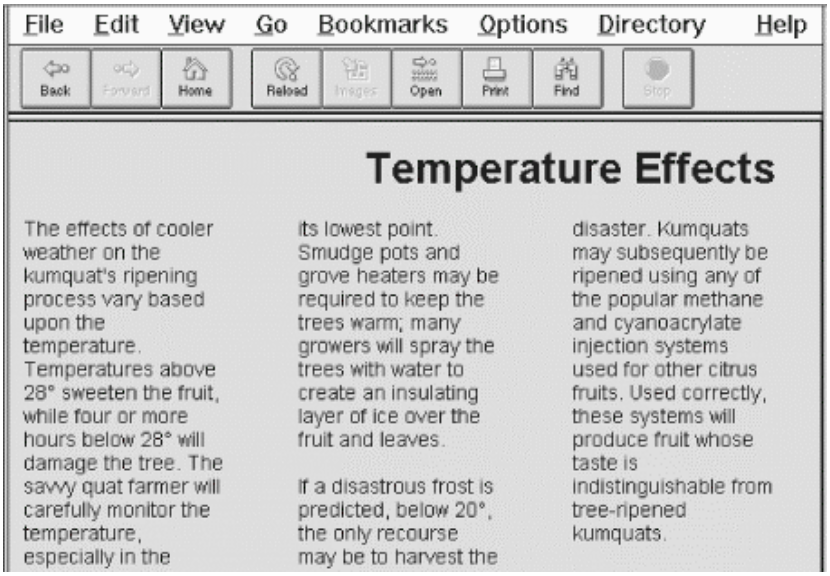


Figure 14-6. Change gutter widths with the <multicol> gutter attribute



14.2.1.3 The width attribute

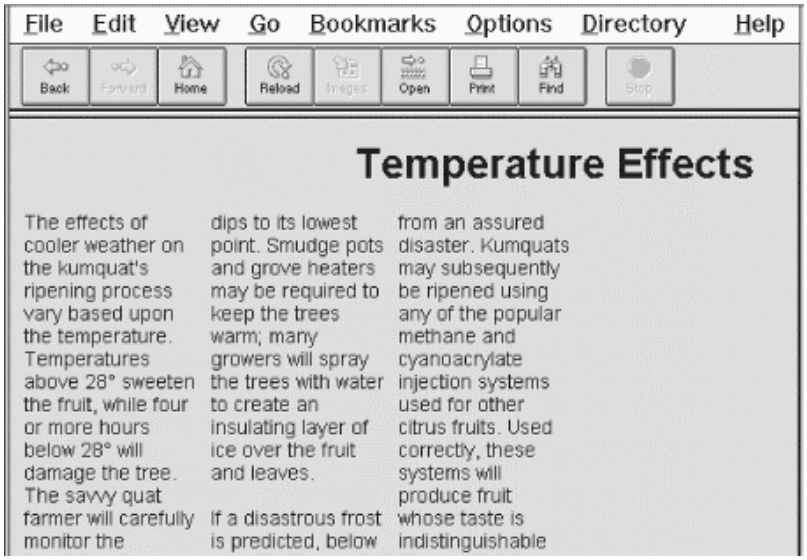
Normally, the `<multicol>` tag fills the current width of the current text flow. To have your multiple columns occupy a thinner space, or to extend them beyond the visible window, use the `width` attribute to specify the overall width of the `<multicol>` tag. The columns will be resized so that the columns plus the gutters fill the width you've specified.[2] The width may be specified as an absolute number of pixels or as a percentage of the width of the current text flow.

[2] To be exact, each column will be $(w\text{-}g\ (n\text{-}1))/n$ pixels wide, where w is the width of the `<multicol>`tag, g is the width of a gutter, and n is the number of columns. Thus, using `<multicol cols=3 gutter=10 width=500>` creates columns that are 160 pixels wide.

Figure 14-7 shows the effects of adding `width="75%"` to our column example, retaining the default gutter width of ten pixels.

Be careful when you reduce the size of your columns if they include images or other fixed-width elements. Netscape will not wrap text around images that extend beyond the boundaries of a column. Instead, the image covers the adjacent columns, ruining your document. Always make sure embedded elements in columns are small enough to fit within your columns, even on fairly small browser displays.

Figure 14-7. Changing the width of <multicol> columns



14.2.1.4 The style and class attributes

Use the `style` attribute with the `<multicol>` tag to create an inline style for all the content inside the tag. The `class` attribute lets you label the section with a name that refers to a predefined class of the `<multicol>` tag declared in some document-level or externally defined style sheet. Section 8.1.1 / Section 8.3.

14.2.2 Multiple Columns and Other Browsers

As we've noted, the `<multicol>` tag is supported only by Netscape Navigator prior to Version 6. Fortunately, when other browsers encounter the `<multicol>` tag, they ignore it and render the enclosed text as part of the normal text flow, usually with little consequent disruption to the document.

The only problem may be that the contents of the `<multicol>` tag flow up into the previous flow, without an intervening break. For that reason, you might consider preceding every `<multicol>` tag with a `<p>` tag. Netscape won't mind, and other browsers will at least perform a paragraph break before rendering your multicolumn text in a single column.

It is possible to emulate the `<multicol>` tag using tables, but the results are crude and difficult to manage across multiple browsers. To do so, create a single row table with a cell for each column. Place an appropriate amount of the text flow in each cell to achieve balanced columns. The difficulty, of course, is that the "appropriate amount" varies wildly between browsers, making it almost impossible to create multiple columns that are attractive on several different browsers.

If you must have multiple columns, and can tolerate your columns reverting to a single column on incompatible browsers, we recommend you use `<multicol>`.

14.2.3 Effective Multicolumn Layouts

We've offered advice on columns throughout these sections. Here is a quick recap of our tips for creating effective column layouts:

- Use a small number of columns.
- Don't use excessively wide gutters.
- Ensure that embedded elements like images and tables fit in your columns on most displays.
- Precede each `<multicol>` tag with a `<p>` tag to improve your document's appearance on other browsers.
- Avoid nesting `<multicol>` tags more than two deep.

14.3 Layers

Spacers and multiple columns are natural extensions to conventional HTML, existing within a document's normal flow. With Navigator Version 4, Netscape took HTML into an entirely new dimension with layers. It transforms the single-element document model into one containing many layered elements that are combined to form the final document. Regrettably, layer support will be omitted from Netscape 6.

Layers supply the layout artist with a most critical element missing in standard HTML: absolute positioning of content within the browser window. Layers let you define a self-contained unit of HTML content that can be positioned anywhere in the browser window, placed above or below other layers, and made to appear and disappear as you desire. Document layouts that were impossible with conventional HTML are trivial with layers.

If you think of your document as a sheet of paper, layers are like sheets of clear plastic placed atop your document. For each layer, you define the content of the layer, its position relative to the base document, and the order in which it is placed on the document. Layers can be transparent or opaque, visible or hidden, providing an endless combination of layout options.

14.3.1 The <layer> Tag

HTML document content layers are each defined with the `<layer>` tag. A layer can be thought of as a miniature HTML document whose content is defined between the `<layer>` and `</layer>` tags. Alternatively, the content of the layer can be retrieved from another HTML document by using the `src` attribute with the `<layer>` tag.

<layer>

Function:

Define a layer of content within a document

Attributes:

ABOVE	NAME
BACKGROUND	SRC
BELOW	STYLE
BGCOLOR	TOP
CLASS	VISIBILITY
CLIP	WIDTH
LEFT	Z-INDEX

End tag:

</layer>; always used

Contains:

body_content

Used in:

block

Regardless of its origin, Netscape formats a layer's content exactly like a conventional document, except that the result is contained within that separate layer, apart from the rest of your document. You control the position and visibility of this layer using the attributes of the `<layer>` tag.

Layers may be nested, too. Nested layers move with the containing layer and are visible only if the containing layer itself is visible.

14.3.1.1 The name attribute

If you plan on creating a layer and never referring to it, you needn't give it a name. However, if you plan to stack other layers relative to the current layer, as we demonstrate later in this chapter, or modify your layer using JavaScript, you'll need to name your layers using the `name` attribute. The value you give `name` is a text string, whose first character must be a letter, not a number or symbol.

Once named, you can refer to the layer elsewhere in the document, and change it while the user interacts with your page. For example, this bit of HTML:

```
<layer name="warning" visibility=hide>
Warning!  Your input parameters were not valid!
</layer>
```

creates a layer named `warning` that is initially hidden. If in the course of validating a form using a JavaScript routine, you find an error and want to display the warning, you would use the command:

```
warning.visibility = "show";
```

Netscape then makes the layer visible to the user.

14.3.1.2 The left and top attributes

Without attributes, a layer gets placed in the document window as if it were part of the normal document flow. Layers at the very beginning of a document get put at the top of the Netscape window; layers that are between conventional document content get placed in line with that content.

The power of layers, however, is that you can place them anywhere in the document. Use the `top` and `left` attributes for the `<layer>` tag to specify its absolute position in the document display.

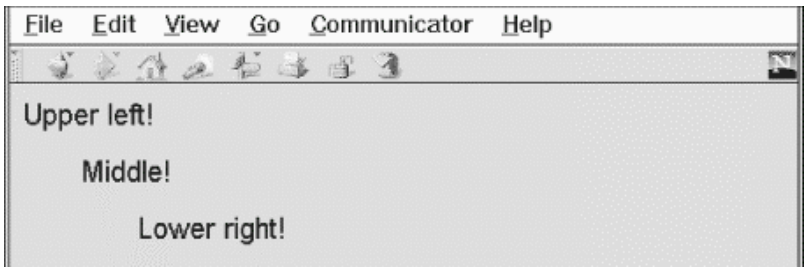
Both attributes accept an integer value equal to the number of pixels offset from the top left (0,0) edge of the document's display space or, if nested inside another layer, the containing layer's display space. As with other document elements whose size or position extends past the edge of the browser's window, Netscape gives the user scrollbars to access layered elements outside the current viewing area.

Here is a simple layer example that staggers three words diagonally down the display—not something you can do easily, and certainly not with the same precision, in conventional HTML:

```
<layer left=10 top=10>
  Upper left!
</layer>
<layer left=50 top=50>
  Middle!
</layer>
<layer left=90 top=90>
  Lower right!
</layer>
```

The result is shown in Figure 14-8.

Figure 14-8. Simple text positioning with the <layer> tag



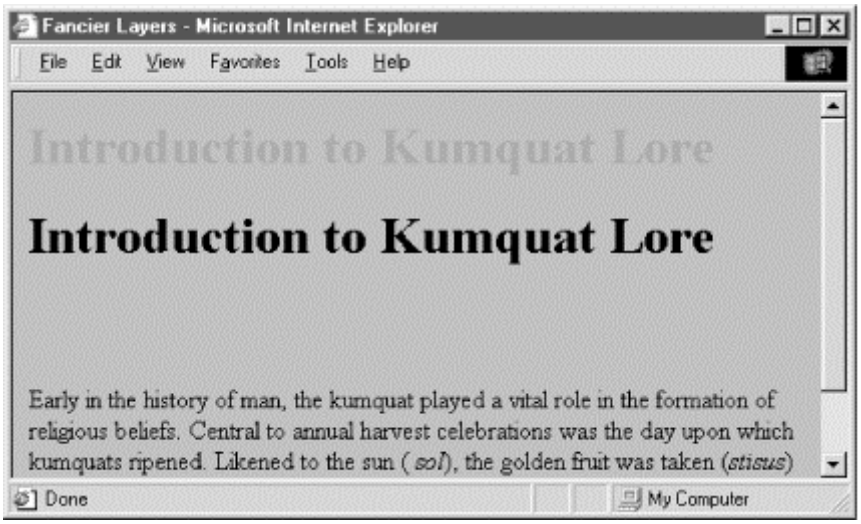
Admittedly, this example is a bit dull. Here's a better one that creates a drop shadow behind a heading:

```
<layer>
  <layer left=2 top=2>
    <h1><font color=gray>Introduction to Kumquat Lore</font></h1>
  </layer><layer left=0 top=0>
    <h1>Introduction to Kumquat Lore</h1>
  </layer>
</layer>
<h1> </h1>
Early in the history of man, the kumquat played a vital role in the
formation of religious beliefs. Central to annual harvest celebrations
was the day upon which kumquats ripened. Likened to the sun (<i>
sol</i>), the golden fruit was taken (<i>stisus</i>) from the trees on
the day the sun stood highest in the sky.  We carry this day forward
even today, as our summer <i>solstice</i>.
```

Figure 14-9 shows the result. Figure 14-10 demonstrates what happens with layers when viewed with a browser other than Netscape.

Figure 14-9. Creating drop shadow effects with multiple layers

Figure 14-10. Internet Explorer doesn't support multiple layers



We used a few tricks to create the drop shadow effect for the example header. First, Netscape covers layers created earlier in the document by later layers. Hence, we create the gray shadow first, followed by the actual heading, so that it appears on top, above the shadow. We also enclosed these two layers in a separate containing layer. This way, the shadow and header positions are relative to the containing layer, not the document itself. The containing layer, lacking an explicit position, gets placed into the document flow as if it were normal content and winds up where a conventional heading would appear in the document.

Normal content, however, still starts at the top of the document and could end up behind the fancy heading in our example. To push content below our layered heading, we include an empty heading (save for a nonbreaking space—` `) before including our conventional document text.

This is important enough to repeat: normal document content following a `<layer>` tag is positioned directly under the layer it follows. This effect can be circumvented using an inline layer, described in Section 14.3.2.

14.3.1.3 The above, below, and z-index attributes

Layers exist in three dimensions, occupying space on the page and stacked atop one another as well as on top of conventional document content. As we mentioned earlier, layers normally get stacked in order of their appearance in the document: layers at the beginning get covered by later layers in the same display area.

You can control the stacking order of the layers with the `above`, `below`, and `z-index` attributes for the `<layer>` tag. These attributes are mutually exclusive; use only one per layer.

The value for the `above` or `below` attribute is the name of another layer in the current document. Of course, that referenced layer must have a `name` attribute whose value is the same name you use with the `above` or `below` attribute in the referring `<layer>` tag. You also must have created the referenced layer earlier in the document; you cannot refer to a layer that comes later.

In direct contradiction with what you might expect, Netscape puts the current layer below the above named layer, and above the below named layer.[3] Oh, well. Note the layers must occupy the same display space for you to see any effects.

[3] One cannot help but imagine that the above and below attributes were implemented in the wee hours.

Let's use our drop shadow layer example again to illustrate the above attribute:

```
<layer>
  <layer name=text left=0 top=0>
    <h1>Introduction to Kumquat Lore</h1>
  </layer>
  <layer name=shadow above=text left=2 top=2>
    <h1><font color=gray>Introduction to Kumquat Lore</font></h1>
  </layer>
</layer>
```

The above attribute in the layer named shadow tells Netscape to position the shadow layer so that the layer named text is above it. The effect is identical to Figure 14-9.

The above and below attributes can get confusing when you stack several layers. We find it somewhat easier to use the z-index attribute for keeping track of which layers go over which. With z-index, you specify the order in which Netscape stacks the layers: higher z-index value layers get put on top of lower z-index value layers.

For example, to create our drop shadow using the z-index attribute:

```
<layer>
  <layer left=0 top=0 z-index=2>
    <h1>Introduction to Kumquat Lore</h1>
  </layer>
  <layer left=2 top=2 z-index=1>
    <h1><font color=gray>Introduction to Kumquat Lore</font></h1>
  </layer>
</layer>
```

Again, the effect is identical to Figure 14-9. Normally, Netscape would display the second layer—the gray one in this case—on top of the first layer. But since we've given the gray layer a lower z-index value, it is placed behind the first layer.

The z-index values need not be sequential, although they must be integers, so we could've used the values 99 and 2, respectively, and gotten the same result in the previous example. And you need not specify a z-index for all the layers that occupy the same display space—only those you want to raise or lower in relation to other layers. However, be aware that the order of precedence may get confusing if you don't z-index all related layers.

For instance, what order of precedence by color would you predict when Netscape renders the following sequence of layers?

```
<layer left=0 top=0 z-index=3>
  <h1><font color=red>Introduction to Kumquat Lore</font></h1>
</layer>
<layer left=4 top=4>
  <h1><font color=green>Introduction to Kumquat Lore</font></h1>
</layer>
<layer left=8 top=8 z-index=2>
  <h1><font color=blue>Introduction to Kumquat Lore</font></h1>
</layer>
```

Give yourself a star if you said that the green header goes on top of the red header which goes on top of the blue header. Why? Because the red header is of lower priority than the green header based on order of appearance, and we forced the blue layer below the red one by giving it a lower z-index value. Netscape displays z-indexed layers according to their given order and non-z-indexed layers according to their order of appearance in the document. Precedence based on order of appearance also applies for layers that have the same z-index value. If you nest layers, all the layers at the same nesting level get ordered according to their z-index attributes. This group is then ordered as a single layer among all the layers at the containing level. In short, layers nested within a layer cannot be interleaved among layers at a different level.

For example, consider these nested layers, with their content and end tags omitted for clarity (indentation indicates nest level):

```
<layer name=a z-index=20>
  <layer name=a1 z-index=5>
  <layer name=a2 z-index=15>
<layer name=b z-index=30>
  <layer name=b1 z-index=10>
  <layer name=b2 z-index=25>
  <layer name=b3 z-index=20>
<layer name=c z-index=10>
```

Layers a, b, and c are at the same level, with layers a1 and a2 nested with a, and b1, b2, and b3 nested within b. Although the z-index numbers might at first glance appear to cause Netscape to interleave the various nested layers, the actual ordering of the layers, from bottom to top, is c, a, a1, a2, b, b1, b3, and b2.

If two layers are nested within the same layer and they have the same `z-index` value, the layer defined later in the document is placed on top of the previously defined layer.[4]

[4] This, of course, applies to layers inside the same containing nest only.

14.3.1.4 The background and bgcolor attributes

Like the corresponding attributes for the `<body>` tag, you may define the background color and an image for a Netscape layer with the `bgcolor` and `background` attributes, respectively.[5] By default, the background of a layer is transparent, allowing lower layers to show through.

[5] Note that you may also control the background color as well as many other display features of not just a single tag but all `<layer>` tags within your document using style sheets. See Section 14.3.1.9.

The `bgcolor` attribute accepts a color name or RGB triplet as its value, as defined in Appendix G. If specified, Netscape sets the entire background of the layer to this color, rendering the layer opaque. This attribute is handy for creating a colored box behind text, as a highlighting or attention-getting mechanism. It will, however, hide any layers below it, including conventional HTML content.

The `background` attribute accepts the URL of an image as its value. The image is tiled to fill the area occupied by the layer. If portions of the image are transparent, those portions of the layer will be transparent, and underlying layers will show through.

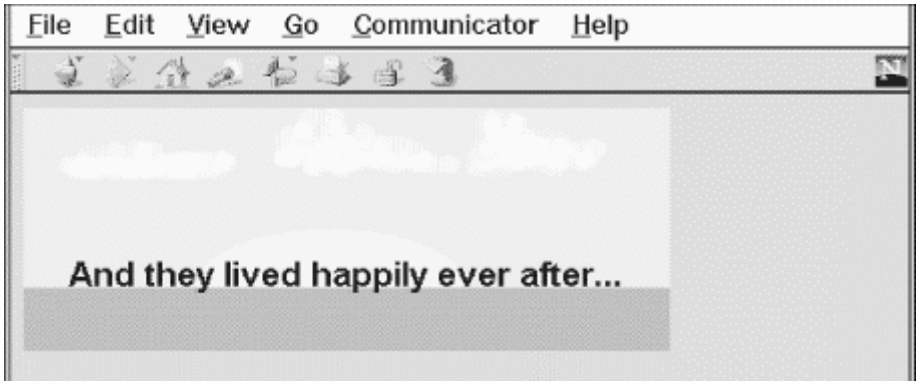
If you include both attributes, the background color will show through the transparent spots in the background image. The whole layer will be opaque.

The `background` attribute is useful for placing a texture behind text, but it fails miserably when the goal is to render text in front of a fixed image. Since the size of a layer is dictated by its contents, not the background image, using the image as the background will cause it to be clipped or tiled, depending on the size of the text. To place text reliably atop an image, use one layer nested within another:

```
<layer>
  <img src="sunset.gif">
  <p>
  <layer top=75>
    <h2 align=center>And they lived happily ever after...</h2>
  </layer>
</layer>
```

Netscape sets aside space for the entire image in the outer layer. The inner layer occupies the same space, except that we shift it down 75 pixels to align the text better over the image. The result is shown in Figure 14-11.

Figure 14-11. Placing text over an image using layers



14.3.1.5 The visibility attribute

Layers, by default, are usually seen (albeit not heard). You can change that by setting the `visibility` attribute to `show`, `hide`, or `inherit`. As expected, `show` forces the layer to be seen, `hide` hides it from view, and `inherit` explicitly declares that you want the layer to inherit its parent's visibility. The default value for this attribute is `inherit`. Layers that are not nested are considered to be children of the main document, which is always visible. Thus, non-nested layers lacking the `visibility` attribute are initially visible.

It makes little sense to hide layers unless you plan to reveal them later. In general, this attribute is used only when you include some JavaScript routines with your document that will reveal the hidden layers as a result of some user interaction. Section 12.3.3

Layers that are hidden do not block layers below them from view. Instead, a hidden layer can best be thought of as being transparent. One way to hide content in the main document is to place an opaque layer over the content. To display the hidden context, hide the opaque layer, revealing the content underneath.

14.3.1.6 The width attribute

Layers are only as big as necessary to contain their content. The initial width of a layer is defined to be the distance from the point at which the layer is created in the current text flow to the right margin. Netscape then formats the layer's contents to that width and makes the height of the layer tall enough to contain all of the layer's contents. If the contents of the layer wind up smaller than the initial width, the layer's width is then reduced to this smaller amount.

You can explicitly set the width of a layer using the `width` attribute. The value of this attribute defines the width of the layer in pixels or as a percentage of the containing layer. As expected, Netscape then sets the height based upon the size of the layer's contents, wrapped to the specified width. If elements in the layer—such as images—cannot be wrapped and instead extend past the right margin of the layer, only a portion of the element will be shown. The remainder will be clipped by the edge of the layer and not shown. This is similar to the behavior of an image in the main document window. If the image extends beyond the edge of the browser window, only a portion of the image is displayed. Unlike the browser window, however, layers cannot sport scrollbars allowing the user to scroll around in the layer's contents.

14.3.1.7 The src attribute

The contents of a layer are not restricted to what you type between its `<layer>` and `</layer>` tags; you can also refer to and automatically load the contents of another document into the layer with the `src` attribute. The value of the `src` attribute is the URL of the document containing the layer's content.

Note that the `layer src`'d document should *not* be a full-fledged HTML document. In particular, it should not contain `<body>` or `<head>` tags, although other HTML content is allowed.

You can combine conventional layer content with content taken from another file by using both the `src` attribute and placing content within the `<layer>` tag. In this case, the content from the file is placed in the layer first, followed by any inline content within the tag itself. If you choose to use the `src` attribute without supplying additional inline content, you still must supply the closing `</layer>` tag to end the definition of the layer.

The `src` attribute provides, for the first time, a source inclusion capability in HTML. Previously, to insert content from one HTML document within another, you had to rely on a server-based capability to read the other file and insert it into your document at the correct location. Since layers are positioned, by default, at their defining point within the current flow, including another file in your document is very simple:

```
...other content
<layer src="boilerplate"></layer>
...more content
```

Since a layer is rendered as a separate HTML entity, the contents of the included file will not be flowed into the containing text. Instead, it is as if the inserted text were contained within a `<div>` tag or other block-level HTML element.

14.3.1.8 The clip attribute

Normally, users see the entire layer unless it is obscured by a covering layer. With the clip attribute, you can mask off portions of a layer, revealing only a rectangular portion within the layer.

The value of the clip attribute is two or four integer values, separated by commas, defining pixel offsets into the layer corresponding to the left, top, right, and bottom edges of the clip area. If only two values are supplied, they correspond to the right and bottom edges of the visible area, and Netscape assumes the top and left values are zero. Therefore, `clip="75,100"` is equivalent to `clip="0,0,75,100"`.

The area of the layer outside the visible area is made transparent, allowing whatever is under the layer to show through.

The `clip` attribute is handy for hiding portions of a layer, or for creating fade and wipe effects using JavaScript functions to change the clipping window over time.

14.3.1.9 The style and class attributes

Use the `style` attribute with the `<layer>` tag to create an inline style for all the content inside a layer. The `class` attribute lets you label the layer with a name that refers to a predefined class of the `<layer>` tag declared in some document-level or externally defined style sheet. Accordingly, you may choose to use a style sheet instead of individual and redundant `bgcolor` tag attributes to define a background color for all your document layers or for a particular class of layers. Section 8.1.1 / Section 8.3.

14.3.2 The <ilayer> Tag

While you control the position of a `<layer>` using `top` and `left` attribute coordinates relative to the document's entire display space, Netscape provides a separate tag, `<ilayer>`,that lets you position individual layers with respect to the current flow of content, much like an inline image.

<ilayer> N

Function:

Define an inline layer of content within a text flow

Attributes:

ABOVE	NAME
BACKGROUND	SRC
BELOW	STYLE
BGCOLOR	TOP
CLASS	VISIBILITY
CLIP	WIDTH
LEFT	Z-INDEX

End tag:

</ilayer>; always used

Contains:

body_content

Used in:

text

An `<ilayer>` tag creates a layer that occupies space in the containing text flow. Subsequent content is placed after the space occupied by the `<ilayer>`. This is in contrast to the `<layer>` tag, which creates a layer above the containing text flow, allowing subsequent content to be placed under the layer just created.

The `<ilayer>` tag removes the need for an enclosing, attribute-free `<layer>` that serves to put a nest of specially positioned layers inline with the content flow, much like we did in most of the examples in the previous sections of this chapter. The attributes of the `<ilayer>` are the same as those for the `<layer>` tag.

14.3.2.1 The top and left attributes

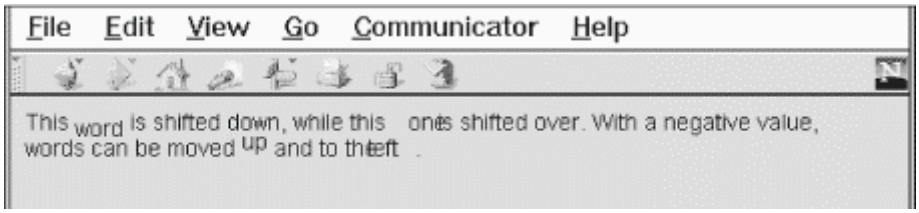
The only attributes that distinguish the actions of the `<ilayer>` tag from its `<layer>` sibling are the `top` and `left` attributes: Netscape renders `<ilayer>` content directly in the containing text flow, offset by the `top` and `left` attribute values from the upper-left corner of that inline position—not the document's upper-left display corner, as with `<layer>`. Netscape will also accept negative values for the `top` and `left` attributes of the `<ilayer>` tag, letting you shift the contents above and to the left of the current flow.

For example, to subscript, superscript, or shift words within the current line, you could use:

```
This <ilayer top=4>word</ilayer> is shifted down, while
this <ilayer left=10>one</ilayer> is shifted over.  With a negative
value, words can be moved <ilayer top=-4>up</ilayer> and to
the <ilayer left=-10>left</ilayer>.
```

The resulting effects are shown in Figure 14-12. Notice how the shifted words overlap and obscure the surrounding text. Netscape makes no effort to make room for the shifted elements; they are simply placed in a different spot on the page.

Figure 14-12. Moving inline layers with respect to the adjacent text



14.3.2.2 Combining <layer> and <ilayer>

Anything you can create with a regular layer can be used within an inline layer. However, do bear in mind always that the `top` and `left` attribute offsets are indeed from the `<ilayer>` content's allotted position, not from the document display space. Accordingly, use `<ilayer>` to position content inline with the conventional HTML document flow, and `<layer>` to position elements and content precisely in the document display space.

Also (and fortunately), Netscape does not distinguish between `<ilayer>` and `<layer>` tags when it comes to order of appearance. You may declare that an `<ilayer>` appear below some `<layer>` by using the `name` and `above` attributes:

```
<layer name=me>I'm on top</layer>
<ilayer above=me>I'm on the bottom</ilayer>
```

Similarly, you may reorder the appearance of both absolute and inline layers where they overlap by assigning `z-index` attribute values to the various elements. Nesting rules apply, as well.

Chapter 15. XML

HTML is a maverick. It follows the rules of formal electronic document-markup design and implementation only loosely. HTML was born out of the need to assemble text, graphics, and other digital content into electronic documents that could be sent over the global Internet. In the early days of the World Wide Web boom, the demand for better browsers and document servers - driven by hordes of new users with insatiable appetites for more and cooler web pages - left little time for worrying about things like standards and practices.

Of course, without guiding standards, HTML would have eventually devolved into Babel. That almost happened during the browser wars in the mid- to late 90s. Chaos is not an acceptable foundation for an industry whose value is already measured in the trillions of dollars. Although the standards people at the W3C managed to rein in the maverick HTML with standard Version 4, it is still too wild for the royal herd of markup languages.

The HTML 4.01 standard is defined using the Standardized Generalized Markup Language (SGML). While more than adequate for formalizing HTML, SGML is far too complex to use as a general tool for extending and enhancing HTML. Instead, the W3C has devised a new standard known as the Extensible Markup Language, or XML. Based upon the simpler features of SGML, XML is kinder, gentler, and more flexible, well-suited to guide the birth and orderly development of new markup languages. With XML, HTML itself is being reborn as XHTML.

In this chapter, we cover the basics of XML, including how to read it, how to create simple XML Document Type Definitions (DTDs), and the ways you might use XML to enhance your use of the Internet. In the next chapter, we explore the depths of XHTML.

You don't have to understand all about XML to write XHTML. We think it's helpful, but if you want to cut to the chase, feel free to skip to the next chapter. However, you may want to take a look at some of the up-and-coming uses of XML covered at the end of this chapter, starting in Section 15.8.

This chapter provides only an overview of XML. Our goal is to whet your appetite and make you conversant in XML. It is only an overview. For full fluency, consult books such as *XML: A Primer* by Simon St. Laurent (IDG Books Worldwide), or *The XML Handbook* by Paul Prescod and Charles Goldfarb (Prentice Hall).

15.1 Languages and Metalanguages

A language is comprised of symbols that we assemble in a meaningful way to express ourselves and pass along information in a way that is intelligible to others. For example, English is a language with rules (grammar) that define how to put its symbols (words) together to form sentences, paragraphs, and, ultimately, books like the one you are holding. If you know the words and understand the grammar, you can read the book, even if you don't necessarily understand its contents.

An important difference between human and computer-based languages is that human languages are self-describing. We use English sentences and paragraphs to define how to create correct English sentences and paragraphs. Our brains are marvelous machines that have no problem understanding that you can use a language to describe itself. However, computer languages are not so rich and computers are not so bright that you could easily define a computer language with itself. Instead, we can define one language - a *metalanguage* - that defines the rules and symbols of another language.

Software developers can use a metalanguage to define the rules for defining a language and then define one or more languages based on those rules.[1] The metalanguage also guides developers creating the automated agents that display or otherwise process the contents of documents that authors have created using that language.

> [1] The use of metalanguages has long been popular in the world of computer programming. The C programming language, for instance, has a set of rules and symbols defined by one of several metalanguages, including *yacc*. Developers use yacc to create compilers, which in turn process language source files into computer-intelligible programs. Hence, its name: Yet Another Compiler Compiler. *yacc*'s only purpose is to help developers create new programming languages.

XML is a metalanguage created by the W3C and is used by developers to define markup languages such as XHTML. Browser developers rely on XML's metalanguage rules to create automated processes that read the language definition of XHTML and implement the processes that ultimately display or otherwise process XHTML documents.

Why bother with a markup metalanguage? Because as the familiar proverb goes, the W3C wants to teach us how to fish so we can feed ourselves for a lifetime. With XML, there is now a standardized way to define markup languages that are customized for different needs rather than having to rely upon HTML extensions. Mathematicians need a way to express mathematical notations; composers need a way to present musical scores; businesses want their web sites to take sales orders from customers; physicians look to exchange medical records; plant managers want to run their factories from web-based documents. All these groups need an acceptable, resilient way to express these different kinds of information, so that the software industry can develop the programs that process and display these diverse documents.

XML provides the answer. Each content sector - the business group, the factory-automation consortium, the trade association - may now define a markup language to suit its particular needs for information exchange and processing over the Web. Computer programmers can create XML-compliant processes - parsers - that read the new language definitions and allow the server to process the documents of those languages.

15.1.1 Creation Versus Display

While there is no limit to the kinds of markup languages you can create with XML, displaying your new documents may be more complicated. When you write HTML, a browser understands what to do with the `<h1>` tag because it is defined in the HTML DTD and browsers have been programmed to display all standard HTML tags.

With XML, you might create a new DTD for describing recipes. It would be a great way to capture and standardize all those kumquat recipes you've been collecting in your kitchen drawers. With special `<ingredient>` and `<portion>` tags, the recipes are easy to define and understand. However, browsers won't know what to do with these new tags unless you attach a style sheet that defines their handling. Without a stylesheet, XML-capable browsers such as Internet Explorer 5 and Netscape 6 will render these tags in a very generic way, cerainly not the flourishing presentation your kumquat recipes deserve.

Even with stylesheets, there are limitations to presenting XML-based information. Let's say you want to create something more challenging, such as a DTD for musical notation or silicon chip design. While describing these data types in a DTD is possible, displaying this information graphically is certainly beyond the capabilites of any stylesheets we've seen yet. It would require a specialized rendering tool to properly display this type of graphically rich information.

Nonetheless, your recipe DTD is a great tool for capturing and sharing recipes. As we'll see later in this chapter, XML isn't simply about creating markup languages for displaying content in browsers. It has great promise for sharing and managing information, so that those precious kumquat dishes will be preserved for many generations to come. Just bear in mind that in addition to writing a DTD to describe your new XML-based markup language, you will in most cases want to supplement the DTD with a stylesheet.[2]

> [2] In fact, it is possible to write XML documents using only a stylesheet. DTDs are highly recommended but optional. See http://www.w3c.org/TR/xml-stylesheet for details.

15.1.2 A Little History

To complete your education into the whys and wherefores of markup languages, it helps to know how all these markup languages came to be.

In the beginning, there was SGML, the Standardized Generalized Markup Language. SGML was intended to be the only markup metalanguage, from which all other markup languages would be created. Everything from hieroglyphics to HTML can be defined using SGML, negating the need for any other metalanguage.

The problem with SGML is that it is so broad and all-encompassing that mere mortals cannot use it. Using SGML effectively requires very expensive and complex tools that are completely beyond the scope of regular people who just want to bang out an HTML document in their spare time. As a result, other markup languages that are greatly reduced in scope and much easier to use have been created. The HTML standards themselves were initially defined using a subset of SGML that eliminated many of the more esoteric features. The DTD in Appendix D uses this subset of SGML to define the HTML 4.01 standard.

Recognizing that SGML was too unwieldy to describe HTML in a useful way and that there was a growing need to define other HTML-like markup languages, the World Wide Web Consortium defined XML. XML is a formal markup metalanguage that uses select features of SGML to define markup languages in a style similar to that of HTML. It eliminates many SGML elements that aren't applicable to languages like HTML and simplifies other elements to make them easier to use and understand.

XML is a middle ground between SGML and HTML, a useful tool for defining a wide variety of markup languages. XML will become increasingly important as the Web extends beyond browsers and moves into the realm of direct data interchange between people, computers, and disparate systems. A small number of people may wind up creating new markup languages with XML, and many more people will want to be able to understand XML DTDs in order to use all these new markup languages.

15.2 Documents and DTDs

To be perfectly correct, we must explain that "XML" has come to mean many subtly different things. An "XML document" is a document containing content that conforms to a markup language defined from the XML standard. An " XML Document Type Definition" (XML DTD) is a set of rules - more formally known as " entity and element declarations" - that define an XML markup language; i.e., how the tags are arranged in a correct ("valid") XML document. To make things even more confusing, entity and element declarations may appear in an XML document itself, as well as within an XML DTD.

An XML document contains character data, which consists of plain content and markup in the form of tags and XML declarations. Thus:

```
<blah>harrumph</blah>
```

is a line in a *well-formed* XML document. Well-formed XML documents follow certain rules, such as the requirement for every tag to have a closing tag. These rules are presented in the context of XHTML in Chapter 16.

To be considered *valid* -- a valid XML document conforms to a DTD - every XML document must have a corresponding set of XML declarations that define how the tags and content should be arranged within it. These declarations may be included directly in the XML document, or they may be stored separately in an XML DTD. If an XML DTD exists that defines the `<blah>` tag, our well-formed XML document is valid, provided you preface it with a `<!DOCTYPE>` tag that explains where to find the appropriate DTD:

```
<?xml version="1.0"?>
<!DOCTYPE blah SYSTEM "blah.dtd">
<blah>harrumph</blah>
```

The example document begins with the optional `<?xml>` directive declaring the version of XML it uses. It then uses the `<!DOCTYPE>` directive to identify the DTD to be used to process the content of the document. In this case, a DTD named `blah.dtd` should be accessible to the browser[3] so the browser can determine whether the `<blah>` tag is valid within the document.

> [3] We use "browser" here because that's what most people will use to process and view XML documents. The XML specification uses the more generic phrase "processing application," since in some cases the XML document will not be processed by a traditional browser but by some other tool that knows how to interpret XML documents.

XML DTDs contain only XML entity and element declarations. XML documents, on the other hand, may contain both XML element declarations and conventional content that uses those elements to create a document. This intermingling of content and declarations is perfectly acceptable to a computer processing an XML document, but it can get confusing for humans trying to learn about XML. For this reason, we will focus our attention in this chapter on the XML entity and element declaration features that you can use to define new tags and document types. In other words, we are addressing only the DTD features of XML; the content features mirror the rules and requirements you already know and use in order to create HTML documents.

15.3 Understanding XML DTDs

To use a markup language defined with XML, you should be able to read and understand the elements and entities found in its XML DTD. But don't be put off: while XML DTDs are verbose, filled with obscure punctuation, and designed primarily for computer consumption, they are actually easy to understand once you get past all the syntactic sugar. Remember, your brain is better at languages than any computer is.

As we said previously, an XML DTD is a collection of XML entity and element declarations and comments. Entities are name/value pairs that make the DTD easier to read and understand, while elements are the actual markup tags defined by the DTD, like HTML's `<p>` or `<h1>` tags. The DTD also describes the content and grammar for each tag in the language. Along with the element declarations, you'll also find attribute declarations that define the attributes authors may use with the tags defined by the element declarations.

There is no required order, although the careful DTD author arranges declarations in such a way that humans can easily find and understand them, computers notwithstanding. The beloved DTD author includes lots of comments, too, that explain the declarations and how they can be used to create a document. Throughout this chapter, we use examples taken from the XHTML 1.0 DTD, which can be found in its entirety at the W3C web site. Although lengthy, you'll find this DTD to be well-written, complete, and, with a little practice, easy to understand.

XML also provides for conditional sections within a DTD, allowing groups of declarations to be optionally included or excluded by the the DTD parser. This is useful when a DTD actually defines several versions of a markup language; the desired version can be derived by including or excluding appropriate sections. The XHTML 1.0 DTD, for example, defines both the "regular" version of HTML and a version that supports frames. By allowing the parser to include only the appropriate sections of the DTD, the rules for the `<html>` tag can change to support either a `<body>` tag or a `<frameset>` tag, as needed.

15.3.1 Comments

The syntax for comments within an XML DTD is exactly like that for HTML comments: comments begin with <!-- and end with -->. Everything between these two elements is ignored by the XML processor. Comments may not be nested.

15.3.2 Entities

An entity is a fancy term for a constant. Entities are crucial to creating modular, easily understood DTDs. Although they may differ in many ways, all entities associate a name with a string of characters. When you use the entity name elsewhere within a DTD, or in an XML document, language parsers replace the name with the corresponding characters. Drawing an example from HTML, the < entity is replaced by the < character wherever it appears in an HTML document.

Entities come in two flavors: *parsed* and *unparsed*. Parsed entities are processed by an XML processor; unparsed ones are ignored. The vast majority of entities are parsed. An unparsed entity is reserved for use within attribute lists of certain tags; it is nothing more than a replacement string used as a value for a tag attribute.

You can further divide the group of parsed entites into *general* entities and *parameter* entities. General entities are used in the XML document, while parameter entities are used in the XML DTD.

You may not realize that you've been using general entities within your HTML documents all along. For example, the entity for the copyright (©) symbol, © is a general entity defined in the HTML DTD. Like all general entities, it is referenced by preceding its name with the ampersand character. All of the other general entities you know and love are listed in Appendix F.

To make life easier, XML predefines the five most common general entities, which can be used in any XML document. While it is still preferred that they be explicitly defined in any DTD that uses them, these five entities are always available to any XML author:

&	&
'	'
>	>
<	<
"	"

You'll find parameter entities littered throughout any well-written DTD, including the HTML DTD. Parameter entities have a percent sign (%) preceding their names. The percent sign tells the XML processor to look up the entity name in the DTD's list of parameter entities, insert the value of the entity into the DTD in place of the entity reference, and process the value of the entity as part of the DTD.

That last bit is important. By processing the contents of the parameter entity as part of the DTD, the XML processor allows you to place any valid XML content in a parameter entity. Many parameter entities contain lengthy XML definitions and may even contain other entity definitions. Parameter entities are the workhorses of the XML DTD; creating DTDs without them would be extremely difficult.[4]

[4] C and C++ programmers may recognize that the entity mechanism in XML is similar to the #define macro mechanism in C and C++. The XML entities provide only simple character string substitution and do not employ C's more elaborate macro parameter mechanism.

15.3.3 Entity Declarations

Let's define an entity with the <!ENTITY> tag in an XML DTD. Inside the tag, first supply the entity name and value, and then indicate whether it is a general or parameter entity:

```
<!ENTITY name value>
<!ENTITY % name value>
```

The first version creates a general entity; the second, because of the percent sign, creates a parameter entity.

For both entity types, the name is simply a sequence of characters beginning with a letter, colon, or underscore, and followed by any combination of letters, numbers, periods, hyphens, underscores, or colons. The only restriction is that names may not begin with the sequence "xml" (either upper or lowercase).

The entity value is either a character string within quotes (unlike HTML markup, you must use quotes even if it is a string of contiguous letters) or a reference to another document containing the value of the entity. For these external entity values, you'll find either the keyword SYSTEM, followed by the URL of the document containing the entity value, or the keyword PUBLIC, followed by the formal name of the document and its URL.

A few examples will make this clear. Here is a simple general entity declaration:

```
<!ENTITY fruit "kumquat or other similar citrus fruit">
```

In this declaration, the entity "&fruit;" within the document will be replaced with the phrase "kumquat or other similar citrus fruit" wherever it appears.

Similarly, here is a parameter entity declaration:

```
<!ENTITY % ContentType "CDATA">
```

Anywhere the reference %ContentType; appears in your DTD, it will be replaced with the word "CDATA". This is the typical way to use parameter entities: to create a more descriptive term for a generic parameter that will be used many times in a DTD.

Here is an external general entity declaration:

```
<!ENTITY boilerplate SYSTEM "http://server.com/boilerplate.txt">
```

It tells the XML processor to retrieve the contents of the file *boilerplate.txt* from *server.com* and use it as the value of the boilerplate entity. Anywhere you use &boilerplate; in your document, the contents of the file will be inserted as part of your document content.

Here is an external parameter entity declaration, lifted from the HTML DTD, that references a public external document:

```
<!ENTITY % HTMLlat1 PUBLIC "-//W3C//ENTITIES Latin 1 for XHTML//EN" "xhtml-lat1.ent">
```

It defines an entity named HTMLlat1 whose contents are to be taken from the public document identified as "*-//W3C//ENTITIES Latin 1 for XHTML//EN*". If the processor does not have a copy of this document available, it can use the URL "*xhtml-lat1.ent* " to find it. This particular public document is actually quite lengthy, containing all of the general entity declarations for the Latin 1 character encodings for HTML.[5]

[5] You can enjoy this document for yourself at http://www.w3.org/TR/xhtml1/DTD/xhtml-symbol.ent.

Accordingly, simply writing in the HTML DTD:

```
%HTMLlat1;
```

causes all of those general entities to be defined as part of the language. A DTD author can use the PUBLIC and SYSTEM external values with general and parameter entity declarations. You should structure your external definitions to make your DTDs and documents easy to read and understand.

You'll recall that we began the section on entities with a mention of unparsed entities whose only purpose is to be used as values to certain attributes. You declare an unparsed entity by appending the keyword NDATA to an external general entity declaration, followed by the name of the unparsed entity. If we wanted to convert our general boilerplate entity to an unparsed general entity for use as an attribute value, we could say:

```
<!ENTITY boilerplate SYSTEM "http://server.com/boilerplate.txt" NDATA text>
```

With this declaration, attributes defined as type ENTITY (as described in Section 15.5.1) could use boilerplate as one of their values.

15.3.4 Elements

Elements are definitions of the tags that can be used in documents based on your XML markup language. In some ways, element declarations are easier than entity declarations, since all you need to do is specify the name of the tag and what sort of content that tag may contain:

```
<!ELEMENT name contents>
```

The name follows the same rules as names for entity definitions. The contents section may be one of four types described here:

- The keyword EMPTY defines a tag with no content, like <hr> and
 in HTML. Empty elements in XML get a bit of special handling, as described in Section 15.4.5.
- The keyword ANY indicates that the tag can have any content, without restriction or further processing by the XML processor.
- The content may be a set of grammar rules that define the order and nesting of tags within the defined element. This content type is used when the tag being defined contains only other tags, without conventional content allowed directly within the tag. In HTML, the <ul> tag is such a tag, as it is only allowed to contain <li> tags.
- Mixed content, denoted by a comma-separated list of element names and the keyword #PCDATA, is enclosed in parentheses. This content type allows tags to have user-defined content, along with other markup elements. The <li> tag, for example, may contain user-defined content as well as other tags.

These last two content types form the meat of most DTD element declarations. This is where the fun begins.

15.4 Element Grammar

The grammar of human language is rich with a variety of sentence structures, verb tenses, and all sorts of irregular constructs and exceptions to the rules. Nonetheless, you mastered most of it by the age of three. Computer language grammars typically are simple, regular, and have few exceptions. In fact, computer grammars use only four rules to define how elements of a language may be arranged: sequence, choice, grouping, and repetition.

15.4.1 Sequence, Choice, Grouping, and Repetition

Sequence rules define the exact order in which elements appear in a language. For instance, if a sequence grammar rule states that element A is followed by B and then by C, your document must provide elements A, B, and C in that exact order. A missing element (A and C, but no B, for example), an extra element (A, B, E, then C), or an element out of place (C, A, then B) violates the rule and does not match the grammar.

In many grammars, XML included, sequences are defined by simply listing the appropriate elements, in order and separated by commas. Accordingly, our example sequence in the DTD would appear simply as `A, B, C`.

Choice grammar rules provide flexibility by letting the DTD author choose one element from among a group of valid elements. For example, a choice rule might state that you may choose elements D, E, or F; any one of these three elements would satisfy the grammar. Like many other grammars, XML denotes choice rules by listing the appropriate choices separated by a vertical bar (`|`). Thus, our simple choice would be written in the DTD as `D | E | F`. If you read the vertical bar as the word *or*, choice rules become easy to understand.

Grouping rules collect two or more rules into a single rule, building richer, more usable languages. For example, a grouping rule might allow a sequence of elements, followed by a choice, followed by a sequence. You can indicate groups within a rule by enclosing them in parentheses in the DTD. For example:

```
Document ::= A, B, C, (D | E | F), G
```

requires that a document begin with elements A, B, and C, followed by a choice of one element out of D, E, or F, followed by element G.

Repetition lets you repeat one or more elements some number of times. With XML, as with many other languages, repetition is denoted by appending a special character suffix to an element or group within a rule. Without the special character, that element or group must appear exactly once in the rule. Special characters include the plus sign (`+`), meaning that the element may appear one or more times in the document; the asterisk (`*`), meaning the element may appear zero or more times; and the question mark (`?`), meaning the element may appear either zero or one time.

For example, the rule:

```
Document ::= A, B?, C*, (D | E | F)+, G*
```

creates an unlimited number of correct documents with the elements A through F. According to the rule, each document must begin with A, optionally followed a B, followed by zero or more occurrences of C, followed by at least one, but perhaps more, of either D, E, or F, followed by zero or more Gs. All of these documents (and many others!) match this rule:

```
ABCDG
ACCCFFGGG
ACDFDFGG
```

You might want to work through these examples to prove to yourself that they are, in fact, correct, with respect to the repetition rule.

15.4.2 Multiple Grammar Rules

By now you can probably imagine that specifying an entire language grammar in a single rule is difficult, although possible. Unfortunately, the result would be an almost unreadable sequence of nearly unintelligible rules. To remedy this situation, the items in a rule may themselves be rules containing other elements and rules. In these cases, the items in a grammar that are themselves rules are known as *nonterminals* , while the items that are elements in the language are known as *terminals*. Eventually, all the nonterminals must reference rules that create sequences of terminals, or the grammar would never produce a valid document.

For example, we can express our sample grammar in two rules:

```
Document ::= A, B?, C*, Choices+, G*
Choices ::= D | E | F
```

In this example, Document and Choices are nonterminals, while A, B, C, D, E, F, and G are terminals.

There is no requirement in XML (or most other grammars) that dictates or limits the number of nonterminals in your grammar. Most grammars use nonterminals wherever it makes sense for clarity and ease of use.

15.4.3 XML Element Grammar

The rules for defining the contents of an element match the grammar rules we just discussed. You may use sequences, choices, groups, and repetition to define the allowable contents of an element. The nonterminals in rules must be names of other elements defined in your DTD.

A few examples show how this works. Consider the declaration of the <html> tag, taken from the HTML DTD:

```
<!ELEMENT html (head, body)>
```

This defines the element named html whose content is a head element followed by a body element. Notice that you do not enclose the element names in angle brackets within the DTD; that notation is used only when the elements are actually used in a document. Within the HTML DTD, you can find the declaration of the <head> tag:

```
<!ELEMENT head (%head.misc;,
     ((title, %head.misc;, (base, %head.misc;)?) |
      (base, %head.misc;, (title, %head.misc;))))>
```

Gulp. What on earth does this mean? First, notice that there is a parameter entity named head.misc used several times in this declaration. Let's go get it:

```
<!ENTITY % head.misc "(script|style|meta|link|object)*">
```

Now things are starting to make sense: head.misc defines a group of elements, from which you may choose one. However the trailing asterisk indicates that you may include zero or more of these elements. The net result is that anywhere %head.misc; appears, you can include zero or more script, style, meta, link, or object elements, in any order. Sound familiar?

Returning to the head declaration, we see that we are allowed to begin with any number of the head miscellaneous elements. We must then make a choice: either a group consisting of a title element, optional miscellaneous items, and an optional base element followed by miscellaneous items; or, a group consisting of a base element, miscellaneous items, a title element, and some more miscellaneous items. Why such a convoluted rule for the <head> tag? Why not just write:

```
<!ELEMENT head (script|style|meta|link|object|base|title)*>
```

which allows any number of the head elements to appear, or none at all? Because the HTML standard requires that every <head> tag contain exactly one <title> tag. It also allows for only one <base> tag, if any. Otherwise, the standard does allow any number of the other head elements, in any order.

Put simply, the head element declaration, while initially confusing, forces the XML processor to ensure that exactly one title element appears in the head element, and that if specified, just one base element appears as well. It then allows for any of the other head elements, in any order.

This one example demonstrates a lot of the power of XML: the ability to define commonly used elements using parameter entities and the use of grammar rules to dictate document syntax. If you can work through the head element declaration and understand it, you are well on your way to reading any XML DTD.

15.4.4 Mixed Element Content

Mixed element content extends the element grammar rules to include the special #PCDATA keyword. "PCDATA" stands for "parsed character data" and signifies that the content of the element will be parsed by the XML processor for general entity references. After the entities are replaced, the character data is passed to the XML application for further processing.

What this boils down to is that parsed character data is the actual content of your XML document. Elements that accept parsed character data may contain plain ol' text, plus whatever other tags you allow, as defined in the DTD.

For instance:

```
<!ELEMENT title (#PCDATA)>
```

means that the title element may contain only text with entities. No other tags are allowed, just as in the HTML standard.

A more complex example is the <p> tag, whose element declaration is:

```
<!ELEMENT p %Inline;>
```

Another parameter entity! The %Inline; entity is defined in the HTML DTD:

```
<!ENTITY % Inline "(#PCDATA | %inline; | %misc;)*">
```

which expands to these entities when you replace the parameters:

```
<!ENTITY % special "br | span | bdo | object | img | map">
<!ENTITY % fontstyle "tt | i | b | big | small">
<!ENTITY % phrase "em | strong | dfn | code | q | sub | sup | samp |
```

```
          kbd | var | cite | abbr | acronym">
<!ENTITY % inline.forms "input | select | textarea | label | button">
<!ENTITY % misc "ins | del | script | noscript">
<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; |
          %inline.forms;">
```

What do we make of all this? The `%Inline;` entity defines the contents of the `p` element as parsed character data, plus any of the elements defined by `%inline;` and any defined by `%misc;`. Notice that case does matter: `%Inline;` is different from `%inline;`.

The `%inline;` entity includes lots of stuff: special elements, font-style elements, phrase elements, and inline form elements. `%misc` includes the `ins`, `del`, `script`, and `noscript` elements. You can read the HTML DTD for the other entity declarations to see which elements are also allowed as the contents of a `p` element.

Why did the HTML DTD authors break up all these elements into separate groups? If they were simply defining elements to be included in the `p` element, they could have built a single long list. However, HTML has rules that govern where inline elements may appear in a document. The authors grouped elements that are treated similarly into separate entities that could be referenced several times in the DTD. This makes the DTD easier to read and understand, as well as easier to maintain when a change is needed.

15.4.5 Empty Elements

Elements whose content is defined to be empty deserve a special mention. XML introduces new notational rules for empty elements, different from the traditional HTML rules that govern them. HTML authors are used to specifying an empty element as a single tag, like `<br>` or `<img>`. XML requires that every element have an opening and a closing tag, so an image tag would be written as `<img></img>` with no embedded content. Other empty elements would be written in a similar manner.

Since this format works well for non-empty tags but is a bit of overkill for empty ones, you can use a special shorthand notation for empty tags. To write an empty tag in XML, just place a slash (`/`) immediately before the closing angle bracket of the tag. Thus, a line break may be written as `<br/>` and an image tag might be specified as `<img src="myimage.gif"/>`. Notice that the attributes of the empty element, if any, appear before the closing slash and bracket.

15.5 Element Attributes

The final piece of the DTD puzzle involves attributes. You know attributes: they are the name/value pairs included with tags in your documents that control the behavior and appearance of those tags. To define attributes and their allowed values within an XML DTD, use the `<!ATTLIST>` directive:

`<!ATTLIST` *`element attributes`*`>`

The `element` is the name of the element to which the attributes apply. The `attributes` are a list of attribute declarations for the element. Each attribute declaration in this list consists of an attribute name, its type, and its default value, if any.

15.5.1 Attribute Values

Attribute values can be of several types, each denoted in an attribute definition with one of the following keywords.

- `CDATA` indicates that the attribute value is a character or string of characters. This is the attribute type you would use to specify URLs or other arbitrary user data. For example, the `src` attribute of the `<img>` tag in HTML has a value of `CDATA`.
- `ID` indicates the attribute value is a unique identifier within the scope of the document. This attribute type is used with an attribute, such as the HTML `id` attribute, whose value defines an ID within the document, as discussed in Appendix B, Section B.1.
- `IDREF` or `IDREFS` indicates that the attribute accepts an ID defined elsewhere in the document via an attribute of type `ID`. The `ID` type is used when defining IDs; the `IDREF` and `IDREFS` are used when referencing a single ID and a list of IDs, respectively.
- `ENTITY` or `ENTITIES` indicates that the attribute accepts the name or list of names of unparsed general entities defined elsewhere in the DTD. The definition and use of unparsed general entities is covered in Section 15.3.2.
- `NMTOKEN` or `NMTOKENS` indicates that the attribute accepts a valid XML name or list of names. These names are given to the processing application as the value of the attribute. How they are used is determined by the application.

In addition to these keyword-based types, you can create an enumerated type by listing the specific values allowed with this attribute. To create an enumerated type, list the allowed values, separated by vertical bars and enclosed in parentheses, as the type of the attribute. For example, here is how the method attribute for the <form> tag is defined in the HTML DTD:

```
method      (get|post)     "get"
```

The method attribute accepts one of two values, either get or post; get is the default value if nothing is specified in the document tag.

15.5.2 Required and Default Attributes

After you define the name and type of an attribute, you must specify how the XML processor should handle default or required values for the attribute. You do this by supplying one of four values after the attribute type.

If you use the #REQUIRED keyword, the associated attribute must always be provided when the element is used in a document. Within the XHTML DTD, the src attribute of the <img> tag is required, since an image tag makes no sense without an image to display.

The #IMPLIED keyword means that the attribute may be used but is not required and that no default value is associated with the attribute. If it is not supplied by the document author, the attribute will have no value when the XML processor handles the element. For the <img> tag, the width and height attributes are implied, since the browser will derive sizing information from the image itself if these attributes are not specified.

If you specify a value, it then becomes the default value for that attribute. If a value for the attribute is not specified by the user, the XML processor will insert the default value (the value specified in the DTD).

If you precede the default value with the keyword #FIXED, the value is not only the default value for the attribute, it is the *only* value that can be used with that attribute, if it is specified.

For example, examine the attribute list for the form element, taken (and abridged) from the HTML DTD:

```
<!ATTLIST form
action           CDATA        #REQUIRED
method           (get|post)   "get"
enctype           CDATA        "application/x-www-form-urlencoded"
onsubmit         CDATA        #IMPLIED
onreset          CDATA        #IMPLIED
accept           CDATA        #IMPLIED
accept-charset   CDATA        #IMPLIED
>
```

This example associates seven attributes with the form element. The action attribute is required and accepts a character string value. The method attribute has one of two values, either get or post. get is the default, so if the document author doesn't include the method attribute in the form tag, method=get will be assumed automatically by the XML parser.

The enctype attribute for the form element accepts a character string value and if not specified, defaults to a value of application/x-www-form-urlencoded. The remaining attributes all accept character strings, are not required, and have no default value if they are not specified.

If you look at the attribute list for the <form> element in the HTML DTD, you'll see that it does not exactly match our example. That's because we've modified our example to show the types of the attributes after any parameter entities have been expanded. In the actual HTML DTD, the attribute types are provided as parameter entities whose names give a hint of kind of values expected by the attribute. For example, the type of the action attribute is %URI;, which elsewhere in the DTD is defined to be CDATA. By using this style, the DTD author lets you know that the string value for this attribute should be a URL, not just any old string. Similarly, the type of the onsubmit and onreset attributes is given as %Script. This is a hint that the character string value should name a script to be executed when the form is submitted or reset.

15.6 Conditional Sections

As we mentioned earlier in this chapter, XML lets you include or ignore whole sections of your DTD so you may can tailor the language for alternative uses. The HTML DTD, for instance, defines transitional, strict, and frame-based versions of the language. DTD authors can select the portions of the DTD they plan to include or ignore by using XML conditional directives:

```
<![INCLUDE [
   ...any XML content...
]]>
```

or:

```
<![IGNORE [
   ...any XML content...
]]>
```

The XML processor will either include or ignore the contents, respectively. Conditional sections may be nested, with the caveat that all sections contained within an ignored section will be ignored, even if they are set to be included.

You will rarely see a DTD with the `INCLUDE` and `IGNORE` keywords spelled out. Instead, you'll see parameter entities that document why the section is being included or ignored. Suppose you are creating a DTD to exchange construction plans among builders. Since you have an international customer base, you build a DTD that can handle both U.S. and metric units. You might define two parameter entities thus:

```
<!ENTITY % English "INCLUDE">
<!ENTITY % Metric "IGNORE">
```

You would then place all the English-specific declarations in a conditional section and isolate the metric declarations similarly:

```
<![%English [
   ...English stuff here...
]]>
<![%Metric [
   ...Metric stuff here...
]]>
```

To use the DTD for English construction jobs, define `%English` as `INCLUDE` and `%Metric` as `IGNORE`, which causes your DTD to use the English declarations. For metric construction, reverse the two settings, ignoring the English section and including the metric section.

15.7 Building an XML DTD

Now that we've emerged from the gory details of XML DTDs, let's see how they work by creating a simple example. You can create a DTD with any text editor and a clear idea of how you want to mark up your XML documents. You'll need an XML parser and processing application to actually interpret and use your DTD as well as a style sheet to permit XML-capable browsers to display your document.

15.7.1 An XML Address DTD

Let's create a simple XML DTD that defines a markup language for specifying documents containing names and addresses. Pretty simple. We start with an `address` element, which contains other elements that tag the address contents. Our `address` element has a single attribute indicating whether it is a work or home address:

```
<!ELEMENT address (name, street+, city, state, zip?)>
<!ATTLIST address type (home|business) #REQUIRED>
```

Voilà! The first declaration creates an element named `address` that contains a name element, one or more street elements, a city and state element, and an optional zip element. The `address` element has a single attribute, `type`, that must be specified and can have a value of either `home` or `business`.

Let's define the `name` elements first:

```
<!ELEMENT name (first, middle?, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT middle (#PCDATA)>
<!ELEMENT last (#PCDATA)>
```

The `name` element also contains other elements: a first name, an optional middle name, and a last name element, each of which are defined in the subsequent DTD lines. These three elements have no nested tags and contain only parsed character data, i.e., the actual name of the person.

The remaining `address` elements are easy, too:

```
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ATTLIST zip length CDATA "5">
```

All these elements contain parsed character data. The zip element has an attribute named `length` that indicates the length of the zip code. If the `length` attribute is not specified, it is set to 5.

15.7.2 Using the Address DTD

Once defined, we can use our address DTD to mark up address documents. For example:

```
<address type="home">
   <name>
      <first>Chuck</first>
      <last>Musciano</last>
   </name>
   <street>123 Kumquat Way</street>
   <city>Cary</city>
   <state>NC</state>
   <zip length="10">27513-1234</zip>
</address>
```

With an appropriate XML parser and an application to use this data, we can parse and store addresses, create addresses to share with other people and applications, and create display tools that would publish addresses in a wide range of styles and media. Although our DTD is simple, it has defined a standard way to capture address data that is easy to use and understand.

15.8 Using XML

Our address example is trivial. It hardly scratches the surface of the wide range of applications that XML is suited for. To whet your appetite, here are some common uses for XML that you will certainly be seeing now and in the future.

15.8.1 Creating Your Own Markup Language

We touched on this earlier when we mentioned that the latest versions of HTML are being reformulated as compliant XML DTDs. We cover the impact XML has on HTML in the next chapter.

But even more significantly, XML enables communities of users to create languages that best capture their unique data and ideas. Mathematicians, chemists, musicians, and professionals from hundreds of other disciplines can create special tags that represent unique concepts in a standardized way. Even if no browser exists that can accurately render these new tags in a displayable form, the ability to capture and standardize information is tremendously important for future extraction and interpretation of these ideas.

For more mainstream XML applications with established audiences, it is easy to envision custom browsers being created to appropriately display the new information. Smaller applications or markets may have more of a challenge creating markup languages that enjoy such wide acceptance. Creating the custom display tool for a markup language is difficult; delivering that tool for multiple platforms is expensive. As we've noted, some of these display concerns can be mitigated by appropriate use of style sheets. Luckily, XML's capabilities extend beyond document display.

15.8.2 Document Exchange

Because XML grew out of the tremendous success of HTML, many people think of XML as yet another document display tool. In fact, the real power of XML lies not in the document display arena, but in the world of data capture and exchange.

Despite the billions of computers deployed worldwide, sharing data is as tedious and error-prone as ever. Competing applications do not operate from common document storage formats, so sending a single document to a number of recipients is fraught with peril. Even when vendors attempt to create an interchange format, it still tends to be proprietary and is often viewed as a competitive advantage for participating vendors. There is little incentive for vendors to release application code for the purpose of creating easy document-exchange tools.

XML avoids these problems. It is platform-neutral, generic and can perform almost any data capture task. It is equally available to all vendors and can be easily integrated into most applications. The stabilization of the XML standard and the increasing availability of XML authoring and parsing tools is making it easier and easier to create XML markup languages for document capture and exchange.

Most importantly, document exchange rarely requires document presentation, thus eliminating "display difficulties" from the equation. Often, an existing application will use XML to include data from another source, and then use its own internal display capabilities to present the data to the end user. The cost of adding XML-based data exchange to existing applications is relatively small.

15.8.3 Connecting Systems

A level below applications, there is also a need for systems to exchange data. As business-to-business communication increases, this need grows even faster. In the past, this meant that someone had to design a protocol to encode and exchange the data. With XML, exchanging data is as easy as defining a DTD and integrating the parser into your existing applications.

The data sets exchanged could in fact be quite small. Imagine shopping for a new PC on the Web. If you could capture your system requirements as a small document using an XML DTD, you could send that spec to a hundred different vendors to quote you a system. If you extend that model to include almost anything you can shop for - from cars to hot tubs - XML provides an elegant base layer of communication among cooperating vendors on the Internet.

Almost any data that is now captured and stored can be more easily shared using XML. For many systems, the XML DTDs will define a data transfer protocol and nothing more. The data may never actually be stored using the XML-defined markup; it may exist in an XML-compatible form only long enough to pass on the wire between two systems.

In conjunction with XML-based data exchange, the Extensible Stylesheet Language, or XSL, will be increasingly used to describe the appearance and definition of the data represented by these XML DTDs. Much like Cascading Style Sheets and their ability to transform HTML documents, XSL will support the creation of style sheets for any XML DTD. CSS can be used with XML documents as well, but it is not as programmatically rich as XSL. While CSS stops with style sheets, XSL is a style language. XSL certainly addresses the need for data display, and it also provides rich tools that allow data represented with one DTD to be transformed into another DTD in a controlled and deterministic fashion. A complete discussion of XSL is beyond the scope of this book; consult an appropriate O'Reilly reference for complete details.

The potential for XML goes well beyond that of traditional markup and presentation tools. What we now see and use in the XML world is only scratching the surface of the potential for this technology.

15.8.4 Standardizing HTML

Last but certainly not least, XML is being used to define a standard version of HTML known as XHTML. XHTML retains almost all of the features of HTML 4.01, but also introduces a number of minor (and a few not-so-minor) differences. The next chapter compares and contrasts XHTML and HTML, mapping out the differences so that you can begin creating documents that comply with both the HTML and XHTML standards.

Chapter 16. XHTML

Despite its name, you don't use Extensible Markup Language (XML) to directly create and mark up web documents. Instead you use XML technology to define a new markup language, which you then use to mark up web documents. This should come as no surprise to anyone who has read the previous chapter in this book. Nor, then, should it surprise you that one of the first languages defined using XML is an XML-ized version of HTML, the most popular markup language ever. HTML is now being disciplined and cleaned up by XML to bring it back into line with the larger family of markup languages. This new standard is XHTML 1.0.[1]

[1] Throughout this chapter, we use "XHTML" to mean the XHTML 1.0 standard. There is a nascent XHTML 1.1 standard which diverges from XHTML 1.0 and HTML 4.01. See http://www.w3.org/TR/xhtml11/ for the details and differences.

Because of HTML's legacy features and oddities, using XML to describe HTML was not an easy job for the W3C. In fact, certain HTML rules, as we'll discuss later, cannot be represented using XML. Nonetheless, if the W3C has its way, XHTML will ultimately replace the HTML we currently know and love. We agree that it should.

So much of XHTML is identical to HTML's current standard, Version 4.01, that almost everything presented elsewhere in this book may be applied to both HTML and XHTML. The differences, both good and bad, are detailed in this chapter. To become fluent in XHTML, you'll first need to absorb the rest of this book, and then adjust your thinking to embrace what we present in this chapter.

16.1 Why XHTML?

HTML, as everyone should know by now, began as a simple markup language similar in appearance and usage to other SGML-based markup languages. In its early years, little effort was put into making HTML perfectly SGML-compliant. As a result, odd features and a lax attitude towards enforcing the rules became a standard part of both HTML and the browsers that processed HTML documents.

As the Web grew from an experiment into an industry, the desire for a standard version of HTML led to the creation of several official versions, culminating most recently with Version 4.01. As HTML has stabilized into this latest version, browsers have become more alike in their support of various HTML features. In general, the world of HTML has settled into a familiar set of constructs and usage rules.

Unfortunately, HTML offers only a limited set of document-creation primitives, is incapable of handling nontraditional content such as chemical formulae, musical notation, or mathematical expressions, and fails to adequately support alternative display media such as handheld computers or intelligent cellular phones. We need new ways to deliver information that can be parsed, processed, displayed, sliced, and diced by the many different communication technologies that have emerged since the Web sparked the digital communication revolution a decade ago.

Rather than trying to rein in another herd of maverick, nonstandard markup languages, the W3C introduced XML as a standard way to create new markup languages. XML is the framework upon which organizations can develop their own markup languages to suit the needs of their users. XML is an updated version of SGML, streamlined and enhanced for today's dynamic systems. And while the W3C originally intended it as a tool to create document markup languages, XML is also becoming quite useful as a standard way to define tiny little languages that are used as data exchange protocols between different applications.

Of course, we don't want to abandon the plethora of documents already marked up with HTML or the infrastructure of knowledge, tools, and technologies that currently support HTML and the Web. Yet, we do not want to miss the opportunities of XML, either. XHTML is the bridge. It uses the features of XML to define a markup language that is nearly identical to standard HTML 4.01 and gets us all started down the XML road.

16.1.1 XHTML Document Type Definitions

HTML 4.01 comes in three variants, each defined by a separate SGML DTD. Similarly XHTML also comes in three variants, with XML DTDs corresponding to the three SGML DTDs that define HTML 4.01. To create an XHTML document, you must choose one of these DTDs and then create a document that uses its particular elements and rules.

The first XHTML DTD corresponds to the "strict" HTML DTD. The strict definition excludes all deprecated elements (tags and attributes) in HTML 4.01 and forces authors to use only those features that are fully supported in HTML. Many of the HTML elements and attributes dealing with presentation and appearance, such as the `<font>` tag and the `align` attribute, are missing from the strict XHTML DTD, replaced by the equivalent properties in the Cascading Style Sheet model.

Most HTML authors find the strict XHTML DTD too restrictive, since many of the deprecated elements and attributes are still in widespread use throughout the Web. More importantly, the popular browsers - while fully supporting the deprecated elements - have yet to fully implement the new standard ones. The only real advantage in using the strict XHTML DTD is that compliant documents are guaranteed to be fully supported in future versions of XHTML.[2]

[2] If the W3C has its way, HTML won't change beyond Version 4.01. No more HTML; all new developments will be in XHTML and many other XML-based languages.

Most authors will probably choose to use the "transitional" XHTML DTD. It's closest to the current HTML standard and includes all those wonderful, but deprecated, features that make life as an HTML author easier. With the transitional XHTML DTD, you can ease into the XML family while staying current with the browser industry.

The third DTD is for frames. It is identical to the transitional DTD in all other respects; the only difference is the replacement of the document body with appropriate frame elements. You might think that, for completeness' sake, there would be strict and transitional frame DTDs, but the W3C decided that if you use frames, you might as well use all the deprecated elements as well.

16.2 Creating XHTML Documents

For the most part, creating an XHTML document is no different than creating an HTML document. Using your favorite text editor, simply add the markup elements in the right order to your document's contents, and display it using your favorite browser. To be strictly correct ("valid," as they say at the W3C), your XHTML document will need a boilerplate declaration up front that specifies the DTD you used to create the document and defines a namespace for the document.

16.2.1 Declaring Document Types

For an XHTML browser to correctly parse and display your XHTML document, you should tell it which version of XML is being used to create the document. You must also state which XHTML DTD defines the elements in your document.

The XML version declaration uses a special XML processing directive. In general, these XML directives begin with `<?` and end with `?>`, but otherwise they look like typical tags in your document.[3]

[3] `<!` was already taken.

To declare that you are using XML Version 1.0, place this directive in the first line in your document:

```
<?xml version="1.0" encoding="UTF-8"?>
```

This tells the browser that you are using XML 1.0 along with the 8-bit Unicode character set, the one most commonly used today. The `encoding` attribute's value should reflect your local character set. Refer to the appropriate ISO standards for other encoding names.

Once you've gotten the important issue of the XML version squared away, you should then declare the markup language's DTD:

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

With this statement, you declare that your document's root element is `html`, as defined in the DTD whose public identifier is defined as `"-//W3C//DTD XHTML 1.0 Strict//EN"`. The browser may know how to find the DTD matching this public identifier. If it does not, it can use the URL following the public identifier as an alternative location for the DTD.

As you may have noticed, the `<!DOCTYPE>` directive has told the browser to use the strict XHTML DTD. Here's the one you'll probably use for your transitional XHTML documents:

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

And, as you might expect, the `<!DOCTYPE>` directive for the frame-based XHTML DTD is:

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

16.2.2 Understanding Namespaces

As described in the last chapter, an XML DTD defines any number of element and attribute names as part of the markup language. These elements and attribute names are stored in a *namespace* that is unique to the DTD. As you reference elements and attributes in your document, the browser looks them up in the namespace to find out how they should be used.

For instance, the <a> tag's name ("a") and attributes like "href " and "style" are defined in the XHTML DTD and their names are placed in the DTD's namespace. Any "processing agent" - usually a browser, but your eyes and brain can serve the same function - can look up the name in the appropriate DTD to figure out what the markup means and what it should do.

With XML, your document actually may use more than one DTD, and therefore need more than one namespace. For example, you might create a transitional XHTML document, but also include special markup for some math expressions according to an XML math language. What happens when both the XHTML DTD and the math DTD use the same name to define different elements, such as <a> for XHTML hypertext and <a> for an absolute value in math? How does the browser choose which namespace to use?

The answer is the xmlns[4] attribute. Use it to define one or more alternative namespaces within your document. It can be placed within the start tag of any element within your document, and its URL-like[5] value defines the namespace that the browser should use for all content within that element.

> [4] XML namespace - xmlns - get it? This is why XML doesn't let you begin any element or attribute with the three-letter prefix "xml": it's reserved for special XML attributes and elements.
>
> [5] It looks like a URL, and you might think that it references a document that contains the namespace, but alas, it doesn't. It is simply a unique name that identifies the namespace. Display agents use that placeholder to refer to their own resources for how to treat the named element or attribute.

With XHTML, according to the new XML conventions, you should at the very least include an xmlns attribute within your document's <html> tag that identifies the primary namespace used throughout the document:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

If and when you need to include math markup, you use the xmlns attribute again to define the math namespace. So, for instance, you could use the xmlns attribute within some math-specific tag of your otherwise common XHTML document (assuming the MATH element exists, of course):

```
<div xmlns="http://www.w3.org/1998/Math/MathML>x2/x</div">
```

In this case, the XML-compliant browser would use the http://www.w3.org/1998/Math/MathML namespace to divine that this is the MATH, not the XHTML, version of the tag, and should therefore be displayed as a division equation.

It would quickly become tedious if you had to embed the xmlns attribute into each and every tag any time you wanted to show a division equation in your document. A better way - particularly if you plan to apply it to many different elements in your document - is to identify and label the namespace at the beginning of your document, and then refer to it by that label as a prefix to the affected element in your document. For example:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:math="http://www.w3.org/1998/Math/MathML">
```

The math namespace can now be abbreviated to "math" later in your document. So the streamlined:

```
<math:div>x2/x</div>
```

now has the same effect as the lengthy earlier example of the math tag containing its own xmlns attribute.

For the most part, the vast majority of XHTML authors will never need to define multiple namespaces and so will never have to use fully qualified names containing the namespace prefix. Even so, you should understand that multiple namespaces exist and that you will need to manage them if you choose to embed content based upon one DTD within content defined by another DTD.

16.2.3 A Minimal XHTML Document

As a courtesy to all fledgling XHTML authors, we now present the minimal and correct XHTML document, including all the appropriate XML, XHTML, and namespace declarations. With this most difficult part out of the way, you need only supply content to create a complete XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
          PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Every document must have a title</title>
  </head>
  <body>
    ...your content goes here...
  </body>
</html>
```

Working through the minimal document one element at a time, we begin by declaring we are basing the document on the XML 1.0 standard and using 8-bit Unicode characters to express its contents and markup. We then announce, in the familiar HTML-like `<!DOCTYPE>` statement, that we are following the markup rules defined in the transitional XHTML 1.0 DTD, which allow us free rein to use nearly any and all HTML 4.01 elements in our document.

Our document content actually begins with the `<html>` tag, which has its `xmlns` attribute declare that the XHTML namespace will be the default namespace for the entire document. Also note the `lang` attribute, in both the XML and XHTML namespaces, which declares that the document language is English.

Finally, we include the familiar document `<head>` and `<body>` tags, along with the required `<title>` tag.

16.3 HTML Versus XHTML

The majority of HTML is completely compatible with XHTML, and this book is devoted to that majority. In this chapter, however, we talk about the minority: where the HTML 4.01 standard and the XHTML DTD differ. If you truly desire to create documents that are both HTML- and XHTML-compliant, you must heed the various warnings and caveats we outline in the following sections.

The biggest difference - that's Difference with a capital D and that spells difficult - is that writing XHTML documents requires much more discipline and attention to detail than even the most fastidious HTML author ever dreamed necessary. In W3C parlance, that means your documents must be impeccably "well-formed." Throughout the history of HTML - and in this book - authors have been encouraged to create well-formed documents, but you would have to break rank with the HTML standards for your documents to be considered well-formed by XML standards.

Nonetheless, your efforts to master XHTML will be rewarded with documents that are well-formed and a sense of satisfaction from playing by the new rules. You will truly benefit in the future, too: through XML, your documents will be able to appear in places you never dreamed would exist (mostly good places, we hope).

16.3.1 Correctly Nested Elements

One requirement of a well-formed XHTML document is that its elements are nested correctly. This isn't any different from HTML standards: simply close the markup elements in the order in which you opened them. If one element is within another, the end tag of the inner element must appear before the end tag of the outer element.

Hence, in the following well-formed XHTML segment, we end the italics tag before we end the bold one, because we'd started italicizing after we had started bolding the content:

```
<b>Close the italics tag <i>first</i></b>.
```

On the other hand, the following:

```
<b>Well-formed, this is <i>not!</b></i>
```

is not well-formed.

XHTML strictly enforces other nesting restrictions that have always been part of HTML but not always enforced. These restrictions are not formally part of the XHTML DTD; they are instead defined as part of the XHTML standard that is based upon the DTD.[6]

[6] This is hair-splitting within the XHTML standard. The XML standard has no mechanism to define which tags may not be placed within another tag. SGML, upon which XML is based, does have such a feature, but it was removed from XML to make the language easier to use and implement. As a result, these restrictions are simply listed in an appendix of the XHTML standard instead of explicitly defined in the XHTML DTD.

Nesting restrictions include:

- The <a> tag cannot contain another <a> tag.
- The <pre> tag cannot contain <img>, <object>, <big>, <small>, <sub>, or <sup>.
- The <button> tag cannot contain <input>, <select>, <textarea>, <label>, <button>, <form>, <fieldset>, <iframe>, or <isindex>.
- The <label> tag cannot contain other <label> tags.
- The <form> tag cannot contain other <form> tags.

These restrictions apply to nesting at any level. For example, an <a> tag cannot contain any other <a> tags, or any tag that in turn contains an <a> tag.

16.3.2 End Tags

As we've documented throughout this book, any HTML tag that contains other tags or content has a corresponding end tag. However, one of the hallmarks of HTML (codified in the 4.01 standard) is that you may leave out the end tags if their presence can be inferred by the processing agent. This is why most of us HTML authors commonly leave out the end tag between adjacent paragraphs. Also, lists and tables can be complicated to wade through and not having to visually stumble over all the , </td>, </th>, and </tr> end tags certainly makes HTML a little clearer and easier to read.

This is not so for XHTML. Every tag that contains other tags or content must have a corresponding end tag present, correctly nested within the XHTML document. A missing end tag is an error and renders the document non-compliant.

16.3.3 Handling Empty Elements

In XML, and thus XHTML, every tag must have a corresponding end tag, even those aren't allowed to contain other tags or content. Accordingly, XHTML expects the line break to appear as
</br> in your document. Ugh.

Fortunately, there is an acceptable alternative: include a slash before the closing brace of the tag to indicate its ending, as in
. If the tag has attributes, the slash comes after all the attributes, so that an image could be defined as:

```
<img src="kumquat.gif" />
```

While this notation may seem foreign and annoying to an HTML author, it actually serves a useful purpose. Any XHTML element that has no content can be written this way. Thus, an empty paragraph can be written as and an empty table cell can be written as <td />. This is a handy way to mark empty table cells.

Clever as it may seem, writing empty tags in this abbreviated way may confuse HTML browsers.

So to avoid compatibility problems, you can fool the HTML browsers by placing a space before the forward slash in an empty element using the XHTML version of its end tag. For example, use
 with its space between the br and /, instead of the XHTML equivalents
 or
</br>. Table 16-1 contains all the empty HTML tags expressed in their acceptable XHTML (transitional DTD) forms.

Table 16-1, HTML Empty Tags in XHTML Format

<area />	<img />
<base />	<input />
<basefont />	<isindex />
 	<link />
<col />	<meta />
<frame />	<param />
<hr />	

16.3.4 Case Sensitivity

If you thought getting all those end tags in the right place and cleaning up the occasional nesting error would make writing XHTML documents difficult, hold on to your hat. XHTML is case-sensitive for *all* tag and attribute names. In an XHTML document, `<a>` and `<A>` are different tags; `src` and `SRC` are different attributes. So are `sRc` and `SrC` ! How forgiving HTML seems now.

The XHTML DTD defines all former HTML tags and attributes using lowercase letters. Uppercase tag or attribute names are not valid XHTML tags or attributes.

This can be a difficult situation for any author wishing to convert existing HTML documents into XHTML-compliant ones. Lots of web pages use uppercase tag and attribute names, to make them stand out from the surrounding lowercase content.

To become compliant, all those names must be converted to lowercase, even the ones you'd used in your CSS style sheet definitions. Fortunately, this kind of change is easily accomplished with various editing tools. And XHTML authoring systems should perform the conversion for you.

16.3.5 Quoted Attribute Values

And as if all those case-sensitive attribute names weren't aggravating enough, XHTML requires that every attribute value - even the numeric ones - be enclosed in double quotes. In HTML, you could quote anything your heart desired, but quotes were only required if the attribute value included whitespace or other special characters. To be XHTML-compliant, every attribute must be enclosed in quotes.

For example:

```
<table rows=3>
```

is wrong in XHTML. It is correctly written:

```
<table rows="3">
```

16.3.6 Explicit Attribute Values

Within HTML, there are a small number of attributes that have no value. Instead, their mere presence within a tag causes that tag to behave differently. In general, these attributes represent a sort of on/off switch for the tag, like the `compact` attribute for the various list tags or the `ismap` attribute for the `<img>` tag.

In XHTML, every attribute must have a value. Those without values must now use their own names. Thus, `compact` in XHTML is correctly specified as `compact="compact"`; `checked` is now `checked="checked"` Each must contain the newly required attribute value enclosed in quotes. Table 16-2 contains a list of newly valued attributes.

Table 16-2, New XHTML Values for Value-Less HTML Attributes

`compact="compact"`	`disabled="disabled"`
`nowrap="nowrap"`	`readonly="readonly"`
`ismap="ismap"`	`multiple="multiple"`
`declare="declare"`	`selected="selected"`
`noshade="noshade"`	`noresize="noresize"`
`checked="checked"`	`defer="defer"`

Be aware that this new attribute value requirement may cause some old HTML browsers to ignore the attribute altogether. HTML 4.0-compliant browsers don't have that problem, so the majority of users won't notice any difference. There is no good solution to this problem other than distributing HTML 4.0-compliant browsers to the needy.

16.3.7 Handling Special Characters

XHTML is more sensitive than HTML is to the use of the `<` and `&` characters in JavaScript and CSS declarations within your documents. In HTML, you can avoid potential conflicts by enclosing your scripts and stylesheets in comments (`<!--` and `-->`). XML browsers, however, may simply remove all the contents of comments from your document, thereby deleting your hidden scripts and stylesheets.

To properly shield your special characters from XML browsers, enclose your styles or scripts in a CDATA section. This tells the XML browser that any characters contained within are plain old characters, without special meanings. For example:

```
<script language="JavaScript">
<![CDATA[
   JavaScript here...
 ]]>
</script>
```

This doesn't solve the problem, though. HTML browsers ignore the contents of the CDATA XML tag, but honor the contents of comment-enclosed scripts and stylesheets, whereas XML browsers do just the opposite. We recommend that you put your scripts and styles in external files and reference them in your document with appropriate external links.

Special characters in attribute values are problematic in XHTML, as well. In particular, an ampersand within an attribute value should always be written using `&` and not simply an `&` character. Similarly, play it safe and encode less-than and greater-than signs using their `<` and `>` entities. For example, while:

```
<img src=seasonings.gif alt="Salt & pepper">
```

is perfectly valid HTML, it must be written as:

```
<img src="seasonings.gif" alt="Salt &amp; pepper" />
```

to be compliant XHTML.

16.3.8 The id and name Attributes

Early versions of HTML used the `name` attribute with the `<a>` tag to create a fragment identifier in the document. This fragment could then be used in a URL to refer to a particular spot within a document. The `name` attribute was later added to other tags like `<frame>` and `<img>`, allowing those elements to also be referenced by name from other spots in the document.

With HTML 4.0, the W3C added the `id` attribute to almost every tag. Like `name`, `id` lets you associate an identifier with nearly any element in a document for later reference and use, perhaps by a hyperlink or a script.

XHTML has a strong preference for the `id` attribute as the anchor of choice within a document. The `name` attribute is defined, but formally deprecated for those elements that have historically used it. With widespread support of HTML 4.0 now in place, you should begin to avoid the `name` attribute where possible, and instead use the `id` attribute to bind names to elements in your documents. If you must use the `name` attribute on certain tags, include an identical `id` attribute to ensure that the tag will behave similarly when processed by an XHTML browser.

16.4 Should You Use XHTML?

For a document author used to HTML, XHTML is clearly a more painful and less forgiving document markup language. Whereas at one time we prided ourselves on being able to crank out HTML with pencil and paper, it's much more tedious to write XHTML without special document-preparation applications. Why should any author want to take on that extra baggage?

16.4.1 The Dusty Deck Problem

Over just a few years, the Web has been filled with billions of pages. It is a safe bet that many of these pages are not compliant with any defined version of HTML. It is an even safer bet that the vast majority of these pages are not XHTML-compliant.

The harsh reality is that these billions of pages will never be converted to XHTML. Who has the time to go back, root out these old pages, and tweak them to make them XHTML-compliant - especially when the end result, as perceived by the user, will not change? Like the dusty decks of COBOL programs that lay unchanged for decades before the Y2K problem forced programmers to bring them up to snuff, these dusty decks of web pages will also lie untouched until a similarly dramatic event forces us to update them.

However, the dusty deck problem is no excuse for not writing compliant documents going forward. Leave those old documents alone, but don't create a new conversion problem every time you create a new document. A little effort now will help your documents work across a wider range of browsers in the future.

16.4.2 Automatic Conversion

If your sense of responsibility leads you to undertake the conversion of your existing HTML document into XHTML, you'll find a utility named Tidy to be exceptionally useful. Written by Dave Raggett, one of the movers and shakers at the W3C, it automates a significant amount of the work required to convert HTML documents into XHTML.

While Tidy's capabilities are too varied and wonderful to be fully listed here, we can at least assure you that the case conversion, quoted attributes, and proper element nesting are all detected and corrected by Tidy. For the complete list of features and the latest version of Tidy for any number of computing platforms, visit http://www.w3.org/People/Raggett/tidy/.

16.4.3 Lenient Browsers and Lazy Authors

There is a good rule of thumb regarding data sharing, especially on the Internet: be lenient in what you accept and strict in what you produce. This is a not a commentary on social policy, but rather a pragmatic admonition to tolerate ambiguity and errors in data you receive while making sure that anything you send is scrupulously correct.

Web browsers are good examples of lenient acceptors. Most current web pages have some sort of error in them, albeit often just an error of omission. Nonetheless, browsers accept the error and present a reasonable document to the user. This leniency lets authors get away with all sorts of things, often without even knowing they've made a mistake.

Most authors stop developing a page when it looks good and works the way they want it to. Very few take the time to run their pages through the various HTML- compliance tools to catch potential errors. Many of those who do try to test for compliance are so overwhelmed by the number of minor errors they have committed that they simply give up and continue to create bad pages that can be handled by good browsers.

Since the number of bad pages continues to grow, browsers cannot afford to start being strict. Any browser that tried to enforce even the most basic rules of the HTML standard would be abandoned by users who want to see web pages, not error messages. A vicious cycle ensues: bad pages force the use of lenient browsers, which encourage the creation of more bad pages. Break the cycle by vowing to create only XHTML-compliant content whenever you can.

16.4.4 Time, Money, and Standards

XHTML was developed as an XML representation of the HTML standard. It is intended, going forward, to become the single standard everyone should use to create content for the Web.

In a perfect world, standards are universally adopted and used. Full compliance is required of any document before it is placed on the Web. Conversion of legacy documents is done immediately.

In the real world, a shortage of time and money prevent the universal use of standards. Under pressure to quickly deliver something that works, developers turn out pages that work only well enough. Since browsers allow second-rate content to exist on the Web, the need to comply with a standard becomes a secondary issue, one that is too quickly ignored in the dizzying pace of web development.

16.4.5 Man Versus Machine

All is not lost, however. While XHTML is painful and tedious for humans to create, it is quite easy for machines to create. As the number of web authoring tools continues to increase, the pages created by these machines should be completely XHTML compliant. While it doesn't make much economic sense for a web author to spend a lot of time getting all those end tags in the right spot, it does make sense for the programmer developing an authoring tool to ensure that the tool generates all those correct end tags. The effort expended by the web author is leveraged exactly once for each page; the effort of the tool creator is leveraged over and over, each time the tool produces a new page.

It seems that the real future of XHTML lies in the realm of machine-generated content. XHTML is far too picky to be successfully used by the millions of casual web authors who create small sites. However, if those same authors use a tool to create their pages, they could be generating XHTML-compliant pages and never even know it.

If you are among that small community of developers who create tools that generate HTML output, you are doing a great disservice to your many potential customers if your tool does not generate excruciatingly correct XHTML-compliant output. There is no technical excuse for any tool not to generate XHTML-compliant output. If there are compatibility issues surrounding how the output might be used (with an non-XHTML browser, perhaps), then the tool should provide a switch that lets the author select XHTML-compliant output as an option.

16.4.6 What To Do?

We recommend that all HTML authors take the time to absorb the differences between HTML and XHTML as we've outlined in this chapter. Given the resources and opportunity, you should try to create XHTML-compliant pages wherever possible for the sites you are creating. Certainly you should choose authoring tools that support XHTML and give you the option of generating XHTML-compliant pages.

One day, XHTML may - and should - replace HTML as the official standard language of the Web. Even so, the number of noncompliant pages on the Web will be overwhelming, forcing browsers to honor old HTML constructs and features for at least the next five to ten years. For better or worse, HTML is here to stay as the *de facto* standard for web authors for years to come.

Chapter 17. Tips, Tricks, and Hacks

We've sprinkled a number of tips, tricks, and hacks throughout this book, along with style guidelines, examples, and instructions. So why have a special chapter on tips, tricks, and hacks? Because it's where many readers will leaf to when they pick up this book for the first time. HTML and XHTML are the languages, albeit constrained, that make the World Wide Web the exciting place that it is. And interested readers want to know, "How do I do the cool stuff ?"

17.1 Top of the Tips

The most important tip for even veteran authors is to surf the Web for yourself. We can show and explain a few neat tricks to get you started, but there are thousands of authors out there combining and recombining HTML and XHTML tags and juggling content to create compelling and useful documents.

Get copies of Netscape Navigator (the browser portion of Communicator), Internet Explorer, and whatever other browser you feel comfortable operating, and cruise. Collect web site URLs from friends, business associates, and the traditional media. Even local TV and radio stations have taken to announcing some of their sponsors' web site URLs. And consult the many different web directories like Yahoo, Excite, Lycos, and AltaVista for new and up-to-date addresses for the web sites that suit your lifestyle or business niche.

Examine (don't steal) their pages for eye-catching and effective pages and use them to guide your own creations. Capture and examine the source documents for the juicy bits. Get a feel of the more effective web collections. How are their documents organized? How large is each document? And so on.

We all learn from experience, so go get it.

17.1.1 Design for Your Audience

We continuously argue throughout the book that content matters most, not look. That doesn't mean presentation doesn't matter.

Effective documents match your target audience's expectations, giving them a familiar environment in which to explore and gather information. Serious academicians, for instance, expect a journal-like appearance for a treatise on the physiology of the kumquat: long on meaningful words, figures, and diagrams and short on frivolous trappings like cute bullets and font abuse. Don't insult the reader's eye, except when exercising artistic license to jar or in order to attack your reader's sensibilities.

By anticipating your audience and designing your documents to appeal to their tastes, you also subtly deflect unwanted surfers from your pages. Undesirables, such as penniless college students surfing your commercial site[1], may hog your server's resources and prevent the buying audience you desire from ready access to your pages.

> [1] Not that there's anything wrong with that. We both started out as penniless college students and years later, wound up writing for O'Reilly & Associates.

You can use subtle colors and muted text transitions between sections for a classical art museum's collection to mimic the hushed environment of a real classical art museum. The typical rock-'n'-roll crazed web surfer maniac probably won't take more than a glance at your site, but the millionaire arts patron might.

Also, use effective layout to gently guide your readers' eyes to areas of interest in your documents. Do that by adhering to the basic rules of document layout and design, such as placing figures and diagrams nearby - if not inline - with their content reference. Nothing's worse than having to scroll up and down the browser window in a desperate search for a picture that can explain everything.

We won't lie and suggest that we're design experts. We aren't, but they're not hard to find. So, another tip for the serious web page author: seek professional help. The best situation is to have design experience yourself. Next best is to have a pro looking over your shoulder, or at least somewhere within earshot.

Make a trip to your local library and do some reading on your own, too. Even better yet, browse the various online guides. Check out *Web Design in a Nutshell, Second Edition* (O'Reilly & Associates). Your readers will be glad you did. Section 1.7

17.1.2 Boilerplate Documents

The next best tip we can give you is to reuse your documents. Don't start from scratch each time. Rather, develop a consistent framework, even to the point of a content outline into which you add the detail and character for each page. You might even endeavor to create style sheets, so that the look and feel of your documents remain consistent.

Here's our contribution to help start your boilerplate document collection. The following sources contain what the HTML and the XHTML standards currently tell us is the minimum content that should appear in every respective document (regardless of what the browsers might let you get away with) and then some added for document clarity. Use them as skeletons for your own documents (they look mighty alike):

```
<html>
  <head>
    <title>Required; replace this title with your own</title>
  </head>
  <body>
    <h3>Reiterate the title here</h3>
      ...Insert your document's contents here...
    <address>Include your name and contact information
             usually at the end of the document
    </address>
  </body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
          PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          " http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Every XHTML document needs a title</title>
  </head>
  <body>
    <h3>Reiterate the title here</h3>
        ...Insert your document's contents here...
    <address>Include your name and contact information
                usually at the end of the document
    </address>
  </body>
</html>
```

17.2 Trivial or Abusive?

There is perhaps no more abused or abusive feature than the <blink> tag extension for text content currently supported by, thank heaven, only the Netscape browser.[2]

> [2] Internet Explorer is just as guilty; be thankful you don't find many documents using the browser's equally tacky <marquee> feature.

It works by alternating the color of the text enclosed between the <blink> tag and its end tag (</blink>) - incessantly! It's not only ugly (reminiscent of the very bad video-text displays on a hotel TV), but it's excruciatingly annoying. The reader can't turn it off except to scroll beyond that portion of the document or hyperlink out of the document altogether. Section 4.5

Okay, so it grabs readers' attention to make an important point. Just make sure it's a *very* important point. And here's a tip: Make it easy for the reader to get by the blinking segment in your document. Surround it with empty space or with pleasant, but vacuous content that they don't have to read. Better yet, don't use the blink tag. Consign it to the scrap heap of Web history where it belongs, where it may rest in peace with counters for page visitors.

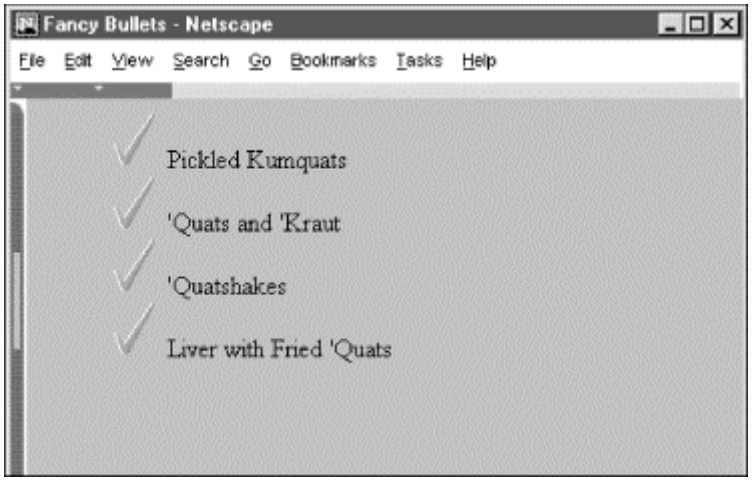
17.3 Custom Bullets

A common use of the definition list has nothing to do with definitions, but instead deals with adding custom bullets to an otherwise unordered list. In this trick, leave the <dt> tag empty, and add an <img> tag that references the desired bullet image at the beginning of each <dd> tag. Section 7.5.1 For example, in HTML (we'll let you add the necessary end tags to make it XHTML-compatible, too):

```
<dl>
  <dt><dd><img src="checkmark.gif"> Pickled Kumquats
  <dt><dd><img src="checkmark.gif"> 'Quats and 'Kraut
  <dt><dd><img src="checkmark.gif"> 'Quatshakes
  <dt><dd><img src="checkmark.gif"> Liver with Fried 'Quats
</dl>
```

The fancier list is shown in Figure 17-1.

Figure 17-1. Custom bullets for unordered lists



Keep in mind that this trick works well only if your list items are short enough to not wrap within the browser window. If the item does wrap, the next line will start aligned with the left edge of the bullet, not the left edge of the text, as you might hope.

Also keep in mind that the `list-style-image` property can be used to achieve the same result with a CSS-compliant browser. For example, the following code has the same result as our trick. The difference is that not all browsers are fully CSS-compliant, but our trick works with all of them. Section 8.4.7.1

```
<style>
ul.checks li {list-style-image: url("pics/checkmark.gif")}
</style>
...later in the document
<ul class="checks">
   <li> Pickled Kumquats </li>
   <li> 'Quats and 'Kraut </li>
   <li> 'Quatshakes </li>
   <li> Liver with Fried 'Quats </li>
</ul>
```

17.4 Tricks with Tables

Enough with the cheap tricks. On to some really good ones: tricks with the `<table>` tag and attributes that add some attractive features to your documents.

By design, tables let authors create appealing, accessible tables of information. But the table tags also can be exploited to create innovative, attractive page designs that are otherwise unattainable in standard HTML or XHTML.

17.4.1 Multicolumn Pages

One very common and popular page-layout element missing from the either HTML or XHTML standard is multiple columns of text. Although Netscape version 4 (and earlier) supports the `<multicol>` extension, for a more universal solution, place your document content inside a table of one row with two or more columns. Section 14.2

17.4.1.1 Basic multicolumn layout

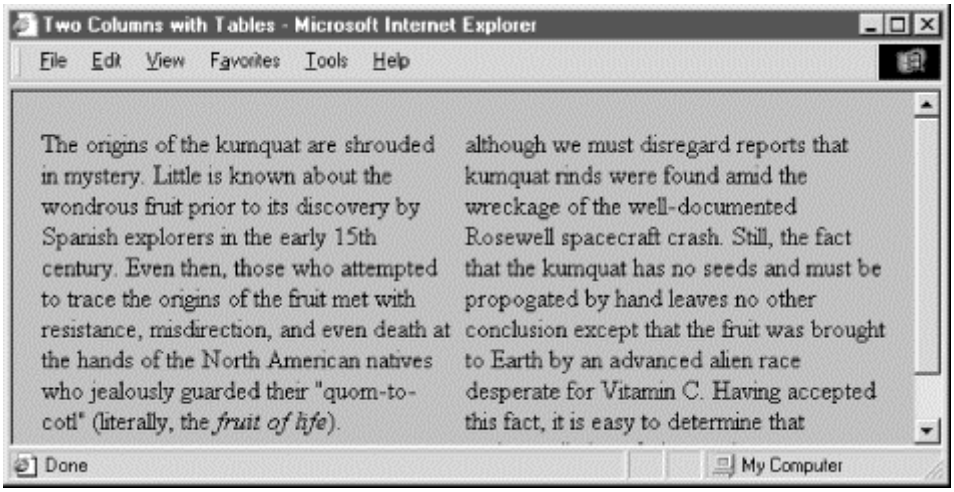
The basic two-column layout using `<table>` has a single table row with three data cells: one each for the columns of text and an intervening empty cell to more attractively separate the two columns. We've also added a large `cellspacing` attribute value to create additional intervening space between the columns.

The following HTML example table is an excellent template for a simple two-column text layout:

```
<table border=0 cellspacing=7>
  <tr>
    <td>Copy for column 1...
    <td><br>
    <td>Copy for column 2...
 </table>
```

See Figure 17-2 for the results.

Figure 17-2. A simple two-column layout



The one thing the browsers won't do is automatically balance the text in the columns, resulting in adjacent columns of approximately the same length. You'll have to experiment with your document, manually shifting text from one column to another until you achieve a nicely balanced page.

Keep in mind, though, that users may resize their display windows and the columns' contents will shift accordingly. So don't spend a lot of time getting the last sentences of each column to line up exactly; they're bound to be skewed in other browser window widths.

Of course, you can easily convert the example layout to three or more columns by dividing the text among more cells in the table. But keep in mind that pages with more than three columns may prove difficult to read on small displays where the actual column width might be quite small.

17.4.1.2 Straddle heads

The basic multicolumn format is just the start. By adding cells that span across the columns, you create headlines. Similarly, you can make figures span across more than one column: simply add the `colspan` attribute to the cell containing the headline or figure. Figure 17-3 shows an attractive three-column layout with straddle heads and a spanning figure, created from the following HTML source with table tags:

```
<table border=0 cellspacing=7>
  <tr>
    <th colspan=5><h2>The History of the Kumquat</h2>
  <tr valign=top>
    <td rowspan=2>Copy for column 1...
    <td rowspan=2 width=24><br>
    <td>Copy for column 2...
    <td width=24><br>
    <td>Copy for column 3...
  <tr>
    <td colspan=3 align=center><img src="pics/fruit.gif">
    <p>
    <i>The Noble Fruit</i>
</table>
```

To achieve this nice layout, we used the `colspan` attribute on the cell in the first row to span all five table columns (three with copy and the two intercolumn spaces). We use the `rowspan` attribute on the first column and its adjacent column spacer to extend the columns down beside the figure. The figure's cell has a `colspan` attribute so that the contents span the other two columns and intervening spaces.

17.4.2 Side Heads

The only text-heading features available in HTML and XHTML are the `<h1>` through `<h6>` tags. These tags are always embedded in the text flow, separating adjacent paragraphs of text. Through multiple columns, you can achieve an alternative style that places headings into a separate side column, running vertically alongside the document text.

Figure 17-4 shows you a fairly fancy pair of side heads, the result of the following bit of source XHTML table code:

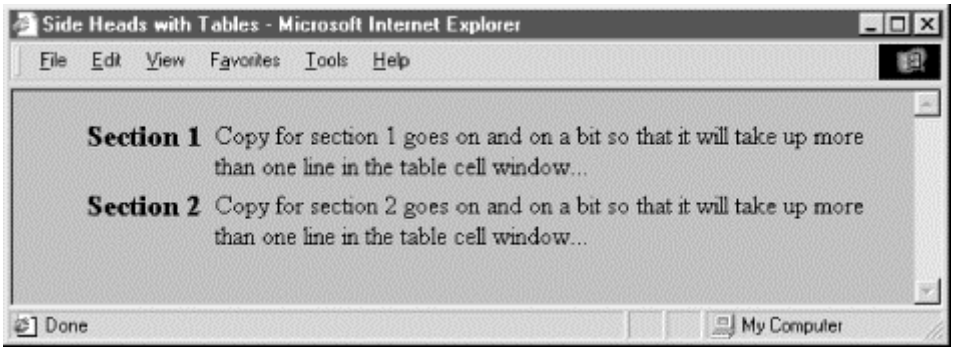
```
<table>
  <tr>
    <th width="20%" align="right">
      <h3>Section 1</h3></th>
    <td></td>
    <td>
      Copy for section 1 goes on and on a bit
      so that it will take up more than one line in the
      table cell window... </td>
  </tr>
  <tr>
    <th align="right">
      <h3>Section 2</h3></th>
    <td></td>
    <td>
      Copy for section 2 goes on and on a bit
      so that it will take up more than one line in the
      table cell window...</td>
  </tr>
</table>
```

Figure 17-3. Fancy straddle heads and spanning figures with HTML table tags

Notice how we create reasonably attractive side heads set off from the left margin of the browser window by adjusting the first header cell's width and right-justifying the cell contents.

Just as in our multicolumn layout, the example side-head layout uses an empty column to create a space between the narrow left column containing the heading and the wider right column containing the text associated with that heading. It's best to specify that column's width as a percentage of the table width, rather than explicitly in numbers of pixels to make sure that the heading column scales to fit both wide and narrow display windows.

Figure 17-4. Table tags created these side heads



17.4.2.1 When tables aren't implemented

One of the dangers of being overly dependent on tables is that your documents are usually unreadable when viewed with browsers that don't support tables. In the case of side heads, though, your document will come out just fine on a "table-challenged" browser.

Most browsers follow one of the Internet's basic tenets: be liberal in what you accept and strict in what you create. This usually means the browser will ignore tags that don't make sense, including all of the markup that creates a table.

In the case of our side-head layout, the browsers that can't do tables ignore the table tags and only see this part of the document:

```
<h3>Section 1</h3>
  Copy for section 1 goes on and on a bit
  so that it will take up more than one line in the
  table cell window...
<h3>Section 2</h3>
  Copy for section 2 goes on and on a bit
  so that it will take up more than one line in the
  table cell window...
```

Of course, this is a perfectly valid sequence of HTML that generates a conventional document with sections divided by `<h3>` headers. Your document will look fine, regardless of the browser, table-capable or not.

17.4.3 Better Forms Layout

Of all the features in HTML and XHTML, forms cry out for better layout control. Unlike other structured elements, forms look best when rendered in a fixed layout with precise margins and vertical alignment of elements. However, except for carefully planned `<pre>` formatted form segments, the common language just doesn't give us any special tools to better control forms layout.

17.4.3.1 Basic form layout

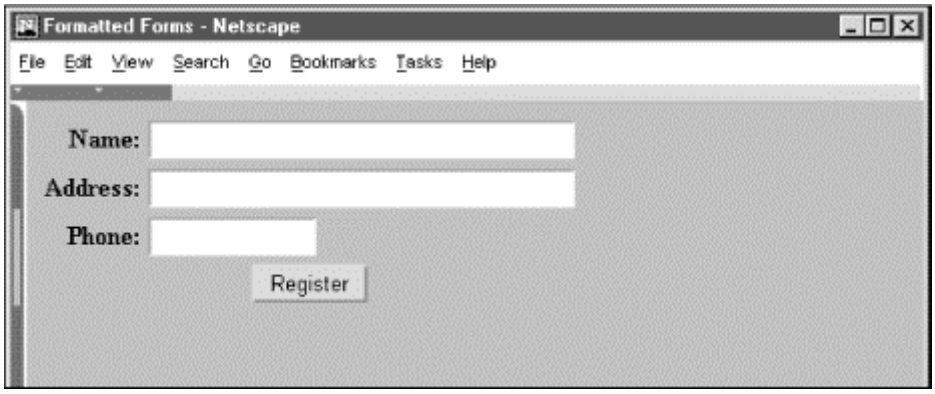
Your forms will almost always look better and be easier for your readers to follow if you use a table to structure and align the form's elements. For example, you might use a vertical alignment to your forms, with field labels to the left and their respective form elements aligned to an adjacent vertical margin on the right. Don't try that with just standard HTML or XHTML.

Rather, prepare a form that contains a two-column table. The following HTML source does just that, as shown in Figure 17-5:

```
<form method=post action="http:/cgi-bin/process">
  <table>
    <tr>
      <th align=right>Name:
      <td><input type=text size=32>
    <tr>
      <th align=right>Address:
      <td><input type=text size=32>
    <tr>
      <th align=right>Phone:
      <td><input type=text size=12>
    <tr>
      <td colspan=2 align=center>
      <input type=submit value="Register">
  </table>
</form>
```

Of course, more complex form layouts can be managed with tables. We recommend you first sketch the form layout on paper and plan how various combinations of table elements, including row- and column-straddled table cells might be used to effect the layout.

Figure 17-5. Align your forms nicely with tables



17.4.3.2 Building forms with nested tables

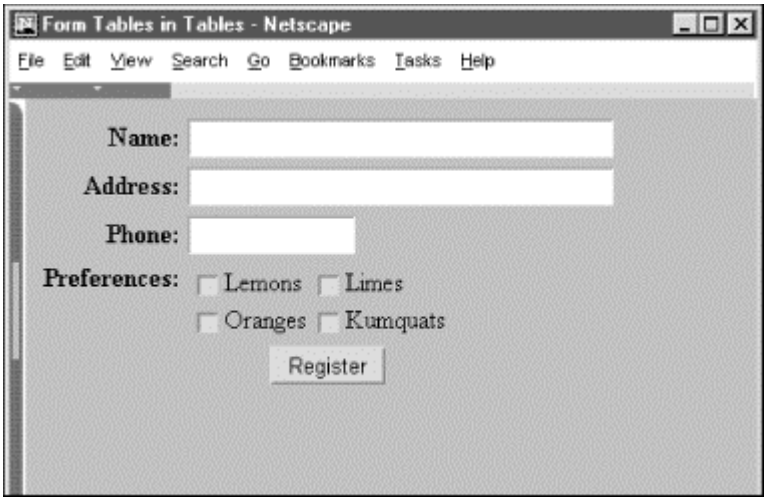
As we mentioned earlier, you may place a table inside the cell of another table. While this alone can lead to some elaborate table designs, nested tables also are useful for managing a subset of form elements within the larger table containing the entire form. The best application for using a nested table in a form is for laying out checkboxes and radio buttons.

For example, insert the following row containing a table into the form table in the previous example. It creates a checkbox with four choices:

```
<tr>
  <th align=right valign=top>Preferences:
  <td>
  <table>
    <tr>
    <td><input type=checkbox name=pref>Lemons
    <td><input type=checkbox name=pref>Limes
    <tr>
    <td><input type=checkbox name=pref>Oranges
    <td><input type=checkbox name=pref>Kumquats
  </table>
```

Figure 17-6 shows you how this nested table attractively formats the checkboxes, which browsers would otherwise render on a single line and not well aligned.

Figure 17-6. Nesting tables to format elements of a form



17.4.4 Embedded Guides

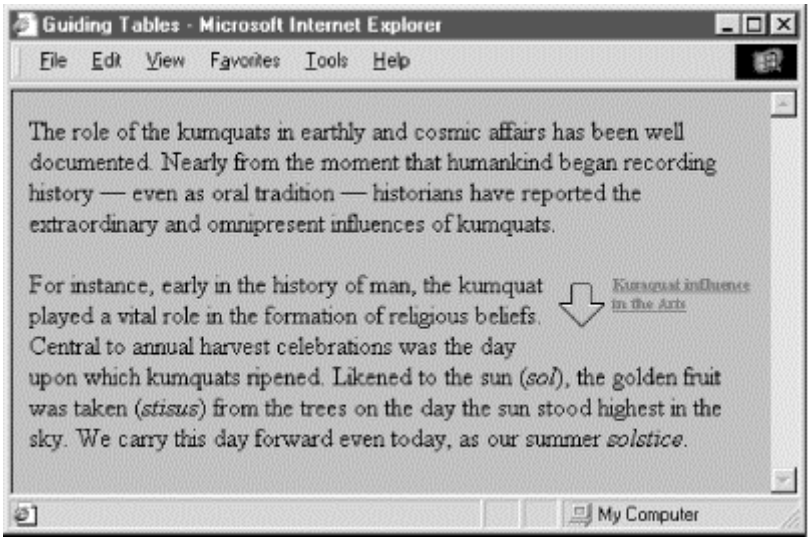
We generally argue for subtlety when you include hyperlinks in your documents, embedding them within the content and within context. But there are times when prominent guides to additional content are appropriate, like street signs in a crowded neighborhood.

Traditionally, HTML authors have placed their street signs (arrow icons with text labels) between major sections or at the beginning and end of the document to guide users back to a home page or on to the next page in the document series. Using `<table>` and its align attribute, you also can embed those guideposts within the document flow, but distinct from the content. And tables help you align the signpost elements for a more pleasing and concise presentation.

The following XHTML segment, for example, uses a two-column table to set a hyperlink guide apart from the document content. The technique also nicely aligns the guide's graphical and textual elements, thereby giving the reader a clear and distinct option to jump to another section of the document, as shown in Figure 17-7:

```
The role of the kumquats in earthly and cosmic affairs has been well documented.
Nearly from the moment that humankind began recording history - even as
oral tradition - historians have reported the extraordinary and
omnipresent influences of kumquats.</p>
<p>
<table align="right">
  <tr>
   <td><a href="#arts"><img src="pics/d_arrow.gif" border="0"></a></td>
   <td><a href="#arts"><h6>Kumquat influence<br/>in the Arts</h6></a></td>
  </tr>
</table>
For instance, early in the history of man, the kumquat played a vital role in the
formation of religious beliefs.  Central to annual harvest celebrations was the day
upon which kumquats ripened.  Likened to the sun (<em>sol</em>), the golden fruit
was taken (<em>stisus</em>) from the trees on the day the sun stood highest in the
sky.  We carry this day forward even today, as our summer <em>solstice</em>.

</p>
```

Figure 17-7. Tables let you embed signposts in your content flow.



17.5 Transparent Images

One of the most popular tricks you'll find on almost everyone's web pages is the transparent image. Transparent images let the background show through, giving the remainder of the image the appearance of floating on the page. The effect is clever, and it is the only way to put nonrectangular images into your document displays. Section 5.2.1

Creating a transparent image is easy, once you understand how the process works and which images are candidates for transparency.

17.5.1 Colors, Maps, and Indexes

Images represent their colors in one of two ways: directly or through a *colormap*.

In the direct method, each pixel in the image contains the actual RGB values that define the color of that pixel. Such images are often called *true color* images, since the number of distinct colors in the image is generally quite large. It is often the case that very few pixels in a true color image share the same color, with many pixels having subtly different variations of the same color. The most popular image format using this representation method is the JPEG format.

Colormap-based images keep all the different colors used in the image in a table known as the colormap. Each pixel in the image contains an index into the table of that pixel's color. In general, the table is fairly small, usually less than 256 colors. This means that many pixels share the same color, and that whole groups of pixels can have their color changed by simply altering the appropriate entry in the colormap. The most common image format using colormaps is the GIF format.

Image transparency is only possible with images containing a colormap and is currently defined only for images using the GIF89a format. In this format, one entry in the colormap is tagged as the transparent color. All pixels containing the index of that entry will be made transparent when the image is displayed.

For example, consider an image containing eight colors. The colormap is eight entries long, with indices numbered zero through seven. Each pixel in the image contains a value from zero to seven, corresponding to its color in the colormap. If you indicate that the second entry in the color map, whose index is one, will be transparent, all pixels with the value one will be made transparent when the image is rendered.

17.5.2 Creating a Potentially Transparent Image

The cookbook way to create a transparent image is easy: take a conventional image, determine the color to be made transparent, and convert the image to GIF89a format, marking that color as transparent.

The most difficult part for most people is finding a conventional image that is suitable for conversion. To make the background of an image transparent, the *entire* background must be one color. Unfortunately, many images do not meet this simple criteria. Scanned images, for example, usually have backgrounds that are a mix of several slightly different shades of one color. Since only one color can be made transparent, the result is a mottled background, part transparent and part opaque.

Many image-editing tools use a process known as *dithering* to create certain colors in an image. Dithered colors are not pure, but are a mix of several other colors. This mixture is not amenable to transparency. You'll often find dithering used on systems with small colormaps, like conventional 16-color VGA displays on some PCs.

Finally, some images have a pure background color, but that color is also used in parts of the image you want to keep opaque. Since every pixel having the appropriate colormap index is made transparent, these portions of the image become transparent as well.

In all cases, the problem can be solved by loading the image into an image editor, turning off dithering, and painting the background areas, usually by hand, to be a single color not used anywhere else in the image. Make sure you save the result as a GIF image, so that the colormap and pixel indexes will be retained.

17.5.3 Converting the Image

Once you have an acceptable image, and you've determined the color you wish to make transparent, you'll need to convert the image to GIF89a format.

For PC and Unix users, a public-domain utility called *giftrans* does the job nicely. To convert an image, use this command:

```
giftrans -t index original.gif > new.gif
```

Replace *index* with the numeric index of the color to be made transparent. *Original.gif* and *new.gif* are the original nontransparent image and the resulting transparent image.

Apple Macintosh users have the advantage, though: they can use a single tool named *Transparency* to accomplish the conversion. It was written by Aaron Giles at Cornell University, who generously makes it available at no charge over the Internet. Try searching for it on the Web.

Finally, several commercial products let you create transparent and interlaced GIF images, including Adobe PhotoShop and PaintShop Pro. Of course, these tools can do far more than simply convert transparent images. For a complete discussion of transparency and image conversion, including links to the actual tools, visit:

http://members.aol.com/htmlguru/transparent_images.html

17.6 Tricks with Windows and Frames

For the vast majority of links in your documents, you'll want the newly loaded document displayed in the same window, replacing the previous one. That makes sense, since your users usually follow a sequential path through your collection.

But sometimes it makes sense to open a document in a new window, so that the new document and the old document are both directly accessible on the user's screen. If the new document is related to the original, for instance, it makes sense to have both in view. Other times, you might want to open more than one document in multiple windows in a frameset. More commonly, the new document starts the user down a new web of documents, and you want them to see and remember where they came from.

Regardless of the reason, it is easy to open a new browser window from your document. All you need to do is add the `target` attribute in the appropriate hyperlink (`<a>`) tag.

17.6.1 Targeting Windows

We normally use the `target` attribute to load a document into a specific frame that you've named in a frameset. It also serves to create a new window by one of two methods:

Reference a new name

If you haven't previously defined a name and then use that new name as the value for the `target` attribute of a hyperlink, Netscape and Internet Explorer automatically create a new window with that name and load the referenced document into that window. This is the preferred way to create new windows, since you can subsequently use the name to load other documents into the same window. Using this technique, you can control which document gets loaded where.

Create an unnamed window

Some browsers like Netscape and Internet Explorer support a special target named `_blank`[3] that lets you create a new window. The `_blank` window has limited use, though, because it is nameless - you cannot direct any other documents into that window. (New documents loaded via hyperlinks selected by the user within the window get displayed in that same window, of course.)

[3] Some browsers also accept the name `_new`. If you can't get `_blank` to work with your browser, try `_new`.

17.6.2 Overriding Others' Targets

Ever visited a site whose home page is a frame document that never gives up? You know, the kind that leaves its great big logo on the top of the window and its site TOC running down the side of the display, staring you in the face long after you've hyperlinked away from the site? What if your site's frameset gets trapped into one of their window frames? What to do? (Apparently their webmasters hadn't heard about the `_blank` target.)

The short answer is to use JavaScript to force open a new window for your documents. But that, too, is potentially confusing for users, since they may already have a full window ready for your document. So, to embellish, let JavaScript discover whether your page is destined for a corner frame, or for the whole window.

Here is an example script that loads a webpage called *index2.html* into its own full window. Note the JavaScript-enabled browsers won't let you clear a previously loaded document display unless your document owns it. So, in the case where the target is not the whole window ("self" is not "window.top"), the example script opens a new window that becomes the target for your pages. The user may choose to close your document window and return to the other one, or vice versa:

```
<html>
<head>
<title>I need a window of my own</title>
<script language="JavaScript">
<!--
  if (self != window.top)
      window.open("http://www.kumquats.com/index2.html");
  else
    self.location.href = "http://www.kumquats.com/index2.html";
//-->
</script>
</head>
<body>
Your browser apparently doesn't support JavaScript. Please
<a href="http://www.kumquats.com/index2.html"> hyperlink to our site manually.</a>
</body>
</html>
```

17.6.3 Multiple Frames in One Link

Loading a new document from a hyperlink is a snap, even if you put the new document into an alternative frame or window from its hyperlink parent. Occasionally, though, you'll want to load documents into two frames when the user clicks just one link. With a bit of trickery you can load two or more frames at one time, provided they are arranged a certain way in the browser window.

Consider this frame layout:

```
<frameset rows=2>
  <frameset cols=2>
    <frame name=A>
    <frame name=B>
  </frameset>
  <frameset>
    <frame name=C>
    <frame name=D>
  </frameset>
</frameset>
```

If someone clicks a link in frame A, the only thing you can do is update one of the four frames. Suppose you wanted to update frames B and D at the same time?

The trick is to replace frames B and D with a single frame, like this:

```
<frameset cols=2>
  <frameset rows=2>
    <frame name=A>
    <frame name=C>
  </frameset>
  <frame name=BD>
</frameset>
```

Ahah! Now you have a single target in which to load a single document, frame BD. The document you load should contain the original frames B and D in one column, like this:

```
<frameset cols=2>
  <frame name=B>
  <frame name=D>
</frameset>
```

The two frames will fill frame BD. When you update frame BD, both frames will be replaced, giving the appearance of two frames being updated at one time.

The drawback to this is that the frames must be adjacent and able to be grouped into a single document. For most pages, though, this solution works fairly well.

We've only scratched the surface of HTML and XHTML tips and tricks here. Our advice: keep hacking!

Appendix A. HTML Grammar

For the most part, the exact syntax of an HTML or XHTML document is not rigidly enforced by a browser. This gives authors wide latitude in creating documents and gives rise to documents that work on most browsers, but are actually incompatible with the HTML and XHTML standards. Stick to the standards unless your documents are fly-by-night affairs.

The standards explicitly define the ordering and nesting of tags and document elements. This syntax is embedded within the appropriate Document Type Definition and is not readily understood by those not versed in SGML (for HTML 4.01, see Appendix D) or XML (for XHTML 1.0, see Appendix E). Accordingly, we provide an alternate definition of the allowable HTML and XHTML syntax, using a fairly common tool called a "grammar."

Grammar, whether it defines English sentences or HTML documents, is just a set of rules that indicates the order of language elements. These language elements can be divided into two sets: *terminal* (the actual words of the language) and *nonterminal* (all other grammatical rules). In HTML and XHTML, the words correspond to the embedded markup tags and text in a document.

To use the grammar to create a valid document, follow the order of the rules to see where the tags and text may be placed to create a valid document.

A.1 Grammatical Conventions

We use a number of typographic and punctuation conventions to make our grammar easy to understand.

A.1.1 Typographic and Naming Conventions

For our grammar, we denote the terminals with a bold, monospaced Courier typeface. The nonterminals appear in italicized text.

We also use a simple naming convention for the majority of our nonterminals: if one defines the syntax of a specific tag, its name will be the tag name followed by *_tag*. If a nonterminal defines the various language elements that may be nested within a certain tag, its name will be the tag name followed by *_content*.

For example, if you are wondering exactly which elements are allowed within an `<a>` tag, you can look for the *a_content* rule within the grammar. Similarly, to determine the correct syntax of a definition list created with the `<dl>` tag, look for the *dl_tag* rule.

A.1.2 Punctuation Conventions

Each rule in the grammar starts with the rule's name, followed by the replacement symbol (::=) and the rule's value. We've intentionally kept the grammar simple, but we do use three punctuation elements to denote alternation, repetition, and optional elements in the grammar.

A.1.2.1 Alternation

Alternation indicates a rule may actually have several different values, and you must choose exactly one of them. Vertical bars (|) separate the alternatives for the rule.

For example, the *heading* rule is equivalent to any one of six HTML heading tags, and so appears in the table as:

heading	::=	*h1_tag*
	|	*h2_tag*
	|	*h3_tag*
	|	*h4_tag*
	|	*h5_tag*
	|	*h6_tag*

The *heading* rule tells us that wherever the *heading* nonterminal appears in a rule, you can replace it with exactly one of the actual heading tags.

A.1.2.2 Repetition

Repetition indicates that an element within a rule may be repeated some number of times. Repeated elements are enclosed in curly braces ({...}). The closing brace has a subscripted number other than one if the element must be repeated a minimum number of times.

For example, the `<ul>` tag may only contain `<li>` tags, or it may actually be empty. The rule, therefore, is:

ul_tag	::=	`<ul>`
		{*li_tag* }o
		`</ul>`

The rule says that the syntax of the `<ul>` tag requires the `<ul>` tag, zero or more `<li>` tags, followed by a closing `</ul>` tag.

We spread this rule across several lines and indented some of the elements to make it more readable only; it does not imply that your documents must actually be formatted this way.

A.1.2.3 Optional elements

Some elements may appear in a document, but are not required. Optional elements are enclosed in square brackets ([and]).

The `<table>` tag, for example, has an optional caption:

table_tag	::=	`<table>`
		[*caption_tag*]
		{*tr_tag* }o
		`</table>`

In addition, the rule says that a table begins with the `<table>` tag, followed by an optional caption, zero or more table-row tags, and ends with the `</table>` tag.

A.1.3 More Details

Our grammar stops at the tag level; it does not delve further to show the syntax of each tag, including tag attributes. For these details, refer to the Quick Reference card included with this book.

A.1.4 Predefined Nonterminals

The HTML and XHTML standards define a few specific kinds of content that correspond to various types of text. We use these content types throughout the grammar. They are:

literal_text

: Text is interpreted exactly as specified; no character entities or style tags are recognized.

plain_text

: Regular characters in the document character encoding, along with character entities denoted by the ampersand character.

style_text

: Like *plain_text*, with physical- and content-based style tags allowed.

A.2 The Grammar

The grammar is a composite of the HTML 4.01 and XHTML 1.0 standard tags and special extensions to the language as supported by the latest versions of Netscape Navigator and Microsoft's Internet Explorer.

The rules are in alphabetical order. The starting rule for an entire document is named *html_document*.

a_tag	::=	<a>
		{*a_content* }o
		
a_content[a]	::=	*heading*
	|	*text*
abbr_tag	::=	<abbr> *text* </abbr>
acronym_tag	::=	<acronym> *text* </acronym>
address_tag	::=	<address>
		{*address_content* }o
		</address>
address_content	::=	*p_tag*
	|	*text*
applet_content	::=	{<param>}o
		body_content
applet_tag	::=	<applet>
		applet_content
		</applet>
b_tag	::=	*text*
basefont_tag	::=	<basefont>
		body_content
		</basefont>
bdo_tag	::=	<bdo> *text* </bdo>
big_tag	::=	<big> *text* </big>

blink_tag	::=	`<blink>` *text* `</blink>`
block	::=	{*block_content* }0
block_content	::=	`<isindex>`
	|	*basefont_tag*
	|	*blockquote_tag*
	|	*center_tag*
	|	*dir_tag*
	|	*div_tag*
	|	*dl_tag*
	|	*form_tag*
	|	*listing_tag*
	|	*menu_tag*
	|	*multicol_tag*
	|	*nobr_tag*
	|	*ol_tag*
	|	*p_tag*
	|	*pre_tag*
	|	*table_tag*
	|	*ul_tag*
	|	*xmp_tag*
blockquote_tag	::=	`<blockquote>`
		body_content
		`</blockquote>`

body_content	::=	<bgsound>
	|	<hr>
	|	*address_tag*
	|	*block*
	|	*del_tag*
	|	*heading*
	|	*ins_tag*
	|	*layer_tag*
	|	*map_tag*
	|	*marquee_tag*
	|	*text*
body_tag	::=	<body>
		{*body_content* }o
		</body>
caption_tag	::=	<caption>
		body_content
		</caption>
center_tag	::=	
		body_content
cite_tag	::=	<cite> *text* </cite>
code_tag	::=	<code> *text* </code>
colgroup_content	::=	{<col>}o
colgroup_tag	::=	<colgroup>
		colgroup_content

content_style	::=	*abbr_tag*
	|	*acronym_tag*
	|	*cite_tag*
	|	*code_tag*
	|	*dfn_tag*
	|	*em_tag*
	|	*kbd_tag*
	|	*q_tag*
	|	*strong_tag*
	|	*var_tag*
dd_tag	::=	<dd>
		flow
		</dd>
del_tag	::=	<del>
		flow
		</del>
dfn_tag	::=	<dfn> *text* </dfn>
dir_tag[b]	::=	<dir>
		{ *li_tag* }
		</dir>
div_tag	::=	
		body_content
dl_content	::=	*dt_tag dd_tag*

dl_tag	::=	<dl>
		{*dl_content* }
		</dl>
dt_tag	::=	<dt>
		text
		</dt>
em_tag	::=	*text*
fieldset_tag	::=	<fieldset>
		[*legend_tag*]
		{*form_content* }o
		</fieldset>
flow	::=	{*flow_content* }o
flow_content	::=	*block*
	|	*text*
font_tag	::=	 *style_text*
form_content[c]	::=	<input>
	|	<keygen>
	|	*body_content*
	|	*fieldset_tag*
	|	*label_tag*
	|	*select_tag*
	|	*textarea_tag*
form_tag	::=	<form>
		{*form_content* }o
		</form>

frameset_content	::=	<frame>
	|	*noframes_tag*
frameset_tag	::=	<frameset>
		{*frameset_content* }0
		</frameset>
h1_tag	::=	*text*
h2_tag	::=	*text*
h3_tag	::=	*text*
h4_tag	::=	*text*
h5_tag	::=	*text*
h6_tag	::=	*text*
head_content	::=	<base>
	|	<isindex>
	|	<link>
	|	<meta>
	|	<nextid>
	|	*style_tag*
	|	*title_tag*
	|	*script_tag*
head_tag	::=	<head>
		{*head_content* }0
		</head>

heading	::=	*h1_tag*
	|	*h2_tag*
	|	*h3_tag*
	|	*h4_tag*
	|	*h5_tag*
	|	*h6_tag*
html_content	::=	*head_tag body_tag*
	|	*head_tag frameset_tag*
html_document	::=	*html_tag*
html_tag	::=	<html>
		html_content
		</html>
i_tag	::=	*text*
ilayer_tag	::=	<ilayer>
		body_content
		</ilayer>
ins_tag	::=	<ins>
		flow
		</ins>
kbd_tag	::=	<kbd> *text* </kbd>
label_content[d]	::=	<input>
	|	*body_content*
	|	*select_tag*
	|	*textarea_tag*

label_tag	::=	<label>
		{*label_content* }o
		</form>
layer_tag	::=	<layer>
		body_content
		</layer>
legend_tag	::=	<legend> *text* </legend>
li_tag	::=	
		flow
listing_tag	::=	<listing>
		literal_text
		</listing>
map_content	::=	{<area>}o
map_tag	::=	<map>
		map_content
		</map>
marquee_tag	::=	<marquee>
		style_text
		</marquee>
menu_tag[e]	::=	<menu>
		{*li_tag* }o
		</menu>

multicol_tag	::=	<multicol>
		body_content
		</multicol>
nobr_tag	::=	<nobr> *text* </nobr>
noembed_tag	::=	<noembed> *text* </noembed>
noframes_tag	::=	<noframes>
		{*body_content* }o
		</noframes>
noscript_tag	::=	<noscript> *text* </noscript>
object_content	::=	{<param>}o
		body_content
object_tag	::=	<object>
		object_content
		</object>
ol_tag	::=	
		{*li_tag* }
optgroup_tag	::=	<optgroup>
		{*option_tag* }o
		</optgroup>
option_tag	::=	<option>
		plain_text
		</option>

p_tag	::=	
		text
physical_style	::=	*b_tag*
	|	*bdo_tag*
	|	*big_tag*
	|	*blink_tag*
	|	*font_tag*
	|	*i_tag*
	|	*s_tag*
	|	*small_tag*
	|	*span_tag*
	|	*strike_tag*
	|	*sub_tag*
	|	*sup_tag*
	|	*tt_tag*
	|	*u_tag*
pre_content	::=	
	|	<hr>
	|	*a_tag*
	|	*style_text*
pre_tag	::=	<pre>
		{*pre_content* }o
		</pre>
q_tag	::=	<q> *text* </q>

s_tag	::=	<s> *text* </s>
samp_tag	::=	<samp> *text* </samp>
script_tag[f]	::=	<script> *plain_text* </script>
select_content	::=	*optgroup_tag*
	|	*option_tag*
select_tag	::=	<select>
		{*select_content*}$_0$
		</select>
server_tag[g]	::=	<server> *plain_text* </server>
small_tag	::=	*text*
span_tag	::=	*text*
strike_tag	::=	<strike> *text* </strike>
strong_tag	::=	*text*
style_tag	::=	<style> *plain_text* </style>
sub_tag	::=	_{*text*}
sup_tag	::=	^{*text*}
table_cell	::=	*td_tag*
	|	*th_tag*
table_content	::=	<tbody>
	|	<tfoot>
	|	<thead>
	|	*tr_tag*

table_tag	::=	<table>
		[*caption_tag*]
		{*colgroup_tag* }o
		{*table_content* }o
		</table>
td_tag	::=	<td>
		body_content
		</td>
text	::=	{*text_content* }o
text_content	::=	
	|	<embed>
	|	<iframe>
	|	<img>
	|	<spacer>
	|	<wbr>
	|	*a_tag*
	|	*applet_tag*
	|	*content_style*
	|	*ilayer_tag*
	|	*noembed_tag*
	|	*noscript_tag*
	|	*object_tag*
	|	*plain_text*
	|	*physical_style*
textarea_tag	::=	<textarea> *plain_text* </textarea>

th_tag	::=	<th>
		body_content
		</th>
title_tag	::=	<title> *plain_text* </title>
tr_tag	::=	<tr>
		{*table_cell* }o
		</tr>
tt_tag	::=	<tt> *text* </tt>
u_tag	::=	<u> *text* </u>
ul_tag	::=	
		{*li_tag* }
var_tag	::=	<var> *text* </var>
xmp_tag	::=	<xmp>
		literal_text
		</xmp>

[a] *a_content* may not contain *a_tags*; you may not nest <a> tags within other <a> tags.

[b] The *li_tag* within the *dir_tag* may not contain any element found in a block.

[c] *form_content* may not contain *form_tags*; you may not nest one <form> within another <form>.

[d] As with the <form> tag, you cannot imbed <form> or <label> tags within a <label> tag.

[e] The *li_tag* within the *menu_tag* may not contain any element found in a block.

[f] A *script_tag* may be placed anywhere within an HTML document, without regard to syntactic rules.

[g] A *server_tag* may be placed anywhere within an HTML document, without regard to syntactic rules.

Appendix B. HTML/XHTML Tag Quick Reference

In this appendix, we list in alphabetical order all the known and some undocumented HTML and XHTML tags and attributes currently supported by one or more of today's popular browsers.

B.1 Core Attributes

Prior to HTML 4.0, there were few attributes that could be used consistently for all the HTML tags. HTML 4.0 changed this, defining a set of sixteen core attributes that can be applied to almost all the elements in both HTML 4.01 and XHTML 1.0. For brevity, we list these core attributes in this section and spare you the redundancies in the table that follows:

`class=`*name*	Specify a style class controlling the appearance of the tag's contents
`dir=`*dir*	Specify the rendering direction for text, either left-to-right (`ltr`) or right-to-left (`rtl`)
`id=`*name*	Define a reference name for the tag that is unique in the document
`lang=`*language*	Specify the human language for the tag's contents with an ISO 639 standard two-character name and optional dialect subcode
`onclick=`*applet*	Specify an applet to be executed when the user clicks the mouse on the tag's contents display area
`ondblclick=`*applet*	Specify an applet to be executed when the user double-clicks the mouse button on the tag's contents display area
`onkeydown=`*applet*	Specify an applet to be executed when the user presses down on a key while the tag's contents have input focus
`onkeypress=`*applet*	Specify an applet to be executed when the user presses and releases a key while the tag's contents have focus
`onkeyup=`*applet*	Specify an applet to be executed when the user releases a pressed key while the tag's contents have focus
`onmousedown=`*applet*	Specify an applet to be executed when the user presses down on the mouse button when pointing to the tag's contents display area
`onmousemove=`*applet*	Specify an applet to be executed when the user moves the mouse in the tag's contents display area
`onmouseout=`*applet*	Specify an applet to be executed when the user moves the mouse off the tag's contents display area
`onmouseover=`*applet*	Specify an applet to be executed when the user moves the mouse into the tag's contents display area
`onmouseup=`*applet*	Specify an applet to be executed when the user releases the mouse button when in the tag's contents display area
`style=`*style*	Specify an inline style for the tag
`title=`*string*	Specify a title for the tag

Only a small handful of tags accept none or only some, but not all, of these attributes. They are:

<applet>		<base>
<basefont>		<bdo>
 		<comment>
<embed>		<font>
<frame>		<frameset>
<head>		<hr>
<html>		<iframe>
<ilayer>		<isindex>
<keygen>		<layer>
<marquee>		<meta>
<multicol>		<nobr>
<noembed>		<param>
<script>		<server>
<spacer>		<style>
<title>	<wbr>	

For convenience, we've marked each of these tags with an asterisk (*) in the following table, and list all of the attributes supported by these special tags, including the common ones. For all other tags (those without an asterisk), assume the common attributes listed above apply. Do note, however, that the popular browsers *do not* support all the HTML standard 4.0 attributes, common or not. Please refer to the main text for details.

B.2 HTML Quick Reference

As with the other sections of this book, we use the Netscape and Internet Explorer icons to the far right of each item to indicate tags and attributes that are extensions to the HTML 4.01 and XHTML 1.0 standards. If no icon is shown, the tag or attribute is part of the HTML 4.01 and XHTML 1.0 standards.

We include each tag's possible attributes (some required) below their respective tags. In the description, we give possible attribute values as either a range of integer numbers or a definitive list of options, where possible.

<a> ... 		Create a hyperlink anchor (href attribute) or fragment identifier (name attribute)	
	accesskey=*char*	Define the hot-key character for this anchor	
	charset=*encoding*	Specify the character set used to encode the target	
	coords=*list*	Specify a list of shape-dependent coordinates	
	href=*url*	Specify the URL of a hyperlink target	
	hreflang=*language*	Specify the language encoding for the target	
	name=*name*	Specify the name of a fragment identifier	
	rel=*relationship*	Indicate the *relationship* from this document to the target	
	rev=*relationship*	Indicate the reverse *relationship* of the target to this document	
	shape=*shape*	Define the region's shape to be circ, circle, poly, polygon, rect, or rectangle	
	tabindex=*value*	Define the position of this anchor inthe document's tabbing order	
	target=*name*	Define the name of the frame or window to receive the referenced document	

<a> ... (cont...)	type=*type*	Specify the MIME type of the target	
<abbr> ... </abbr>		The enclosed text is an abbreviation	
<acronym> ... </acronym>		The enclosed text is an acronym	
<address> ... </address>		The enclosed text is an address	
<applet> ... </applet>		Define an executable applet within a text flow	*
	align=*position*	Align the <applet> region to either the top, middle, bottom (default), left, right, absmiddle, baseline, or absbottom of the text in the line	
	alt=*string*	Specify alternative text to replace the <applet> region within browsers that support the <applet> tag, but cannot execute the application	
	archive=*url*	Specify a class archive to be downloaded to the browser and then searched for code class	
	class=*name*	Specify a style class controlling the appearance of this tag	
	code=*class*	Specify the class name of the code to be executed (required)	
	codebase=*url*	URL from which the code is retrieved	
	height=*n*	Specify the height, in pixels, of the <applet> region	
	hspace=*n*	Specify additional space, in pixels, to allow to the left and right of the <applet> region	
	id=*name*	Define a name for this applet that is unique to this document	
	mayscript	If present, allows the applet to access JavaScript within the page	N
	name=*name*	Specify the name of this particular instance of the <applet>	
	object=*data*	Specify a representation of the object's execution state	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Provide a title for the applet	
	vspace=*n*	Specify additional space, in pixels, to allow above and below the <applet> region	

<applet> ... </applet> (cont...)	width=*n*	Specify the width, in pixels, of the <applet> region	
<area>		Define a mouse-sensitive area in a client-side image map	
	accesskey=*char*	Define the hot-key character for this area	
	alt=*string*	Provide alternative text to be displayed by nongraphical browsers	
	coords=*list*	Specify a comma-separated *list* of shape-dependent coordinates that define the edge of this area	
	href=*url*	Specify the URL of a hyperlink target associated with this area	
	nohref	Indicate that no document is associated with this area; clicking in the area has no effect	
	notab	Do not include this area in the tabbing order	O
	onblur=*applet*	Specify an applet to be run when the mouse leaves the area	
	onfocus=*applet*	Specify an applet to be run when the mouse enters the area	
	shape=*shape*	Define the region's shape to be circ, circle, poly, polygon, rect, or rectangle	
	tabindex=*value*	Define the position of this area in the document's tabbing order	
	taborder=*n*	Specify this area's position in the tabbing order	O
	target=*name*	Specify the frame or window to receive the document linked by this area	N O
...		Format the enclosed text using a bold typeface	
<base>		Specify the base URL for all relative URLs in this document	*
	href=*url*	Specify the base URL (required)	
	target=*name*	Define the default target of all <a> links in the document	
<basefont>		Specify the font size for subsequent text	*
	color=*color*	Specify the base font's color	

<basefont> (cont...)	face=*name*	Specify local font to be used for the base font	
	id=*name*	Define a name for this tag that is unique to this document	
	name=*name*	Specify local font to be used for the base font	I
	size=*value*	Set the basefont size of 1 to 7 (required; default is 3)	
<bdo> ... </bdo>		Bidirectional override, changing the rendering direction of the enclosed text	*
	class=*name*	Specify a style class controlling the appearance of this tag	
	dir=*dir*	Specify the rendering direction for text, either left-to-right (ltr) or right-to-left (rtl)	
	id=*name*	Define a name for this tag that is unique to this document	
	lang=*language*	Specify the language used for this tag's contents using a standard two-character ISO language name	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Specify a title for this tag	
<bgsound>		Define background audio for the document	*
	loop=*value*	Set the number of times to play the audio; *value* may be an integer or the value infinite	
	src=*url*	Provide the URL of the audio file to be played	
<big> ... </big>		Format the enclosed text using a bigger typeface	
<blink> ... </blink>		Cause the enclosed content to blink	N
<blockquote> ... </blockquote>		The enclosed text is a block quotation	
	cite=*url*	Specify the URL of source of the quoted material	
<body> ... </body>		Delimit the beginning and end of the document body	
	alink=*color*	Set the color of active hypertext links in the document	
	background=*url*	Specify the URL of an image to be tiled in the document background	

<body> ... </body> (cont...)	bgcolor=*color*	Set the background color of the document	
	bgproperties=*value*	With *value* set to fixed, prevent the background image from scrolling with the document content	IE
	leftmargin=*value*	Set the size, in pixels, of the document's left margin	IE
	link=*color*	Set the color of unvisited hypertext links in the document	
	onblur=*applet*	Specify an applet to be run when the mouse leaves the document window	N IE
	onfocus=*applet*	Specify an applet to be run when the mouse enters the document window	N IE
	onload=*applet*	Specify an applet to be run when the document is loaded	
	onunload=*applet*	Specify an applet to be run when the document is unloaded	
	text=*color*	Set the color of regular text in the document	
	topmargin=*value*	Set the size, in pixels, of the document's top margin	IE
	vlink=*color*	Set the color of visited links in the document	
 		Break the current text flow, resuming at the beginning of the next line	*
	class=*name*	Specify a style class controlling the appearance of this tag	
	clear=*margin*	Break the flow and move downward until the desired *margin*, either left, right, none, or all, is clear	
	id=*name*	Define a name for this tag that is unique to this document	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Specify a title for this tag	
<button>		Create a pushbutton element within a <form>	
	accesskey=*char*	Define the hot-key character for this button	
	disabled	Disable the button, preventing the user from clicking it	

<button> (cont...)	name=*name*	Specify the name of the parameter to be passed to the form-processing application if the input element is selected (required)	
	onblur=*applet*	Specify an applet to be run when the mouse moves out of the button	
	onfocus=*applet*	Specify an applet to be run when the mouse moves into the button	
	tabindex=*n*	Specify this element's position in the tabbing order	
	type=*type*	Specify the button type, either button, submit, or reset	
	value=*string*	Specify the value of the parameter sent to the form-processing application if this form element is selected (required)	
<caption> ... </caption>		Define a caption for a table	
	align=*position*	For Netscape, set the vertical position of the caption to either top or bottom. For Internet Explorer, set the horizontal alignment of the caption to either left, center, or right.	
	valign=*position*	Set the vertical position of the caption to either top or bottom	
...		Center the enclosed text	
<cite> ... </cite>		The enclosed text is a citation	
<code> ... </code>		The enclosed text is a code sample	
<col>		Define a column within a <colgroup>	
	align=*position*	Set the column alignment to left, center, or right	
	char=*character*	Specify the alignment character for text in these cells	
	charoff=*value*	Set the offset within the cell at which the alignment character will be placed	
	span=*n*	The number of columns affected by this <col> tag	
	valign=*position*	Set the vertical alignment of text within the column to either top, middle, or bottom	
	width=*n*	Set the width, in pixels or as a percentage, of the column	

<colgroup>		Define a column group within a table	
	align=*position*	Set the horizontal alignment of text within the columns to either left, center, or right	
	char=*character*	Specify the alignment character for text in these cells	
	charoff=*value*	Set the offset within the cell at which the alignment character will be placed	
	span=*n*	Define the number of columns in the group	
	valign=*position*	Set the vertical alignment of text within the columns to either top, middle, or bottom	
	width=*n*	Set the width, in pixels or as a percentage, of each column in the group	
<comment> ... </comment>		Place a comment in the document (comments will be visible in all other browsers)	O *
<dd> ... </dd>		Define the definition portion of an element in a definition list	
<del> ... </del>		Delineate a deleted section of a document	
	cite=*url*	Cite a document justifying the deletion	
	datetime=*date*	Specify the date and time of the deletion	
<dfn> ... </dfn>		Format the enclosed text as a definition	
<dir> ... </dir>		Create a directory list containing tags	
	compact	Make the list more compact if possible	
	type=*bullet*	Set the bullet style for this list to either circle, disc (default), or square	N O
...		Create a division within a document	
	align=*type*	Align the text within the division to left, center, or right	
	nowrap	Suppress word wrapping within this division	O
<dl> ... </dl>		Create a definition list containing <dt> and <dd> tags	
	compact	Make the list more compact if possible	
<dt> ... </dt>		Define the definition term portion of an element in a definition list	

...		Format the enclosed text with additional emphasis	
<embed>		Embed an application in a document	N O *
	align=*position*	Align the applet area to either the top or bottom of the adjacent text, or to the left or right margin of the page, with subsequent text flowing around the applet	N
	border=*n*	Specify the size, in pixels, of the border around the applet	N
	height=*n*	Specify the height, in pixels, of the applet	N O
	hidden	If present, hide the applet on the page	N O
	hspace=*n*	Define, in pixels, additional space to be placed to the left and right of the applet	N
	name=*name*	Provide a name for the applet	N O
	palette=*value*	In Netscape, a value of foreground causes the applet to use the foreground palette in Windows only; background uses the background palette. In Internet Explorer, provides the foreground and background colors for the applet, specified as two color values separated by a vertical bar (|)	N O
	pluginspage=*url*	Provide the URL of the page containing instructions for installing the plug-in associated with the applet	N
	src=*url*	Supply the URL of the data to be fed to the applet	N O
	type=*type*	Specify the MIME type of the plug-in to be used	N
	units=*type*	Set the units for the height and width attributes to either pixels (the default) or en (half the text point size)	N O
	vspace=*n*	Define, in pixels, additional space to be placed above and below the applet	N
	width=*n*	Specify the width, in pixels, of the applet	N O
<fieldset> ... </fieldset>		Create a group of elements in a form	
 ... 		Set the size or color of the enclosed text	*

 ... (cont...)	class=*name*	Specify a style class controlling the appearance of this tag	
	color=*color*	Set the color of the enclosed text to the desired color	
	dir=*dir*	Specify the rendering direction for text, either left-to-right (ltr) or right-to-left (rtl)	
	face=*list*	Set the typeface of the enclosed text to the first available font in the comma-separated list of font names	
	id=*name*	Define a name for this tag that is unique to this document	
	lang=*language*	Specify the language used for this tag's contents using a standard two-character ISO language name	
	size=*value*	Set the size to absolute size 1 to 7, or relative to the <basefont> size using +n or -n (required)	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Specify a title for this tag	
<form> ... </form>		Delimit a form	
	accept-charset=*list*	Specify a list of character sets accepted by the server processing this form	
	action=*url*	Specify the URL of the application that will process the form (required)	
	enctype=*encoding*	Specify how the form element values will be encoded	
	method=*style*	Specify the parameter-passing *style*, either get or post (required)	
	name=*name*	Supply a name for this form for use by JavaScript	N
	onreset=*applet*	Specify an applet to be run when the form is reset	
	onsubmit=*applet*	Specify an applet to be run when the form is submitted	
	target=*name*	Specify the name of the frame or window to receive the results of the form after submission	
<frame> ... </frame>		Define a frame within a frameset	*
	bordercolor=*color*	Set the color of the frame's border to *color*	N O

<frame> ... </frame> (cont...)	class=*name*	Specify a style class controlling the appearance of this tag	
	frameborder=*n*	If *value* is yes (Netscape only) or 1 (Netscape and Internet Explorer), enable frame borders. If *value* is no (Netscape only) or 0 (Netscape and Internet Explorer), disable frame borders.	
	id=*name*	Define a name for this tag that is unique to this document	
	longdesc=*url*	Provide the URL of a document describing the contents of the frame	
	marginheight=*n*	Place *n* pixels of space above and below the frame contents	
	marginwidth=*n*	Place *n* pixels of space to the left and right of the frame contents	
	name=*name*	Define the name of the frame	
	noresize	Disable user resizing of the frame	
	scrolling=*type*	Always add scrollbars (yes), never add scrollbars (no), or for Netscape only, add scrollbars when needed (auto)	
	src=*url*	Define the URL of the source document for this frame	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Specify a title for this tag	
<frameset> ... </frameset>		Define a collection of frames or other framesets	*
	border=*n*	Set the thickness of the frame borders in this frameset	NO
	bordercolor=*color*	Define the color of the borders in this frameset	NO
	cols=*list*	Specify the number and width of frames within a frameset	
	frameborder=*value*	If *value* is yes (Netscape only) or 1 (Netscape and Internet Explorer), enable frame borders. If *value* is no (Netscape only) or 0 (Netscape and Internet Explorer), disable frame borders.	NO
	framespacing=*n*	Define the thickness of the frame borders in this frameset	O

<frameset> ... </frameset> (cont...)	onblur=*applet*	Define an applet to be run when the mouse leaves this frameset	
	onfocus=*applet*	Define an applet to be run when the mouse enters this frameset	
	onload=*applet*	Define an applet to be run when this frameset is loaded	
	onunload=*applet*	Define an applet to be run when this frameset is removed from the display	
	rows=*list*	Specify the number and height of frames within a frameset	
<h*n*> ... </h*n*>		The enclosed text is a level *n* header; for level *n* from 1 to 6	
	align=*type*	Specify the heading alignment as either left (default), center, or right	
<head> ... </head>		Delimit the beginning and end of the document head	*
	dir=*dir*	Specify the rendering direction for text, either left-to-right (ltr) or right-to-left (rtl)	
	lang=*language*	Specify the language used for this tag's contents using a standard two-character ISO language name	
	profile=*url*	Provide the URL of a profile for this document	
<hr>		Break the current text flow and insert a horizontal rule	*
	align=*type*	Specify the rule alignment as either left, center (default), or right	
	class=*name*	Specify a style class controlling the appearance of the rule	
	color=*color*	Define the color of the rule	I
	id=*name*	Define a name for this tag that is unique to this document	
	noshade	Do not use 3D shading to render the rule	
	onclick=*applet*	Specify an applet to be executed when the mouse button is clicked on this tag	
	ondblclick=*applet*	Specify an applet to be executed when the mouse button is double-clicked on this tag	

<hr> (cont...)	onkeydown=*applet*	Specify an applet to be executed when a key is pressed down while this tag has input focus	
	onkeypress=*applet*	Specify an applet to be executed when a key is pressed and released while this tag has focus	
	onkeyup=*applet*	Specify an applet to be executed when a key is released while this tag has focus	
	onmousedown=*applet*	Specify an applet to be executed when a mouse button is pressed down on this tag	
	onmousemove=*applet*	Specify an applet to be executed when the mouse is moved over this tag	
	onmouseout=*applet*	Specify an applet to be executed when the mouse out of this tag's display area	
	onmouseover=*applet*	Specify an applet to be executed when the mouse moves into this tag's display area	
	onmouseup=*applet*	Specify an applet to be executed when a mouse button is released while over this tag	
	size=*pixels*	Set the thickness of the rule to an integer number of pixels	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Specify a title for this tag	
	width=*value* or *%*	Set the width of the rule to either an integer number of pixels or a percentage of the page width	
<html> ... </html>		Delimit the beginning and end of the entire HyperText Markup Language document	*
	dir=*dir*	Specify the rendering direction for text, either left-to-right (ltr) or right-to-left (rtl)	
	lang=*language*	Specify the language used for this tag's contents using a standard two-character ISO language name	
	version=*string*	Indicate the HTML version used to create this document	
...		Format the enclosed text in an *italic* typeface	
<iframe> ... </iframe>		Define an inline frame	*
	align=*position*	Set the position of the frame aligned to the top, center, or bottom of the surrounding text, or flush against the left or right margins with subsequent text flowing around the frame	

<iframe> ... </iframe> (cont...)	class=*name*	Specify a style class controlling the appearance of the frame	
	frameborder=*value*	If *value* is 1, enable frame borders. If *value* is 0, disable frame borders.	
	height=*n*	Set the height, in pixels, of the frame	
	id=*name*	Define a name for this tag that is unique to this document	
	longdesc=*url*	Provide the URL of a document describing the contents of the frame	
	marginheight=*n*	Place *n* pixels of space above and below the frame contents	
	marginwidth=*n*	Place *n* pixels of space to the left and right of the frame contents	
	name=*name*	Define the name of the frame	
	scrolling=*type*	Always add scrollbars (yes) or never add scrollbars (no)	
	src=*url*	Define the URL of the source document for this frame	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Specify a title for this tag	
	width=*n*	Set the width, in pixels, of the frame	
<ilayer> ... </ilayer>		Define an inline layer	N *
	above=*name*	Place this layer above the named layer	N
	background=*url*	Specify a background image for the layer	N
	below=*name*	Place this layer below the named layer	N
	bgcolor=*color*	Specify the background color for the layer	N
	class=*name*	Specify a style class controlling the appearance of this tag	N
	clip=*edge*	Define the layer's clipping region, in pixels. If *left* and *top* are 0, they may be omitted	N
	left=*n*	Define, in pixels, the position of the layer's left edge from the containing line of text	N

<ilayer> ... </ilayer> (cont...)	name=*name*	Provide a name for the layer	N
	src=*url*	Supply the content of the layer from another document	N
	style=*style*	Specify an inline style for this tag	N
	top=*n*	Define, in pixels, the position of the layer's top edge from the containing line of text	N
	visibility=*value*	Determine whether to show the layer, hide the layer, or inherit the visibility attribute from a containing layer	N
	width=*n*	Define the width, in pixels, of the layer	N
	z-index=*n*	Specify the layer's position in the stacking order	N
<img>		Insert an image into the current text flow	
	align=*type*	Align the image to either the top, middle, bottom (default), left, or right of the text in the line. For Netscape Navigator, additionally to the absmiddle, baseline, or absbottom of the text.	
	alt=*text*	Provide text for non-image-capable browsers	
	border=*n*	Set the pixel thickness of the border around images contained within hyperlinks	
	controls	Add playback controls for embedded video clips	I
	dynsrc=*url*	Specify the URL of a video clip to be displayed	I
	height=*n*	Specify the height of the image in scan lines	
	hspace=*n*	Specify the space, in pixels, to be added to the left and right of the image	
	ismap	Indicate that the image is mouse-selectable when used within an <a> tag	
	longdesc=*url*	Provide the URL of a document describing the image	
	loop=*value*	Set the number of times to play the video; *value* may be an integer or the value infinite	I
	lowsrc=*url*	Specify a low-resolution image to be loaded by the browser first, followed by the image specified by the src attribute	N
	name=*name*	Provide a name for the image for use by JavaScript	N

<img> (cont...)	onabort=*applet*	Provide an applet to be run if the loading of the image is aborted	N
	onerror=*applet*	Provide an applet to be run if the loading of the image is unsuccessful	N
	onload=*applet*	Provide an applet to be run if the loading of the image is successful	N
	src=*url*	Specify the source URL of the image to be displayed (required)	
	start=*start*	Specify when to play the video clip, either fileopen or mouseover	I
	usemap=*url*	Specify the map of coordinates and links that define the hypertext links within this image	
	vspace=*n*	Specify the vertical space, in pixels, added at the top and bottom of the image	
	width=*n*	Specify the width of the image in pixels	
<input type=button>		Create a pushbutton element within a <form>	
	accesskey=*char*	Define the hot-key character for this element	
	disabled	Disable this control, making it inactive	
	name=*name*	Specify the name of the parameter to be passed to the form-processing application if the input element is selected (required)	
	notab	Specify that this element is not part of the tabbing order	I
	onblur=*applet*	Specify an applet to be run when the mouse leaves this control	
	onfocus=*applet*	Specify an applet to be run when the mouse enters this control	
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	I
	value=*string*	Specify the value of the parameter sent to the form-processing application if this form element is selected (required)	
<input type=checkbox>		Create a checkbox input element within a <form>	

<input type=checkbox> (cont...)	accesskey=*char*	Define the hot-key character for this element	
	checked	Mark the element as initially selected	
	disabled	Disable this control, making it inactive	
	name=*string*	Specify the name of the parameter to be passed to the form-processing application if the input element is selected (required)	
	notab	Specify that this element is not part of the tabbing order	N
	readonly	Prevent user modification of this element	
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	N
	value=*string*	Specify the value of the parameter sent to the form-processing application if this form element is selected (required)	
<input type=file>		Create a file-selection element within a <form>	
	accept=*list*	Specify list of MIME types that can accepted by this element	
	accesskey=*char*	Define the hot-key character for this element	
	disabled	Disable this control, making it inactive	
	maxlength=*n*	Specify the maximum number of characters to accept for this element	
	name=*name*	Specify the name of the parameter that is passed to the form-processing application for this input element (required)	
	notab	Specify that this element is not part of the tabbing order	N
	onblur=*applet*	Specify an applet to be run when the mouse leaves this control	
	onchange=*applet*	Specify an applet to be run when the user changes the value of this element	
	onfocus=*applet*	Specify an applet to be run when the mouse enters this control	
	readonly	Prevent user modification of this element	

<input type=file> (cont...)	size=*n*	Specify the number of characters to display for this element	
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	
	value=*string*	Specify the value of the parameter sent to the form-processing application if this form element is selected (required)	
<input type=hidden>		Create a hidden element within a <form>	
	name=*name*	Specify the name of the parameter that is passed to the form-processing application for this input element (required)	
	value=*string*	Specify the value of this element that is passed to the form-processing application	
<input type=image>		Create an image input element within a <form>	
	accesskey=*char*	Define the hot-key character for this element	
	align=*type*	Align the image to either the top, middle, or bottom of the form element's text	
	alt=*string*	Provide an alternative description for the image	
	border=*n*	Set the pixel thickness of the border of the image	
	disabled	Disable this control, making it inactive	
	name=*name*	Specify the name of the parameter to be passed to the form-processing application for this input element (required)	
	notab	Specify that this element is not part of the tabbing order	
	src=*url*	Specify the source URL of the image (required)	
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	
	usemap=*url*	Specify the URL of a map to be used within this image	
<input type=password>		Create a content-protected text-input element within a <form>	

<input type=password> (cont...)	accesskey=*char*	Define the hot-key character for this element	
	disabled	Disable this control, making it inactive	
	maxlength=*n*	Specify the maximum number of characters to accept for this element	
	name=*name*	Specify the name of the parameter to be passed to the form-processing application for this input element (required)	
	notab	Specify that this element is not part of the tabbing order	
	onblur=*applet*	Specify an applet to be run when the mouse leaves this element	
	onchange=*applet*	Specify an applet to be run when the user changes the value of this element	
	onfocus=*applet*	Specify an applet to be run when the mouse enters this element	
	onselect=*applet*	Specify an applet to be run if the user clicks this element	
	readonly	Prevent user modification of this element	
	size=*n*	Specify the number of characters to display for this element	
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	
	value=*string*	Specify the initial value for this element	
<input type=radio>		Create a radio-button input element within a <form>	
	accesskey=*char*	Define the hot-key character for this element	
	checked	Mark the element as initially selected	
	disabled	Disable this control, making it inactive	
	name=*string*	Specify the name of the parameter to be passed to the form-processing application if the input element is selected (required)	
	notab	Specify that this element is not part of the tabbing order	

<input type=radio> (cont...)	readonly	Prevent user modification of this element	
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	I
	value=*string*	Specify the value of the parameter sent to the form-processing application if this form element is selected (required)	
<input type=reset>		Create a reset button within a <form>	
	accesskey=*char*	Define the hot-key character for this element	
	disabled	Disable this control, making it inactive	
	notab	Specify that this element is not part of the tabbing order	I
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	I
	value=*string*	Specify an alternate label for the reset button (default is "Reset")	
<input type=submit>		Create a submit button within a <form>	
	accesskey=*char*	Define the hot-key character for this element	
	disabled	Disable this control, making it inactive	
	name=*name*	Specify the name of the parameter that is passed to the form-processing application for this input element (required)	
	notab	Specify that this element is not part of the tabbing order	I
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	I
	value=*string*	Specify an alternate label for the submit button, as well as the value passed to the form-processing application for this parameter if this button is clicked	
<input type=text>		Create a text input element within a <form>	
	accesskey=*char*	Define the hot-key character for this element	

<input type=text> (cont...)	disabled	Disable this control, making it inactive	
	maxlength=*n*	Specify the maximum number of characters to accept for this element	
	name=*name*	Specify the name of the parameter that is passed to the form-processing application for this input element (required)	
	notab	Specify that this element is not part of the tabbing order	I
	onblur=*applet*	Specify an applet to be run when the mouse leaves this element	
	onchange=*applet*	Specify an applet to be run when the user changes the value of this element	
	onfocus=*applet*	Specify an applet to be run when the mouse enters this element	
	onselect=*applet*	Specify an applet to be run if the user clicks this element	
	readonly	Prevent user modification of this element	
	size=*n*	Specify the number of characters to display for this element	
	tabindex=*n*	Specify this element's position in the tabbing order	
	taborder=*n*	Specify this element's position in the tabbing order	I
	value=*string*	Specify the initial value for this element	
<ins> ... </ins>		Delineate an inserted section of a document	
	cite=*url*	Cite a document justsifying the insertion	
	datetime=*date*	Specify the date and time of the insertion	
<isindex>		Create a "searchable" HTML document	*
	action=*url*	For Internet Explorer only, provide the URL of the program that will perform the searching action	I
	class=*name*	Specify a style class controlling the appearance of this tag	
	dir=*dir*	Specify the rendering direction for text, either left-to-right (ltr) or right-to-left (rtl)	

<isindex> (cont...)	id=*name*	Define a name for this tag that is unique to this document	
	lang=*language*	Specify the language used for this tag's contents using a standard two-character ISO language name	
	prompt=*string*	Provide an alternate prompt for the input field	
	style=*style*	Specify an inline style for this tag	
	title=*string*	Specify a title for this tag	
<kbd> ... </kbd>		The enclosed text is keyboard-like input	
<keygen>		Generate key information in a form	N
	challenge=*string*	Provide a challenge string to be packaged with the key	N
	name=*name*	Provide a name for the key	
<label> ... </label>		Define a label for a form control	
	accesskey=*char*	Define the hot-key character for this label	
	for=*name*	Specify the form element associated with this label	
	onblur=*applet*	Specify an applet to be run when the mouse leaves this label	
	onfocus=*applet*	Specify an applet to be run when the mouse enters this label	
<layer> ... </layer>		Define a layer	N *
	above=*name*	Place this layer above the named layer	N
	background=*url*	Specify a background image for the layer	N
	below=*name*	Place this layer below the named layer	N
	bgcolor=*color*	Specify the background color for the layer	N
	class=*name*	Specify a style class controlling the appearance of this tag	N
	clip=*edge*	Define the layer's clipping region, in pixels. If *left* and *top* are 0, they may be omitted	N

<layer> ... </layer> (cont...)	left=*n*	Define, in pixels, the position of the layer's left edge from the containing document or layer	N
	name=*name*	Provide a name for the layer	N
	src=*url*	Supply the content of the layer from another document	N
	style=*style*	Specify an inline style for this tag	N
	top=*n*	Define, in pixels, the position of the layer's top edge from the containing document or layer	N
	visibility=*value*	Determine whether to show the layer, hide the layer, or inherit the visibility attribute from a containing layer	N
	width=*n*	Define the width, in pixels, of the layer	N
	z-index=*n*	Specify the layer's position in the stacking order	N
<legend> ... </legend>		Define a legend for a form field set	
	accesskey=*char*	Define the hot-key character for this legend	
	align=*position*	Align the legend to the top, bottom, left, or right of the field set	
...		Delimit a list item in an ordered ( ) or unordered ( ) list	
	type=*format*	Set the type of this list element to the desired *format*. For within : A (capital letters), a (lowercase letters), I (capital Roman numerals), i (lowercase Roman numerals), or 1 (Arabic numerals; default). For within : circle, disc (default), or square.	
	value=*n*	Set the number for this list item to *n*	
<link>		Define a link between this document and another document in the document <head>	
	charset=*charset*	Specify the character set used to encode the target of this link	
	href=*url*	Specify the hypertext reference URL of the target document	
	hreflang=*language*	Specify the language used for the target's contents using a standard two-character ISO language name	
	media=*list*	Specify a list of media types upon which this object can be rendered	

<link> (cont...)	rel=*relation*	Indicate the relationship from this document to the target	
	rev=*relation*	Indicate the reverse relationship from the target to this document	
	type=*string*	Specify the MIME type for the linked document. Usually used in conjunction with links to stylesheets, when the type is set to text/css	
<listing> ... </listing>		Same as <pre width=132> ... </pre>; deprecated, do not use	
<map> ... </map>		Define a map containing hotspots in a client-side image map	
	name=*name*	Define the name of this map (required)	
<marquee> ... </marquee>	Create a scrolling-text marquee (Internet Explorer only)	IE*	
	align=*position*	Align the marquee to the top, middle, or bottom of the surrounding text	IE
	behavior=*style*	Define marquee style to be scroll, slide, or alternate	IE
	bgcolor=*color*	Set the background color of the marquee	IE
	class=*name*	Specify a style class controlling the appearance of this tag	IE
	direction=*dir*	Define the direction, left or right, the text is to scroll	IE
	height=*n*	Define the height, in pixels, of the marquee area	IE
	hspace=*n*	Define the space, in pixels, to be inserted left and right of the marquee	IE
	loop=*value*	Set the number of times to animate the marquee; *value* is an integer or infinite	IE
	scrollamount=*value*	Set the number of pixels to move the text for each scroll movement	IE
	scrolldelay=*value*	Specify the delay, in milliseconds, between successive movements of the marquee text	IE
	style=*style*	Specify an inline style for this tag	IE
	vspace=*n*	Define the space, in pixels, to be inserted above and below of the marquee	IE

<marquee> ... </marquee> (cont...)	width=*n*	Define the width, in pixels, of the marquee area	O
<menu> ... </menu>		Define a menu list containing tags	
	compact	Make the list more compact if possible	
	type=*bullet*	Set the bullet style for this list to either circle, disc (default), or square	N O
<meta>		Provides additional information about a document	*
	charset=*name*	Specify the character set to be used with this document	O
	content=*string*	Specify the value for the meta-information (required)	
	dir=*dir*	Specify the rendering direction for text, either left-to-right (ltr) or right-to-left (rtl)	
	http-equiv=*string*	Specify the HTTP equivalent name for the meta-information and cause the server to include the name and content in the HTTP header for this document when it is transmitted to the client	
	lang=*language*	Specify the language used for this tag's contents using a standard two-character ISO language name	
	name=*string*	Specify the name of the meta-information	
	scheme=*scheme*	Specify the profile scheme used to interpret this property	
<multicol> ... </multicol>		Define a multi-column text flow	N *
	class=*name*	Specify a style class controlling the appearance of this tag	N
	cols=*n*	Specify the number of columns	N
	gutter=*n*	Define the spacing, in pixels, between columns	N
	style=*style*	Specify an inline style for this tag	N
	width=*n*	Define the width of the entire column group	N
<nobr> ... </nobr>		No breaks allowed in the enclosed text	N O *

<noembed> ... </noembed>		Define content to be presented by browsers that do not support the <embed> tag	N O *
<noframes> ... </noframes>		Define content to be presented by browsers that do not support frames	
<noscript> ... </noscript>		Define content to be presented by browsers that do not support the <script> tag	
<object>		Insert an object into a document	
	align=*position*	Align the object with the surrounding text (texttop, middle, textmiddle, baseline, textbottom, and center) or against the margin with subsequent text flowing around the object (left and right)	
	archive=*list*	Specify a list of URLs of archives containing resources used by this object	
	border=*n*	Define, in pixels, the object's border width	
	classid=*url*	Supply the URL of the object	
	codebase=*url*	Supply the URL of the object's code base	
	codetype=*type*	Specify the MIME type of the code base	
	data=*url*	Supply data for the object	
	declare	Declare this object without instantiating it	
	height=*n*	Define, in pixels, the height of the object	
	hspace=*n*	Provide extra space, in pixels, to the right and left of the object	
	name=*name*	Define the name of this object	
	notab	Do not make this object part of the tabbing order	O
	shapes	Specify that this object has shaped hyperlinks	O
	standby=*string*	Define a message to display while the object loads	
	tabindex=*n*	Specify this object's position in the document tab order	
	type=*type*	Specify the MIME type for the object data	
	usemap=*url*	Define an image map for use with this object	

<object> (cont...)	vspace=*n*	Provide extra space, in pixels, above and below of the object	
	width=*n*	Define, in pixels, the width of the object	
...		Define an ordered list containing numbered (ascending) elements	
	compact	Present the list in a more compact manner	
	start=*n*	Start numbering the list at *n*, instead of 1	
	type=*format*	Set the numbering *format* for this list to either A (capital letters), a (lowercase letters), I (capital Roman numerals), i (lowercase Roman numerals), or 1 (Arabic numerals; default)	
<optgroup> ... </optgroup>		Define a group of options within a <select> element	
	disabled	Disable this group, making it inactive	
	label=*string*	Provide a label for this group	
<option> ... </option>		Define an option within a <select> item in a <form>	
	disabled	Disable this option, making it inactive	
	label=*string*	Provide a label for this option	
	selected	Make this item initially selected	
	value=*string*	Return the specified value to the form-processing application instead of the <option> contents	
...		Start and end a paragraph	
	align=*type*	Align the text within the paragraph to left, center, or right	
<param> ... </param>		Supply a parameter to a containing <applet>	*
	id=*name*	Define the unique identifier for this parameter	
	name=*name*	Define the name of the parameter	
	type=*type*	Specify the MIME type of the parameter	
	value=*string*	Define the value of the parameter	

<param> ... </param> (cont...)	valuetype=*type*	Define the type of the value attribute, either as data, ref (the value is a URL pointing to the data), or object (the value is the name of an object in this document)	
<plaintext>		Render the remainder of the document as preformatted plain text	
<pre> ... </pre>		Render the enclosed text in its original, preformatted style, honoring line breaks and spacing verbatim	
	width=*n*	Size the text, if possible, so that *n* characters fit across the display window	
<q> ... </q>		The enclosed text is an inline quotation	
	cite=*url*	Specify the URL of source of the quoted material	
<s> ... </s>		Same as <strike>. The enclosed text is struck through with a horizontal line	
<samp> ... </samp>		The enclosed text is a sample	
<script> ... </script>		Define a script within a document	*
	charset=*encoding*	Specify the character set used to encode the script	
	defer	Defer execution of this script	
	language=*encoding*	Specify the language used to create the script	
	src=*url*	Provide the URL of the document containing the scripts	
	type=*encoding*	Specify the MIME type of the script	
<select> ... </select>		Define a multiple-choice menu or scrolling list within a <form>, containing one or more <option> tags	
	disabled	Disable this control, making it inactive	
	multiple	Allow user to select more than one <option> within the <select>	
	name=*name*	Define the name for the selected <option> values that, if selected, are passed to the form-processing application (required)	
	onblur=*applet*	Specify an applet to be run when the mouse leaves this element	
	onchange=*applet*	Specify an applet to be run when the user changes the value of this element	

<select> ... </select> (cont...)	onfocus=*applet*	Specify an applet to be run when the mouse enters this element	
	size=*n*	Display *n* items using a pulldown menu for size=1 (without multiple specified) and a scrolling list of *n* items otherwise	
	tabindex=*n*	Specify this element's position in the tabbing order	
<server> ... </server>		Define a LiveWire script	N
...		Format the enclosed text using a smaller typeface	
<spacer>		Create blank space in a document	I *
	align=*position*	Align a block spacer with either the surrounding text (top, texttop, middle, absmiddle, baseline, bottom, absbottom) or against a margin with subsequent text flowing around the spacer (left and right)	N
	height=*n*	Define the height, in pixels, of a block spacer	N
	size=*n*	Define the length, in pixels, of a horizontal or vertical spacer	N
	type=*type*	Set spacer type to one of block, horizontal, or vertical	N
	width=*value*	Define the width, in pixels, of a block spacer	N
...		Define a span of text for style application	
<strike> ... </strike>		The enclosed text is struck through with a horizontal line	
...		Strongly emphasize the enclosed text	
<style> ... </style>		Define one or more document level styles	*
	dir=*dir*	Specify the rendering direction for the title text, either left-to-right (ltr) or right-to-left (rtl)	
	lang=*language*	Specify the language used for this tag's title using a standard two-character ISO language name	
	media=*list*	Specify a list of media types upon which this object can be rendered	
	title=*string*	Specify a title for this tag	

<style> ... </style> (cont...)	type=*type*	Define the format of the styles (always text/css)	
_{...}		Format the enclosed text as subscript	
^{...}		Format the enclosed text as superscript	
<table> ... </table>		Define a table	
	align=*position*	Align the table in the center and flow the subsequent text around the table	
	background=*url*	Define a background image for the table	O
	bgcolor=*color*	Define the background color for the entire table	
	border=*n*	Create a border *n* pixels wide	
	bordercolor=*color*	Define the border color for the entire table	O
	bordercolordark=*color*	Define the dark border-highlighting color for the entire table	O
	bordercolorlight=*color*	Define the light border-highlighting color for the entire table	O N
	cellpadding=*n*	Place *n* pixels of padding around each cell's contents	
	cellspacing=*n*	Place *n* pixels of spacing between cells	
	cols=*n*	Specify the number of columns in this table	N O
	frame=*type*	Define where table borders are displayed, either border (default), void, above, below, hsides, lhs, rhs, vsides, or box	
	height=*n*	Define the height of the table in pixels	N O
	hspace=*n*	Specify the horizontal space, in pixels, added at the left and right of the table	N
	nowrap	Suppress text wrapping in table cells	O
	rules=*edges*	Determine where inner dividers are drawn, either all (default), groups (only around row and column groups), rows, cols, or none	
	summary=*string*	Provide a summary description of this table	

<table> ... </table> (cont...)	valign=*position*	Align text in the table to either the top, center, bottom, or baseline	I
	vspace=*n*	Specify the vertical space, in pixels, added at the top and bottom of the table	N
	width=*n*	Set the width of the table to *n* pixels or a percentage of the window width	
<tbody> ... </tbody>		Create a row group within a table	
	align=*position*	Align the table body cell contents to the left, center, or right	
	char=*char*	Specify the body group cell alignment character	
	charoff=*value*	Specify the offset within the cells of the alignment position	
	valign=*position*	Vertically align the body group cells' contents to the top, center, bottom, or baseline of the cell	
<td> ... </td>		Define a table data cell	
	abbr=*string*	Specify an abbreviation for the cell's contents	
	align=*position*	Align the cell contents to the left, center, or right	
	axis=*string*	Provide a name for a related group of cells	
	background=*url*	Define a background image for this cell	I
	bgcolor=*color*	Define the background color for the cell	
	bordercolor=*color*	Define the border color for the cell	I
	bordercolordark=*color*	Define the dark border highlighting color for the cell	I
	bordercolorlight=*color*	Define the light border highlighting color for the cell	I
	char=*char*	Specify the cell alignment character	
	charoff=*value*	Specify the offset of the alignment position within the cell	
	colspan=*n*	Have this cell straddle *n* adjacent columns	
	headers=*list*	Provide a list of header cell names associated with this cell	
	height=*n*	Define the height, in pixels, for this cell	

`<td>` ... `</td>` (cont...)	`nowrap`	Do not automatically wrap and fill text in this cell	
	`rowspan=`*n*	Have this cell straddle *n* adjacent rows	
	`scope=`*scope*	Define the scope of this header cell, either `row`, `col`, `rowgroup`, or `colgroup`	
	`valign=`*position*	Vertically align this cell's contents to the `top`, `center`, `bottom`, or `baseline` of the cell	
	`width=`*n*	Set the width of this cell to *n* pixels or a percentage of the table width	
`<textarea>` ... `</textarea>`		Define a multiline text input area within a `<form>`; content of the `<textarea>` tag is the initial, default value	
	`accesskey=`*char*	Define the hot-key character for this element	
	`cols=`*n*	Display *n* columns (characters) of text within the text area	
	`disabled`	Disable this control, making it inactive	
	`name=`*string*	Define the name for the text-area value that is passed to the form-processing application (required)	
	`onblur=`*applet*	Specify an applet to be run when the mouse leaves this element	
	`onchange=`*applet*	Specify an applet to be run when the user changes the value of this element	
	`onfocus=`*applet*	Specify an applet to be run when the mouse enters this element	
	`onselect=`*applet*	Specify an applet to be run if the user clicks this element	
	`readonly`	Prevent user modification of this element	
	`rows=`*n*	Display *n* rows of text within the text area	
	`tabindex=`*n*	Specify this element's position in the tabbing order	
	`wrap=`*style*	Set word wrapping within the text area to `off`, `virtual` (display wrap, but do not transmit to server), or `physical` (display and transmit wrap)	N
`<tfoot>` ... `</tfoot>`		Define a table footer	
	`align=`*position*	Align the footer cell contents to the `left`, `center`, or `right`	

<tfoot> ... </tfoot> (cont...)	char=*char*	Specify the cell alignment character	
	charoff=*value*	Specify the offset within the cell of the alignment position	
	valign=*position*	Vertically align the footer cells' contents to the top, center, bottom, or baseline of the cell	
<th> ... </th>		Define a table header cell	
	abbr=*string*	Specify an abbreviation for the cell's contents	
	align=*position*	Align the cell contents to the left, center, or right	
	axis=*string*	Provide a name for a related group of cells	
	background=*url*	Define a background image for this cell	
	bgcolor=*color*	Define the background color for the cell	
	bordercolor=*color*	Define the border color for the cell	
	bordercolordark=*color*	Define the dark border highlighting color for the cell	
	bordercolorlight=*color*	Define the light border highlighting color for the cell	
	char=*char*	Specify the cell alignment character	
	charoff=*value*	Specify the offset of the alignment position within the cell	
	colspan=*n*	Have this cell straddle *n* adjacent columns	
	headers=*list*	Provide a list of header cell names associated with this cell	
	height=*n*	Define the height, in pixels, for this cell	
	nowrap	Do not automatically wrap and fill text in this cell	
	rowspan=*n*	Have this cell straddle *n* adjacent rows	
	scope=*scope*	Define the scope of this header cell, either row, col, rowgroup, or colgroup	
	valign=*position*	Vertically align this cell's contents to the top, center, bottom, or baseline of the cell	
	width=*n*	Set the width of this cell to *n* pixels or a percentage of the table width	

<thead> ... </thead>		Define a table heading	
	align=*position*	Define the horizontal text alignment in the heading, either left, center, right, or justify	
	char=*char*	Specify the cell alignment character for heading cells	
	charoff=*value*	Specify the offset within the cells of the alignment position	
	valign=*position*	Define the vertical text alignment in the heading, either left, center, right, or justify	
<title> ... </title>		Define the HTML document's title	*
	dir=*dir*	Specify the rendering direction for text, either left-to-right (ltr) or right-to-left (rtl)	
	lang=*language*	Specify the language used for this tag's contents using a standard two-character ISO language name	
<tr> ... </tr>		Define a row of cells within a table	
	align=*type*	Align the cell contents in this row to the left, center, or right	
	background=*url*	Define a background image for this cell	N
	bgcolor=*color*	Define the background color for this row	N O
	bordercolor=*color*	For Internet Explorer, define the border color for this row	O
	bordercolordark=*color*	For Internet Explorer, define the dark border-highlighting color for this row	O
	bordercolorlight=*color*	For Internet Explorer, define the light border-highlighting color for this row	O
	char=*char*	Specify the cell alignment character for this row	
	charoff=*value*	Specify the offset of the alignment position within the cells of this row	
	nowrap	Disable word wrap for all cells in this row	N O
	valign=*position*	Vertically align the cell contents in this row to the top, center, bottom, or baseline of the cell	

`<tt>` ... `</tt>`		Format the enclosed text in teletype-style (monospaced) font	
`<u>` ... `</u>`		The enclosed text is underlined	
`<ul>` ... `</ul>`		Define an unordered list of bulleted `<li>` elements	
	`compact`	Display the list in a more compact manner, if possible	
	`type=`*bullet*	Set the bullet style for this list to either `circle`, `disc` (default), or `square`	
`<var>` ... `</var>`		The enclosed text is a variable's name	
`<wbr>`		Indicate a potential word break point within a `<nobr>` section	N O *
`<xmp>` ... `</xmp>`		Same as `<pre width=80>` ... `</pre>`; deprecated, do not use	

Appendix C. Cascading Style Sheet Properties Quick Reference

In the following table, we list in alphabetical order all the properties defined in the World Wide Web Consortium's Recommended Specification for Cascading Style Sheets, Level 1 (http://www.w3.org/pub/WWW/TR/REC-CSS1).

We know that browser support of style sheets will change faster than we can reprint this book, so we have created a separate "compliance document" that you can use to determine how style sheets are implemented by the latest releases of the browsers. You can find this document at http://www.oreilly.com/catalog/html4/. Whenever that document and this appendix differ, the document should be considered more accurate. In this printing, we cover Netscape Navigator 4.7 and Internet Explorer 5.0.

As in other sections of this book, we use the Netscape and Internet Explorer icons to the far right of each property to show which browser supports that property. Properties with no icons are not currently supported by any browser. We also include the number of the section in this book that fully defines the property.

We include each property's possible values, defined as either an explicit keyword (shown in `constant width`) or as one of these values:

color

Either a color name or hexadecimal RGB value, as defined in Appendix G, or an RGB triple of the form:

```
rgb(red, green, blue)
```

where *red*, *green*, and *blue* are either numbers in the range to 255 or percentage values indicating the brightness of that color component. Values of `255` or `100%` indicate that the corresponding color component is at its brightest; values of `0` or `0%` indicate that the corresponding color component is turned completely off. For example:

```
rgb(27, 119, 207)
rgb(50%, 75%, 0%)
```

are both valid color specifications.

length

An optional sign (either + or -) immediately followed by a number (with or without a decimal point) immediately followed by a two-character unit identifier. For values of zero, the unit identifier may be omitted.

The unit identifiers `em` and `ex` refer to the overall height of the font and to the height of the letter "x", respectively. The unit identifier `px` is equal to a single pixel on the display device. The unit identifiers `in`, `cm`, `mm`, `pt`, and `pc` refer to inches, centimeters, millimeters, points, and picas, respectively. There are 72.27 points per inch, and 12 points in a pica.

number

An optional sign, immediately followed by a number (with or without a decimal point).

percent

An optional sign, immediately followed by a number (with or without a decimal point), immediately followed by a percent sign. The actual value is computed as a percentage of some other element property, usually the element's size.

url

The keyword `url`, immediately followed (no spaces) by a left parenthesis, followed by a URL optionally enclosed in single or double quotes, followed by a matching right parenthesis. For example:

```
url("http://members.aol.com/htmlguru")
```

is a valid URL value.

Finally, some values are lists of other values and are described as a "list of" some other value. In these cases, a list consists of one or more of the allowed values, separated by commas.

If there are several different values allowed for a property, these alternative choices are separated by vertical bars (|).

If the standard defines a default value for the property, that value is **boldfaced**.

background		Composite property for the `background-attachment`,`background-color`, `background-image`, `background-position`, and `background-repeat` properties; value is any of these properties' values, in any order	Section 8.4.4.6	N I
background-attachment	`scroll` | `fixed`	Determines if the background image is fixed in the window or scrolls as the document scrolls	Section 8.4.4.1	I
background-color	*color* | **transparent**	Sets the background color of an element	Section 8.4.4.2	N I
background-image	*url* | **none**	Sets the background image of an element	Section 8.4.4.3	N I
background-position	*percent* | *length* | `top` | `center` | `bottom` | `left` | `right`	Sets the initial position of the element's background image, if specified; values are normally paired to provide x, y positions. Default position is `0% 0%`	Section 8.4.4.4	I
background-repeat	**repeat** | `repeat-x` | `repeat-y` | `no-repeat`	Determines how the background image is repeated (tiled) across an element	Section 8.4.4.5	N I
border		Sets all four borders on an element; value is one or more of a *color*, a value for `border-width`, and a value for `border-style`	Section 8.4.6.6	N I
border-bottom		Sets the bottom border on an element; value is one or more of a *color*, a value for `border-bottom-width`, and a value for `border-style`	Section 8.4.6.6	I
border-bottom-width	*length* | `thin` | **medium** | `thick`	Sets the thickness of an element's bottom border	Section 8.4.6.4	N I
border-color	*color*	Sets the color of all four of an element's borders; default is the color of the element	Section 8.4.6.3	N
border-left		Sets the left border on an element; value is one or more of a *color*, a value for `border-left-width`, and a value for `border-style`	Section 8.4.6.6	I
border-left-width	*length* | `thin` | **medium** | `thick`	Sets the thickness of an element's left border	Section 8.4.6.4	N I
border-right		Sets the right border on an element; value is one or more of a *color*, a value for `border-right-width`, and a value for `border-style`	Section 8.4.6.6	I
border-right-width	*length* | `thin` | **medium** | `thick`	Sets the thickness of an element's right border	Section 8.4.6.4	N I

border-style	dashed | dotted | double | groove | inset | **none** | outset | ridge | solid	Sets the style of all four of an element's borders	Section 8.4.6.5	N O
border-top		Sets the top border on an element; value is one or more of a *color*, a value for border-top-width, and a value for border-style	Section 8.4.6.6	O
border-top-width	*length* | thin | **medium** | thick	Sets the thickness of an element's top border	Section 8.4.6.4	N O
border-width	*length* | thin | **medium** | thick	Sets the thickness of all four of an element's borders	Section 8.4.6.4	N O
clear	both | left | **none** | right	Sets which margins of an element must not be adjacent to a floating element; the element will be moved down until that margin is clear	Section 8.4.6.7	N O
color	*color*	Sets the color of an element	Section 8.4.4.7	N O
display	**block** | inline | list-item | none	Controls how an element is displayed	Section 8.4.8.1	N O
float	left | **none** | right	Determines if an element will float to the left or right, allowing text to wrap around it, or be displayed inline (using none)	Section 8.4.6.8	N O
font		Sets all the font attributes for an element; value is any of the values for font-style, font-variant, font-weight, font-size, line-height, and font-family, in that order	Section 8.4.3.8	N O
font-family	list of font names	Defines the font for an element, either as a specific font or as one of the generic fonts serif, sans-serif, cursive, fantasy, and monospace	Section 8.4.3.1	N O
font-size	xx-small | x-small | small | **medium** | large | x-large | xx-large | larger | smaller | *length* | *percent*	Defines the font size	Section 8.4.3.2	N O
font-size-adjust	none | *ratio*	Adjusts the current font aspect ratio	Section 8.4.3.4	
font-stretch	wider | normal | narrower | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded	Determines the amount to stretch the current font	Section 8.4.3.3	

font-style	**normal** | `italic` | `oblique`	Defines the style of the face, either normal or some type of slanted style	Section 8.4.3.5	N O
font-variant	**normal** | `small-caps`	Defines a font to be in small caps	Section 8.4.3.6	O
font-weight	**normal** | `bold` | `bolder` | `lighter` | *number*	Defines the font weight. If a *number* is used, it must be a multiple of 100 between 100 and 900; 400 is normal, 700 is the same as the keyword `bold`.	Section 8.4.3.7	N O
height	*length* | **auto**	Defines the height of an element	Section 8.4.6.9	N O
letter-spacing	*length* | **normal**	Inserts additional space between text characters	Section 8.4.5.1	N O
line-height	*length* | *number* | *percent* | **normal**	Sets the distance between adjacent text baselines	Section 8.4.5.2	N O
list-style		Defines list-related styles using any of the values for `list-style-image`, `list-style-position`, and `list-style-type`	Section 8.4.7.4	N O
list-style-image	*url* | **none**	Defines an image to be used as a list item's marker, in lieu of the value for `list-style-type`	Section 8.4.7.1	N O
list-style-position	`inside` | **outside**	Indents or extends (default) a list item's marker with respect to the item's content	Section 8.4.7.2	
list-style-type	`circle` | **disc** | `square` | `decimal` | `lower-alpha` | `lower-roman` | `none` | `upper-alpha` | `upper-roman`	Defines a list item's marker for either unordered lists (`circle`, `disc`, or `square`) or for ordered lists (`decimal`, `lower-alpha`, `lower-roman`, `none`, `upper-alpha`, or `upper-roman`)	Section 8.4.7.3	N O
margin	*length* | *percent* | `auto`	Defines all four of an element's margins	Section 8.4.6.10	N O
margin-bottom	*length* | *percent* | `auto`	Defines the bottom margin of an element; default value is 0	Section 8.4.6.10	N O
margin-left	*length* | *percent* | `auto`	Defines the left margin of an element; default value is 0	Section 8.4.6.10	N O
margin-right	*length* | *percent* | `auto`	Defines the right margin of an element; default value is 0	Section 8.4.6.10	N O
margin-top	*length* | *percent* | `auto`	Defines the top margin of an element; default value is 0	Section 8.4.6.10	N O
padding		Defines all four padding amounts around an element	Section 8.4.6.11	N O

padding-bottom	*length* | *percent*	Defines the bottom padding of an element; default value is 0	Section 8.4.6.11	
padding-left	*length* | *percent*	Defines the left padding of an element; default value is 0	Section 8.4.6.11	
padding-right	*length* | *percent*	Defines the right padding of an element; default value is 0	Section 8.4.6.11	
padding-top	*length* | *percent*	Defines the top padding of an element; default value is 0	Section 8.4.6.11	
text-align	`center` | `justify` | `left` | `right`	Set the text alignment style for an element	Section 8.4.5.3	
text-decoration	`blink` | `line-through` | **none** | `overline` | `underline`	Defines any decoration for the text; values may be combined	Section 8.4.5.4	
text-indent	*length* | *percent*	Defines the indentation of the first line of text in an element; default value is	Section 8.4.5.5	
text-shadow	`See text`	Creates text drop shadows of varying colors and offsets	Section 8.4.5.6	
text-transform	`capitalize` | `lowercase` | **none** | `uppercase`	Transforms the text in the element accordingly	Section 8.4.5.7	
vertical-align	*percent* | `baseline` | `bottom` | `middle` | `sub` | `super` | `text-bottom` | `text-top` | `top`	Sets the vertical positioning of an element	Section 8.4.5.8	
word-spacing	*length* | **normal**	Inserts additional space between words	Section 8.4.5.9	
white-space	**normal** | `nowrap` | `pre`	Defines how whitespace within an element is handled	Section 8.4.8.2	
width	*length* | *percent* | **auto**	Defines the width of an element	Section 8.4.6.12	

Appendix D. The HTML 4.01 DTD

The HTML 4.01 standard is formally defined as three SGML Document Type Definitions (DTDs): the Strict DTD, the Transitional DTD, and the Frameset DTD. The Strict DTD defines only those elements that are not deprecated in the 4.0 standard. Ideally, everyone would create HTML documents that conform to the Strict DTD. The Transitional DTD includes all those deprecated elements and more accurately reflects the HTML in use today, with many older elements still in common use. The Frameset DTD is identical to the Transitional DTD, with the exception that the document <body> is replaced by the <frameset> tag.

Since the Transitional DTD provides the broadest coverage of all HTML elements currently in use, it is the DTD upon which this book is based and the one we reproduce here. Note that we have reprinted this DTD verbatim and have not attempted to add extensions to it. Where our description and the DTD deviate, assume the DTD is correct:

```
<!--
    This is the HTML 4.01 Transitional DTD, which includes
    presentation attributes and elements that W3C expects to phase out
    as support for style sheets matures. Authors should use the Strict
    DTD when possible, but may use the Transitional DTD when support
    for presentation attribute and elements is required.
    HTML 4 includes mechanisms for style sheets, scripting,
    embedding objects, improved support for right to left and mixed
    direction text, and enhancements to forms for improved
    accessibility for people with disabilities.
          Draft: $Date: 2001/08/22 23:32:17 $
          Authors:
              Dave Raggett <dsr@w3.org>
              Arnaud Le Hors <lehors@w3.org>
              Ian Jacobs <ij@w3.org>
    Further information about HTML 4.01 is available at:
        http://www.w3.org/TR/1999/REC-html401-19991224
    The HTML 4.01 specification includes additional
    syntactic constraints that cannot be expressed within
    the DTDs.
-->
<!ENTITY % HTML.Version "-//W3C//DTD HTML 4.01 Transitional//EN"
  -- Typical usage:
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
            "http://www.w3.org/TR/html4/loose.dtd">
    <html>
    <head>
    ...
    </head>
    <body>
    ...
    </body>
    </html>
    The URI used as a system identifier with the public identifier allows
    the user agent to download the DTD and entity sets as needed.
    The FPI for the Strict HTML 4.01 DTD is:
        "-//W3C//DTD HTML 4.01//EN"
    This version of the strict DTD is:
        http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd
    Authors should use the Strict DTD unless they need the
    presentation control for user agents that don't (adequately)
    support style sheets.
    If you are writing a document that includes frames, use
    the following FPI:
        "-//W3C//DTD HTML 4.01 Frameset//EN"
    This version of the frameset DTD is:
        http://www.w3.org/TR/1999/REC-html401-19991224/frameset.dtd
    Use the following (relative) URIs to refer to
    the DTDs and entity definitions of this specification:
    "strict.dtd"
    "loose.dtd"
    "frameset.dtd"
    "HTMLlat1.ent"
    "HTMLsymbol.ent"
    "HTMLspecial.ent"
-->
<!--================== Imported Names ====================================-->
<!-- Feature Switch for frameset documents -->
<!ENTITY % HTML.Frameset "IGNORE">
<!ENTITY % ContentType "CDATA"
    -- media type, as per [RFC2045]
    -->
<!ENTITY % ContentTypes "CDATA"
    -- comma-separated list of media types, as per [RFC2045]
    -->
<!ENTITY % Charset "CDATA"
    -- a character encoding, as per [RFC2045]
    -->
```

```
<!ENTITY % Charsets "CDATA"
    -- a space-separated list of character encodings, as per [RFC2045]
    -->
<!ENTITY % LanguageCode "NAME"
    -- a language code, as per [RFC1766]
    -->
<!ENTITY % Character "CDATA"
    -- a single character from [ISO10646]
    -->
<!ENTITY % LinkTypes "CDATA"
    -- space-separated list of link types
    -->
<!ENTITY % MediaDesc "CDATA"
    -- single or comma-separated list of media descriptors
    -->
<!ENTITY % URI "CDATA"
    -- a Uniform Resource Identifier,
       see [URI]
    -->
<!ENTITY % Datetime "CDATA" -- date and time information. ISO date format -->
<!ENTITY % Script "CDATA" -- script expression -->
<!ENTITY % StyleSheet "CDATA" -- style sheet data -->
<!ENTITY % FrameTarget "CDATA" -- render in this frame -->
<!ENTITY % Text "CDATA">
<!-- Parameter Entities -->
<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK|OBJECT" -- repeatable head elements -->
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % list "UL | OL |  DIR | MENU">
<!ENTITY % preformatted "PRE">
<!ENTITY % Color "CDATA" -- a color using sRGB: #RRGGBB as Hex values -->
<!-- There are also 16 widely known color names with their sRGB values:
    Black  = #000000    Green  = #008000
    Silver = #C0C0C0    Lime   = #00FF00
    Gray   = #808080    Olive  = #808000
    White  = #FFFFFF    Yellow = #FFFF00
    Maroon = #800000    Navy   = #000080
    Red    = #FF0000    Blue   = #0000FF
    Purple = #800080    Teal   = #008080
    Fuchsia= #FF00FF    Aqua   = #00FFFF
 -->
<!ENTITY % bodycolors "
  bgcolor     %Color;        #IMPLIED  -- document background color --
  text        %Color;        #IMPLIED  -- document text color --
  link        %Color;        #IMPLIED  -- color of links --
  vlink       %Color;        #IMPLIED  -- color of visited links --
  alink       %Color;        #IMPLIED  -- color of selected links --
  ">
<!--================ Character mnemonic entities =========================-->
<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin1//EN//HTML"
   "HTMLlat1.ent">
%HTMLlat1;
<!ENTITY % HTMLsymbol PUBLIC
   "-//W3C//ENTITIES Symbols//EN//HTML"
   "HTMLsymbol.ent">
%HTMLsymbol;
<!ENTITY % HTMLspecial PUBLIC
   "-//W3C//ENTITIES Special//EN//HTML"
   "HTMLspecial.ent">
%HTMLspecial;
<!--================== Generic Attributes ===============================-->
<!ENTITY % coreattrs
 "id          ID             #IMPLIED  -- document-wide unique id --
  class       CDATA          #IMPLIED  -- space-separated list of classes --
  style       %StyleSheet;   #IMPLIED  -- associated style info --
  title       %Text;         #IMPLIED  -- advisory title --"
  >
<!ENTITY % i18n
 "lang        %LanguageCode; #IMPLIED  -- language code --
  dir         (ltr|rtl)      #IMPLIED  -- direction for weak/neutral text --"
  >
<!ENTITY % events
 "onclick     %Script;       #IMPLIED  -- a pointer button was clicked --
  ondblclick  %Script;       #IMPLIED  -- a pointer button was double clicked--
  onmousedown %Script;       #IMPLIED  -- a pointer button was pressed down --
  onmouseup   %Script;       #IMPLIED  -- a pointer button was released --
  onmouseover %Script;       #IMPLIED  -- a pointer was moved onto --
  onmousemove %Script;       #IMPLIED  -- a pointer was moved within --
  onmouseout  %Script;       #IMPLIED  -- a pointer was moved away --
  onkeypress  %Script;       #IMPLIED  -- a key was pressed and released --
  onkeydown   %Script;       #IMPLIED  -- a key was pressed down --
  onkeyup     %Script;       #IMPLIED  -- a key was released --"
  >
```

```
<!-- Reserved Feature Switch -->
<!ENTITY % HTML.Reserved "IGNORE">
<!-- The following attributes are reserved for possible future use -->
<![ %HTML.Reserved; [
<!ENTITY % reserved
 "datasrc     %URI;          #IMPLIED  -- a single or tabular Data Source --
  datafld     CDATA          #IMPLIED  -- the property or column name --
  dataformatas (plaintext|html) plaintext -- text or html --"
  >
]]>
<!ENTITY % reserved "">
<!ENTITY % attrs "%coreattrs; %i18n; %events;">
<!ENTITY % align "align (left|center|right|justify)  #IMPLIED"
                 -- default is left for ltr paragraphs, right for rtl --
  >
<!--================== Text Markup ===========================================-->
<!ENTITY % fontstyle
 "TT | I | B | U | S | STRIKE | BIG | SMALL">
<!ENTITY % phrase "EM | STRONG | DFN | CODE |
                   SAMP | KBD | VAR | CITE | ABBR | ACRONYM" >
<!ENTITY % special
   "A | IMG | APPLET | OBJECT | FONT | BASEFONT | BR | SCRIPT |
    MAP | Q | SUB | SUP | SPAN | BDO | IFRAME">
<!ENTITY % formctrl "INPUT | SELECT | TEXTAREA | LABEL | BUTTON">
<!-- %inline; covers inline or "text-level" elements -->
<!ENTITY % inline "#PCDATA | %fontstyle; | %phrase; | %special; | %formctrl;">
<!ELEMENT (%fontstyle;|%phrase;) - - (%inline;)*>
<!ATTLIST (%fontstyle;|%phrase;)
  %attrs;                              -- %coreattrs, %i18n, %events --
  >
<!ELEMENT (SUB|SUP) - - (%inline;)*    -- subscript, superscript -->
<!ATTLIST (SUB|SUP)
  %attrs;                              -- %coreattrs, %i18n, %events --
  >
<!ELEMENT SPAN - - (%inline;)*         -- generic language/style container -->
<!ATTLIST SPAN
  %attrs;                              -- %coreattrs, %i18n, %events --
  %reserved;                           -- reserved for possible future use --
  >
<!ELEMENT BDO - - (%inline;)*          -- I18N BiDi over-ride -->
<!ATTLIST BDO
  %coreattrs;                          -- id, class, style, title --
  lang        %LanguageCode; #IMPLIED  -- language code --
  dir         (ltr|rtl)      #REQUIRED -- directionality --
  >
<!ELEMENT BASEFONT - O EMPTY           -- base font size -->
<!ATTLIST BASEFONT
  id          ID             #IMPLIED  -- document-wide unique id --
  size        CDATA          #REQUIRED -- base font size for FONT elements --
  color       %Color;        #IMPLIED  -- text color --
  face        CDATA          #IMPLIED  -- comma-separated list of font names --
  >
<!ELEMENT FONT - - (%inline;)*         -- local change to font -->
<!ATTLIST FONT
  %coreattrs;                          -- id, class, style, title --
  %i18n;                               -- lang, dir --
  size        CDATA          #IMPLIED  -- [+|-]nn e.g. size="+1", size="4" --
  color       %Color;        #IMPLIED  -- text color --
  face        CDATA          #IMPLIED  -- comma-separated list of font names --
  >
<!ELEMENT BR - O EMPTY                 -- forced line break -->
<!ATTLIST BR
  %coreattrs;                          -- id, class, style, title --
  clear       (left|all|right|none) none -- control of text flow --
  >
<!--================== HTML content models ================================-->
<!--
    HTML has two basic content models:
        %inline;     character level elements and text strings
        %block;      block-like elements e.g. paragraphs and lists
-->
<!ENTITY % block
     "P | %heading; | %list; | %preformatted; | DL | DIV | CENTER |
      NOSCRIPT | NOFRAMES | BLOCKQUOTE | FORM | ISINDEX | HR |
      TABLE | FIELDSET | ADDRESS">
<!ENTITY % flow "%block; | %inline;">
<!--================== Document Body =====================================-->
<!ELEMENT BODY O O (%flow;)* +(INS|DEL) -- document body -->
<!ATTLIST BODY
  %attrs;                              -- %coreattrs, %i18n, %events --
  onload          %Script;   #IMPLIED  -- the document has been loaded --
  onunload        %Script;   #IMPLIED  -- the document has been removed --
  background      %URI;      #IMPLIED  -- texture tile for document
                                          background --
  %bodycolors;                         -- bgcolor, text, link, vlink, alink --
  >
```

```
<!ELEMENT ADDRESS - - ((%inline;)|P)*  -- information on author -->
<!ATTLIST ADDRESS
  %attrs;                              -- %coreattrs, %i18n, %events --
  >
<!ELEMENT DIV - - (%flow;)*            -- generic language/style container -->
<!ATTLIST DIV
  %attrs;                              -- %coreattrs, %i18n, %events --
  %align;                              -- align, text alignment --
  %reserved;                           -- reserved for possible future use --
  >
<!ELEMENT CENTER - - (%flow;)*         -- shorthand for DIV align=center -->
<!ATTLIST CENTER
  %attrs;                              -- %coreattrs, %i18n, %events --
  >
<!--================== The Anchor Element ================================-->
<!ENTITY % Shape "(rect|circle|poly|default)">
<!ENTITY % Coords "CDATA" -- comma-separated list of lengths -->
<!ELEMENT A - - (%inline;)* -(A)       -- anchor -->
<!ATTLIST A
  %attrs;                              -- %coreattrs, %i18n, %events --
  charset     %Charset;      #IMPLIED  -- char encoding of linked resource --
  type        %ContentType;  #IMPLIED  -- advisory content type --
  name        CDATA          #IMPLIED  -- named link end --
  href        %URI;          #IMPLIED  -- URI for linked resource --
  hreflang    %LanguageCode; #IMPLIED  -- language code --
  target      %FrameTarget;  #IMPLIED  -- render in this frame --
  rel         %LinkTypes;    #IMPLIED  -- forward link types --
  rev         %LinkTypes;    #IMPLIED  -- reverse link types --
  accesskey   %Character;    #IMPLIED  -- accessibility key character --
  shape       %Shape;        rect      -- for use with client-side image maps --
  coords      %Coords;       #IMPLIED  -- for use with client-side image maps --
  tabindex    NUMBER         #IMPLIED  -- position in tabbing order --
  onfocus     %Script;       #IMPLIED  -- the element got the focus --
  onblur      %Script;       #IMPLIED  -- the element lost the focus --
  >
<!--================== Client-side image maps ============================-->
<!-- These can be placed in the same document or grouped in a
     separate document although this isn't yet widely supported -->
<!ELEMENT MAP - - ((%block;) | AREA)+ -- client-side image map -->
<!ATTLIST MAP
  %attrs;                              -- %coreattrs, %i18n, %events --
  name        CDATA          #REQUIRED -- for reference by usemap --
  >
<!ELEMENT AREA - O EMPTY               -- client-side image map area -->
<!ATTLIST AREA
  %attrs;                              -- %coreattrs, %i18n, %events --
  shape       %Shape;        rect      -- controls interpretation of coords --
  coords      %Coords;       #IMPLIED  -- comma-separated list of lengths --
  href        %URI;          #IMPLIED  -- URI for linked resource --
  target      %FrameTarget;  #IMPLIED  -- render in this frame --
  nohref      (nohref)       #IMPLIED  -- this region has no action --
  alt         %Text;         #REQUIRED -- short description --
  tabindex    NUMBER         #IMPLIED  -- position in tabbing order --
  accesskey   %Character;    #IMPLIED  -- accessibility key character --
  onfocus     %Script;       #IMPLIED  -- the element got the focus --
  onblur      %Script;       #IMPLIED  -- the element lost the focus --
  >
<!--================== The LINK Element ==================================-->
<!--
  Relationship values can be used in principle:
   a) for document specific toolbars/menus when used
      with the LINK element in document head e.g.
        start, contents, previous, next, index, end, help
   b) to link to a separate style sheet (rel=stylesheet)
   c) to make a link to a script (rel=script)
   d) by stylesheets to control how collections of
      html nodes are rendered into printed documents
   e) to make a link to a printable version of this document
      e.g. a postscript or pdf version (rel=alternate media=print)
-->
<!ELEMENT LINK - O EMPTY               -- a media-independent link -->
<!ATTLIST LINK
  %attrs;                              -- %coreattrs, %i18n, %events --
  charset     %Charset;      #IMPLIED  -- char encoding of linked resource --
  href        %URI;          #IMPLIED  -- URI for linked resource --
  hreflang    %LanguageCode; #IMPLIED  -- language code --
  type        %ContentType;  #IMPLIED  -- advisory content type --
  rel         %LinkTypes;    #IMPLIED  -- forward link types --
  rev         %LinkTypes;    #IMPLIED  -- reverse link types --
  media       %MediaDesc;    #IMPLIED  -- for rendering on these media --
  target      %FrameTarget;  #IMPLIED  -- render in this frame --
  >
<!--================== Images ============================================-->
<!-- Length defined in strict DTD for cellpadding/cellspacing -->
<!ENTITY % Length "CDATA" -- nn for pixels or nn% for percentage length -->
<!ENTITY % MultiLength "CDATA" -- pixel, percentage, or relative -->
<![ %HTML.Frameset; [
<!ENTITY % MultiLengths "CDATA" -- comma-separated list of MultiLength -->
]]>
```

```
<!ENTITY % Pixels "CDATA" -- integer representing length in pixels -->
<!ENTITY % IAlign "(top|middle|bottom|left|right)" -- center? -->
<!-- To avoid problems with text-only UAs as well as
   to make image content understandable and navigable
   to users of non-visual UAs, you need to provide
   a description with ALT, and avoid server-side image maps -->
<!ELEMENT IMG - O EMPTY                -- Embedded image -->
<!ATTLIST IMG
  %attrs;                              -- %coreattrs, %i18n, %events --
  src         %URI;          #REQUIRED -- URI of image to embed --
  alt         %Text;         #REQUIRED -- short description --
  longdesc    %URI;          #IMPLIED  -- link to long description
                                          (complements alt) --
  name        CDATA          #IMPLIED  -- name of image for scripting --
  height      %Length;       #IMPLIED  -- override height --
  width       %Length;       #IMPLIED  -- override width --
  usemap      %URI;          #IMPLIED  -- use client-side image map --
  ismap       (ismap)        #IMPLIED  -- use server-side image map --
  align       %IAlign;       #IMPLIED  -- vertical or horizontal alignment --
  border      %Pixels;       #IMPLIED  -- link border width --
  hspace      %Pixels;       #IMPLIED  -- horizontal gutter --
  vspace      %Pixels;       #IMPLIED  -- vertical gutter --
  >
<!-- USEMAP points to a MAP element which may be in this document
  or an external document, although the latter is not widely supported -->
<!--================== OBJECT ====================================-->
<!--
  OBJECT is used to embed objects as part of HTML pages
  PARAM elements should precede other content. SGML mixed content
  model technicality precludes specifying this formally ...
-->
<!ELEMENT OBJECT - - (PARAM | %flow;)*
 -- generic embedded object -->
<!ATTLIST OBJECT
  %attrs;                              -- %coreattrs, %i18n, %events --
  declare     (declare)      #IMPLIED  -- declare but don't instantiate flag --
  classid     %URI;          #IMPLIED  -- identifies an implementation --
  codebase    %URI;          #IMPLIED  -- base URI for classid, data, archive--
  data        %URI;          #IMPLIED  -- reference to object's data --
  type        %ContentType;  #IMPLIED  -- content type for data --
  codetype    %ContentType;  #IMPLIED  -- content type for code --
  archive     CDATA          #IMPLIED  -- space-separated list of URIs --
  standby     %Text;         #IMPLIED  -- message to show while loading --
  height      %Length;       #IMPLIED  -- override height --
  width       %Length;       #IMPLIED  -- override width --
  usemap      %URI;          #IMPLIED  -- use client-side image map --
  name        CDATA          #IMPLIED  -- submit as part of form --
  tabindex    NUMBER         #IMPLIED  -- position in tabbing order --
  align       %IAlign;       #IMPLIED  -- vertical or horizontal alignment --
  border      %Pixels;       #IMPLIED  -- link border width --
  hspace      %Pixels;       #IMPLIED  -- horizontal gutter --
  vspace      %Pixels;       #IMPLIED  -- vertical gutter --
  %reserved;                           -- reserved for possible future use --
  >
<!ELEMENT PARAM - O EMPTY              -- named property value -->
<!ATTLIST PARAM
  id          ID             #IMPLIED  -- document-wide unique id --
  name        CDATA          #REQUIRED -- property name --
  value       CDATA          #IMPLIED  -- property value --
  valuetype   (DATA|REF|OBJECT) DATA   -- How to interpret value --
  type        %ContentType;  #IMPLIED  -- content type for value
                                          when valuetype=ref --
  >
<!--================== Java APPLET ===================================-->
<!--
  One of code or object attributes must be present.
  Place PARAM elements before other content.
-->
<!ELEMENT APPLET - - (PARAM | %flow;)* -- Java applet -->
<!ATTLIST APPLET
  %coreattrs;                          -- id, class, style, title --
  codebase    %URI;          #IMPLIED  -- optional base URI for applet --
  archive     CDATA          #IMPLIED  -- comma-separated archive list --
  code        CDATA          #IMPLIED  -- applet class file --
  object      CDATA          #IMPLIED  -- serialized applet file --
  alt         %Text;         #IMPLIED  -- short description --
  name        CDATA          #IMPLIED  -- allows applets to find each other --
  width       %Length;       #REQUIRED -- initial width --
  height      %Length;       #REQUIRED -- initial height --
  align       %IAlign;       #IMPLIED  -- vertical or horizontal alignment --
  hspace      %Pixels;       #IMPLIED  -- horizontal gutter --
  vspace      %Pixels;       #IMPLIED  -- vertical gutter --
  >
```

```
<!--==================== Horizontal Rule ================================-->
<!ELEMENT HR - O EMPTY -- horizontal rule -->
<!ATTLIST HR
  %attrs;                               -- %coreattrs, %i18n, %events --
  align       (left|center|right) #IMPLIED
  noshade     (noshade)       #IMPLIED
  size        %Pixels;        #IMPLIED
  width       %Length;        #IMPLIED
  >
<!--==================== Paragraphs ====================================-->
<!ELEMENT P - O (%inline;)*             -- paragraph -->
<!ATTLIST P
  %attrs;                               -- %coreattrs, %i18n, %events --
  %align;                               -- align, text alignment --
  >
<!--==================== Headings ======================================-->
<!--
  There are six levels of headings from H1 (the most important)
  to H6 (the least important).
-->
<!ELEMENT (%heading;)  - - (%inline;)* -- heading -->
<!ATTLIST (%heading;)
  %attrs;                               -- %coreattrs, %i18n, %events --
  %align;                               -- align, text alignment --
  >
<!--==================== Preformatted Text ==============================-->
<!-- excludes markup for images and changes in font size -->
<!ENTITY % pre.exclusion "IMG|OBJECT|APPLET|BIG|SMALL|SUB|SUP|FONT|BASEFONT">
<!ELEMENT PRE - - (%inline;)* -(%pre.exclusion;) -- preformatted text -->
<!ATTLIST PRE
  %attrs;                               -- %coreattrs, %i18n, %events --
  width       NUMBER          #IMPLIED
  >
<!--==================== Inline Quotes ==================================-->
<!ELEMENT Q - - (%inline;)*             -- short inline quotation -->
<!ATTLIST Q
  %attrs;                               -- %coreattrs, %i18n, %events --
  cite        %URI;           #IMPLIED  -- URI for source document or msg --
  >
<!--==================== Block-like Quotes ==============================-->
<!ELEMENT BLOCKQUOTE - - (%flow;)*      -- long quotation -->
<!ATTLIST BLOCKQUOTE
  %attrs;                               -- %coreattrs, %i18n, %events --
  cite        %URI;           #IMPLIED  -- URI for source document or msg --
  >
<!--==================== Inserted/Deleted Text ==========================-->
<!-- INS/DEL are handled by inclusion on BODY -->
<!ELEMENT (INS|DEL) - - (%flow;)*       -- inserted text, deleted text -->
<!ATTLIST (INS|DEL)
  %attrs;                               -- %coreattrs, %i18n, %events --
  cite        %URI;           #IMPLIED  -- info on reason for change --
  datetime    %Datetime;      #IMPLIED  -- date and time of change --
  >
<!--==================== Lists =========================================-->
<!-- definition lists - DT for term, DD for its definition -->
<!ELEMENT DL - - (DT|DD)+               -- definition list -->
<!ATTLIST DL
  %attrs;                               -- %coreattrs, %i18n, %events --
  compact     (compact)       #IMPLIED  -- reduced interitem spacing --
  >
<!ELEMENT DT - O (%inline;)*            -- definition term -->
<!ELEMENT DD - O (%flow;)*              -- definition description -->
<!ATTLIST (DT|DD)
  %attrs;                               -- %coreattrs, %i18n, %events --
  >
<!-- Ordered lists (OL) Numbering style
    1   arablic numbers     1, 2, 3, ...
    a   lower alpha         a, b, c, ...
    A   upper alpha         A, B, C, ...
    i   lower roman         i, ii, iii, ...
    I   upper roman         I, II, III, ...
    The style is applied to the sequence number which by default
    is reset to 1 for the first list item in an ordered list.
    This can't be expressed directly in SGML due to case folding.
-->
<!ENTITY % OLStyle "CDATA"      -- constrained to: "(1|a|A|i|I)" -->
<!ELEMENT OL - - (LI)+                  -- ordered list -->
<!ATTLIST OL
  %attrs;                               -- %coreattrs, %i18n, %events --
  type        %OLStyle;       #IMPLIED  -- numbering style --
  compact     (compact)       #IMPLIED  -- reduced interitem spacing --
  start       NUMBER          #IMPLIED  -- starting sequence number --
  >
<!-- Unordered Lists (UL) bullet styles -->
<!ENTITY % ULStyle "(disc|square|circle)">
<!ELEMENT UL - - (LI)+                  -- unordered list -->
```

```
<!ATTLIST UL
  %attrs;                              -- %coreattrs, %i18n, %events --
  type        %ULStyle;      #IMPLIED  -- bullet style --
  compact     (compact)      #IMPLIED  -- reduced interitem spacing --
  >
<!ELEMENT (DIR|MENU) - - (LI)+ -(%block;) -- directory list, menu list -->
<!ATTLIST DIR
  %attrs;                              -- %coreattrs, %i18n, %events --
  compact     (compact)      #IMPLIED -- reduced interitem spacing --
  >
<!ATTLIST MENU
  %attrs;                              -- %coreattrs, %i18n, %events --
  compact     (compact)      #IMPLIED -- reduced interitem spacing --
  >
<!ENTITY % LIStyle "CDATA" -- constrained to: "(%ULStyle;|%OLStyle;)" -->
<!ELEMENT LI - O (%flow;)*             -- list item -->
<!ATTLIST LI
  %attrs;                              -- %coreattrs, %i18n, %events --
  type        %LIStyle;      #IMPLIED  -- list item style --
  value       NUMBER         #IMPLIED  -- reset sequence number --
  >
<!--================ Forms ===============================================-->
<!ELEMENT FORM - - (%flow;)* -(FORM)   -- interactive form -->
<!ATTLIST FORM
  %attrs;                              -- %coreattrs, %i18n, %events --
  action      %URI;          #REQUIRED -- server-side form handler --
  method      (GET|POST)     GET       -- HTTP method used to submit the form--
  enctype     %ContentType;  "application/x-www-form-urlencoded"
  accept      %ContentTypes; #IMPLIED  -- list of MIME types for file upload --
  name        CDATA          #IMPLIED  -- name of form for scripting --
  onsubmit    %Script;       #IMPLIED  -- the form was submitted --
  onreset     %Script;       #IMPLIED  -- the form was reset --
  target      %FrameTarget;  #IMPLIED  -- render in this frame --
  accept-charset %Charsets;  #IMPLIED  -- list of supported charsets --
  >
<!-- Each label must not contain more than ONE field -->
<!ELEMENT LABEL - - (%inline;)* -(LABEL) -- form field label text -->
<!ATTLIST LABEL
  %attrs;                              -- %coreattrs, %i18n, %events --
  for         IDREF          #IMPLIED  -- matches field ID value --
  accesskey   %Character;    #IMPLIED  -- accessibility key character --
  onfocus     %Script;       #IMPLIED  -- the element got the focus --
  onblur      %Script;       #IMPLIED  -- the element lost the focus --
  >
<!ENTITY % InputType
  "(TEXT | PASSWORD | CHECKBOX |
    RADIO | SUBMIT | RESET |
    FILE | HIDDEN | IMAGE | BUTTON)"
   >
<!-- attribute name required for all but submit and reset -->
<!ELEMENT INPUT - O EMPTY              -- form control -->
<!ATTLIST INPUT
  %attrs;                              -- %coreattrs, %i18n, %events --
  type        %InputType;    TEXT      -- what kind of widget is needed --
  name        CDATA          #IMPLIED  -- submit as part of form --
  value       CDATA          #IMPLIED  -- Specify for radio buttons and checkboxes --
  checked     (checked)      #IMPLIED  -- for radio buttons and check boxes --
  disabled    (disabled)     #IMPLIED  -- unavailable in this context --
  readonly    (readonly)     #IMPLIED  -- for text and passwd --
  size        CDATA          #IMPLIED  -- specific to each type of field --
  maxlength   NUMBER         #IMPLIED  -- max chars for text fields --
  src         %URI;          #IMPLIED  -- for fields with images --
  alt         CDATA          #IMPLIED  -- short description --
  usemap      %URI;          #IMPLIED  -- use client-side image map --
  ismap       (ismap)        #IMPLIED  -- use server-side image map --
  tabindex    NUMBER         #IMPLIED  -- position in tabbing order --
  accesskey   %Character;    #IMPLIED  -- accessibility key character --
  onfocus     %Script;       #IMPLIED  -- the element got the focus --
  onblur      %Script;       #IMPLIED  -- the element lost the focus --
  onselect    %Script;       #IMPLIED  -- some text was selected --
  onchange    %Script;       #IMPLIED  -- the element value was changed --
  accept      %ContentTypes; #IMPLIED  -- list of MIME types for file upload --
  align       %IAlign;       #IMPLIED  -- vertical or horizontal alignment --
  %reserved;                           -- reserved for possible future use --
  >
<!ELEMENT SELECT - - (OPTGROUP|OPTION)+ -- option selector -->
<!ATTLIST SELECT
  %attrs;                              -- %coreattrs, %i18n, %events --
  name        CDATA          #IMPLIED  -- field name --
  size        NUMBER         #IMPLIED  -- rows visible --
  multiple    (multiple)     #IMPLIED  -- default is single selection --
  disabled    (disabled)     #IMPLIED  -- unavailable in this context --
  tabindex    NUMBER         #IMPLIED  -- position in tabbing order --
  onfocus     %Script;       #IMPLIED  -- the element got the focus --
  onblur      %Script;       #IMPLIED  -- the element lost the focus --
  onchange    %Script;       #IMPLIED  -- the element value was changed --
  %reserved;                           -- reserved for possible future use --
  >
```

```
<!ELEMENT OPTGROUP - - (OPTION)+ -- option group -->
<!ATTLIST OPTGROUP
  %attrs;                              -- %coreattrs, %i18n, %events --
  disabled    (disabled)     #IMPLIED  -- unavailable in this context --
  label       %Text;         #REQUIRED -- for use in hierarchical menus --
  >
<!ELEMENT OPTION - O (#PCDATA)         -- selectable choice -->
<!ATTLIST OPTION
  %attrs;                              -- %coreattrs, %i18n, %events --
  selected    (selected)     #IMPLIED
  disabled    (disabled)     #IMPLIED  -- unavailable in this context --
  label       %Text;         #IMPLIED  -- for use in hierarchical menus --
  value       CDATA          #IMPLIED  -- defaults to element content --
  >
<!ELEMENT TEXTAREA - - (#PCDATA)       -- multi-line text field -->
<!ATTLIST TEXTAREA
  %attrs;                              -- %coreattrs, %i18n, %events --
  name        CDATA          #IMPLIED
  rows        NUMBER         #REQUIRED
  cols        NUMBER         #REQUIRED
  disabled    (disabled)     #IMPLIED  -- unavailable in this context --
  readonly    (readonly)     #IMPLIED
  tabindex    NUMBER         #IMPLIED  -- position in tabbing order --
  accesskey   %Character;    #IMPLIED  -- accessibility key character --
  onfocus     %Script;       #IMPLIED  -- the element got the focus --
  onblur      %Script;       #IMPLIED  -- the element lost the focus --
  onselect    %Script;       #IMPLIED  -- some text was selected --
  onchange    %Script;       #IMPLIED  -- the element value was changed --
  %reserved;                           -- reserved for possible future use --
  >
<!--
  #PCDATA is to solve the mixed content problem,
  per specification only whitespace is allowed there!
 -->
<!ELEMENT FIELDSET - - (#PCDATA,LEGEND,(%flow;)*) -- form control group -->
<!ATTLIST FIELDSET
  %attrs;                              -- %coreattrs, %i18n, %events --
  >
<!ELEMENT LEGEND - - (%inline;)*       -- fieldset legend -->
<!ENTITY % LAlign "(top|bottom|left|right)">
<!ATTLIST LEGEND
  %attrs;                              -- %coreattrs, %i18n, %events --
  accesskey   %Character;    #IMPLIED  -- accessibility key character --
  align       %LAlign;       #IMPLIED  -- relative to fieldset --
  >
<!ELEMENT BUTTON - -
     (%flow;)* -(A|%formctrl;|FORM|ISINDEX|FIELDSET|IFRAME)
     -- push button -->
<!ATTLIST BUTTON
  %attrs;                              -- %coreattrs, %i18n, %events --
  name        CDATA          #IMPLIED
  value       CDATA          #IMPLIED  -- sent to server when submitted --
  type        (button|submit|reset) submit -- for use as form button --
  disabled    (disabled)     #IMPLIED  -- unavailable in this context --
  tabindex    NUMBER         #IMPLIED  -- position in tabbing order --
  accesskey   %Character;    #IMPLIED  -- accessibility key character --
  onfocus     %Script;       #IMPLIED  -- the element got the focus --
  onblur      %Script;       #IMPLIED  -- the element lost the focus --
  %reserved;                           -- reserved for possible future use --
  >
<!--========================= Tables ================================-->
<!-- IETF HTML table standard, see [RFC1942] -->
<!--
 The BORDER attribute sets the thickness of the frame around the
 table. The default units are screen pixels.
 The FRAME attribute specifies which parts of the frame around
 the table should be rendered. The values are not the same as
 CALS to avoid a name clash with the VALIGN attribute.
 The value "border" is included for backwards compatibility with
 <TABLE BORDER> which yields frame=border and border=implied
 For <TABLE BORDER=1> you get border=1 and frame=implied. In this
 case, it is appropriate to treat this as frame=border for backwards
 compatibility with deployed browsers.
-->
<!ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vsides|box|border)">
<!--
 The RULES attribute defines which rules to draw between cells:
 If RULES is absent then assume:
     "none" if BORDER is absent or BORDER=0 otherwise "all"
-->
<!ENTITY % TRules "(none | groups | rows | cols | all)">

<!-- horizontal placement of table relative to document -->
<!ENTITY % TAlign "(left|center|right)">
<!-- horizontal alignment attributes for cell contents -->
```

```
<!ENTITY % cellhalign
  "align      (left|center|right|justify|char) #IMPLIED
   char       %Character;    #IMPLIED  -- alignment char, e.g. char=':' --
   charoff    %Length;       #IMPLIED  -- offset for alignment char --"
  >
<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cellvalign
  "valign     (top|middle|bottom|baseline) #IMPLIED"
  >
<!ELEMENT TABLE - -
     (CAPTION?, (COL*|COLGROUP*), THEAD?, TFOOT?, TBODY+)>
<!ELEMENT CAPTION  - - (%inline;)*     -- table caption -->
<!ELEMENT THEAD    - O (TR)+           -- table header -->
<!ELEMENT TFOOT    - O (TR)+           -- table footer -->
<!ELEMENT TBODY    O O (TR)+           -- table body -->
<!ELEMENT COLGROUP - O (COL)*          -- table column group -->
<!ELEMENT COL      - O EMPTY           -- table column -->
<!ELEMENT TR       - O (TH|TD)+        -- table row -->
<!ELEMENT (TH|TD)  - O (%flow;)*       -- table header cell, table data cell-->
<!ATTLIST TABLE                        -- table element --
  %attrs;                              -- %coreattrs, %i18n, %events --
  summary     %Text;         #IMPLIED  -- purpose/structure for speech output--
  width       %Length;       #IMPLIED  -- table width --
  border      %Pixels;       #IMPLIED  -- controls frame width around table --
  frame       %TFrame;       #IMPLIED  -- which parts of frame to render --
  rules       %TRules;       #IMPLIED  -- rulings between rows and cols --
  cellspacing %Length;       #IMPLIED  -- spacing between cells --
  cellpadding %Length;       #IMPLIED  -- spacing within cells --
  align       %TAlign;       #IMPLIED  -- table position relative to window --
  bgcolor     %Color;        #IMPLIED  -- background color for cells --
  %reserved;                           -- reserved for possible future use --
  datapagesize CDATA         #IMPLIED  -- reserved for possible future use --
  >
<!ENTITY % CAlign "(top|bottom|left|right)">
<!ATTLIST CAPTION
  %attrs;                              -- %coreattrs, %i18n, %events --
  align       %CAlign;       #IMPLIED  -- relative to table --
  >
<!--
COLGROUP groups a set of COL elements. It allows you to group
several semantically related columns together.
-->
<!ATTLIST COLGROUP
  %attrs;                              -- %coreattrs, %i18n, %events --
  span        NUMBER         1         -- default number of columns in group --
  width       %MultiLength;  #IMPLIED  -- default width for enclosed COLs --
  %cellhalign;                         -- horizontal alignment in cells --
  %cellvalign;                         -- vertical alignment in cells --
  >
<!--
 COL elements define the alignment properties for cells in
 one or more columns.
 The WIDTH attribute specifies the width of the columns, e.g.
     width=64        width in screen pixels
     width=0.5*      relative width of 0.5
 The SPAN attribute causes the attributes of one
 COL element to apply to more than one column.
-->
<!ATTLIST COL                          -- column groups and properties --
  %attrs;                              -- %coreattrs, %i18n, %events --
  span        NUMBER         1         -- COL attributes affect N columns --
  width       %MultiLength;  #IMPLIED  -- column width specification --
  %cellhalign;                         -- horizontal alignment in cells --
  %cellvalign;                         -- vertical alignment in cells --
  >
<!--
    Use THEAD to duplicate headers when breaking table
    across page boundaries, or for static headers when
    TBODY sections are rendered in scrolling panel.
    Use TFOOT to duplicate footers when breaking table
    across page boundaries, or for static footers when
    TBODY sections are rendered in scrolling panel.
    Use multiple TBODY sections when rules are needed
    between groups of table rows.
-->
<!ATTLIST (THEAD|TBODY|TFOOT)          -- table section --
  %attrs;                              -- %coreattrs, %i18n, %events --
  %cellhalign;                         -- horizontal alignment in cells --
  %cellvalign;                         -- vertical alignment in cells --
  >
<!ATTLIST TR                           -- table row --
  %attrs;                              -- %coreattrs, %i18n, %events --
  %cellhalign;                         -- horizontal alignment in cells --
  %cellvalign;                         -- vertical alignment in cells --
  bgcolor     %Color;        #IMPLIED  -- background color for row --
  >
<!-- Scope is simpler than headers attribute for common tables -->
<!ENTITY % Scope "(row|col|rowgroup|colgroup)">
```

```
<!-- TH is for headers, TD for data, but for cells acting as both use TD -->
<!ATTLIST (TH|TD)                       -- header or data cell --
  %attrs;                               -- %coreattrs, %i18n, %events --
  abbr        %Text;          #IMPLIED  -- abbreviation for header cell --
  axis        CDATA           #IMPLIED  -- comma-separated list of related headers--
  headers     IDREFS          #IMPLIED  -- list of id's for header cells --
  scope       %Scope;         #IMPLIED  -- scope covered by header cells --
  rowspan     NUMBER          1         -- number of rows spanned by cell --
  colspan     NUMBER          1         -- number of cols spanned by cell --
  %cellhalign;                          -- horizontal alignment in cells --
  %cellvalign;                          -- vertical alignment in cells --
  nowrap      (nowrap)        #IMPLIED  -- suppress word wrap --
  bgcolor     %Color;         #IMPLIED  -- cell background color --
  width       %Length;        #IMPLIED  -- width for cell --
  height      %Length;        #IMPLIED  -- height for cell --
  >
<!--================== Document Frames ===================================-->
<!--
  The content model for HTML documents depends on whether the HEAD is
  followed by a FRAMESET or BODY element. The widespread omission of
  the BODY start tag makes it impractical to define the content model
  without the use of a marked section.
-->
<![ %HTML.Frameset; [
<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?) -- window subdivision-->
<!ATTLIST FRAMESET
  %coreattrs;                           -- id, class, style, title --
  rows        %MultiLengths; #IMPLIED   -- list of lengths,
                                           default: 100% (1 row) --
  cols        %MultiLengths; #IMPLIED   -- list of lengths,
                                           default: 100% (1 col) --
  onload      %Script;        #IMPLIED  -- all the frames have been loaded  --
  onunload    %Script;        #IMPLIED  -- all the frames have been removed --
  >
]]>
<![ %HTML.Frameset; [
<!-- reserved frame names start with "_" otherwise starts with letter -->
<!ELEMENT FRAME - O EMPTY               -- subwindow -->
<!ATTLIST FRAME
  %coreattrs;                           -- id, class, style, title --
  longdesc    %URI;           #IMPLIED  -- link to long description
                                           (complements title) --
  name        CDATA           #IMPLIED  -- name of frame for targetting --
  src         %URI;           #IMPLIED  -- source of frame content --
  frameborder (1|0)           1         -- request frame borders? --
  marginwidth %Pixels;        #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels;       #IMPLIED  -- margin height in pixels --
  noresize    (noresize)      #IMPLIED  -- allow users to resize frames? --
  scrolling   (yes|no|auto)   auto      -- scrollbar or none --
  >
]]>
<!ELEMENT IFRAME - - (%flow;)*          -- inline subwindow -->
<!ATTLIST IFRAME
  %coreattrs;                           -- id, class, style, title --
  longdesc    %URI;           #IMPLIED  -- link to long description
                                           (complements title) --
  name        CDATA           #IMPLIED  -- name of frame for targetting --
  src         %URI;           #IMPLIED  -- source of frame content --
  frameborder (1|0)           1         -- request frame borders? --
  marginwidth %Pixels;        #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels;       #IMPLIED  -- margin height in pixels --
  scrolling   (yes|no|auto)   auto      -- scrollbar or none --
  align       %IAlign;        #IMPLIED  -- vertical or horizontal alignment --
  height      %Length;        #IMPLIED  -- frame height --
  width       %Length;        #IMPLIED  -- frame width --
  >
<![ %HTML.Frameset; [
<!ENTITY % noframes.content "(BODY) -(NOFRAMES)">
]]>
<!ENTITY % noframes.content "(%flow;)*">
<!ELEMENT NOFRAMES - - %noframes.content;
 -- alternate content container for non frame-based rendering -->
<!ATTLIST NOFRAMES
  %attrs;                               -- %coreattrs, %i18n, %events --
  >
<!--================ Document Head ======================================-->
<!-- %head.misc; defined earlier on as "SCRIPT|STYLE|META|LINK|OBJECT" -->
<!ENTITY % head.content "TITLE & ISINDEX? & BASE?">
<!ELEMENT HEAD O O (%head.content;) +(%head.misc;) -- document head -->
<!ATTLIST HEAD
  %i18n;                                -- lang, dir --
  profile     %URI;           #IMPLIED  -- named dictionary of meta info --
  >
<!-- The TITLE element is not considered part of the flow of text.
       It should be displayed, for example as the page header or
       window title. Exactly one title is required per document.
    -->
<!ELEMENT TITLE - - (#PCDATA) -(%head.misc;) -- document title -->
<!ATTLIST TITLE %i18n>
```

```
<!ELEMENT ISINDEX - O EMPTY            -- single line prompt -->
<!ATTLIST ISINDEX
  %coreattrs;                          -- id, class, style, title --
  %i18n;                               -- lang, dir --
  prompt      %Text;         #IMPLIED  -- prompt message -->
<!ELEMENT BASE - O EMPTY               -- document base URI -->
<!ATTLIST BASE
  href        %URI;          #IMPLIED  -- URI that acts as base URI --
  target      %FrameTarget;  #IMPLIED  -- render in this frame --
  >
<!ELEMENT META - O EMPTY               -- generic metainformation -->
<!ATTLIST META
  %i18n;                               -- lang, dir, for use with content --
  http-equiv  NAME           #IMPLIED  -- HTTP response header name  --
  name        NAME           #IMPLIED  -- metainformation name --
  content     CDATA          #REQUIRED -- associated information --
  scheme      CDATA          #IMPLIED  -- select form of content --
  >
<!ELEMENT STYLE - - %StyleSheet        -- style info -->
<!ATTLIST STYLE
  %i18n;                               -- lang, dir, for use with title --
  type        %ContentType;  #REQUIRED -- content type of style language --
  media       %MediaDesc;    #IMPLIED  -- designed for use with these media --
  title       %Text;         #IMPLIED  -- advisory title --
  >
<!ELEMENT SCRIPT - - %Script;          -- script statements -->
<!ATTLIST SCRIPT
  charset     %Charset;      #IMPLIED  -- char encoding of linked resource --
  type        %ContentType;  #REQUIRED -- content type of script language --
  language    CDATA          #IMPLIED  -- predefined script language name --
  src         %URI;          #IMPLIED  -- URI for an external script --
  defer       (defer)        #IMPLIED  -- UA may defer execution of script --
  event       CDATA          #IMPLIED  -- reserved for possible future use --
  for         %URI;          #IMPLIED  -- reserved for possible future use --
  >
<!ELEMENT NOSCRIPT - - (%flow;)*
  -- alternate content container for non script-based rendering -->
<!ATTLIST NOSCRIPT
  %attrs;                              -- %coreattrs, %i18n, %events --
  >
<!--================ Document Structure ===================================-->
<!ENTITY % version "version CDATA #FIXED '%HTML.Version;'">
<![ %HTML.Frameset; [
<!ENTITY % html.content "HEAD, FRAMESET">
]]>
<!ENTITY % html.content "HEAD, BODY">
<!ELEMENT HTML O O (%html.content;)    -- document root element -->
<!ATTLIST HTML
  %i18n;                               -- lang, dir --
  %version;
  >
```

Appendix E. The XHTML 1.0 DTD

The XHTML 1.0 standard is formally defined as three XML Document Type Definitions (DTDs): the Strict DTD, the Transitional DTD, and the Frameset DTD. These DTDs correspond to the respective HTML 4.01 DTDs, defining the same elements and attributes using XML instead of SGML as the DTD authoring language.

The Strict DTD defines only those elements that are not deprecated in the HTML 4.01 standard. Ideally, everyone would create XHTML documents that conform to the Strict DTD. The Transitional DTD includes all those deprecated elements and more accurately reflects the HTML in use today, with many older elements still in common use. The Frameset DTD is identical to the Transitional DTD, with the exception that the document `<body>` is replaced by the `<frameset>` element.

Since the HTML Transitional DTD is the one upon which this book is based, it is only appropriate that we include the corresponding XHTML DTD. Note that we have reprinted this DTD verbatim and have not attempted to add extensions to it. Where our description and the DTD deviate, assume the DTD is correct:

```
<!--
   Extensible HTML version 1.0 Transitional DTD
   This is the same as HTML 4.0 Transitional except for
   changes due to the differences between XML and SGML.
   Namespace = http://www.w3.org/1999/xhtml
   For further information, see: http://www.w3.org/TR/xhtml1
   Copyright (c) 1998-2000 W3C (MIT, INRIA, Keio),
   All Rights Reserved.
   This DTD module is identified by the PUBLIC and SYSTEM identifiers:
   PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
   $Revision: 1.4 $
   $Date: 2001/08/22 23:32:18 $
-->
<!--================ Character mnemonic entities =========================-->
<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin 1 for XHTML//EN"
   "xhtml-lat1.ent">
%HTMLlat1;
<!ENTITY % HTMLsymbol PUBLIC
   "-//W3C//ENTITIES Symbols for XHTML//EN"
   "xhtml-symbol.ent">
%HTMLsymbol;
<!ENTITY % HTMLspecial PUBLIC
   "-//W3C//ENTITIES Special for XHTML//EN"
   "xhtml-special.ent">
%HTMLspecial;
<!--================== Imported Names ====================================-->
<!ENTITY % ContentType "CDATA">
    <!-- media type, as per [RFC2045] -->
<!ENTITY % ContentTypes "CDATA">
    <!-- comma-separated list of media types, as per [RFC2045] -->
<!ENTITY % Charset "CDATA">
    <!-- a character encoding, as per [RFC2045] -->
<!ENTITY % Charsets "CDATA">
    <!-- a space separated list of character encodings, as per [RFC2045] -->
<!ENTITY % LanguageCode "NMTOKEN">
    <!-- a language code, as per [RFC1766] -->
<!ENTITY % Character "CDATA">
    <!-- a single character from [ISO10646] -->
<!ENTITY % Number "CDATA">
    <!-- one or more digits -->
<!ENTITY % LinkTypes "CDATA">
    <!-- space-separated list of link types -->
<!ENTITY % MediaDesc "CDATA">
    <!-- single or comma-separated list of media descriptors -->
<!ENTITY % URI "CDATA">
    <!-- a Uniform Resource Identifier, see [RFC2396] -->
<!ENTITY % UriList "CDATA">
    <!-- a space separated list of Uniform Resource Identifiers -->
<!ENTITY % Datetime "CDATA">
    <!-- date and time information. ISO date format -->
<!ENTITY % Script "CDATA">
    <!-- script expression -->
<!ENTITY % StyleSheet "CDATA">
    <!-- style sheet data -->
<!ENTITY % Text "CDATA">
    <!-- used for titles etc. -->
<!ENTITY % FrameTarget "NMTOKEN">
    <!-- render in this frame -->
<!ENTITY % Length "CDATA">
    <!-- nn for pixels or nn% for percentage length -->
<!ENTITY % MultiLength "CDATA">
    <!-- pixel, percentage, or relative -->
<!ENTITY % MultiLengths "CDATA">
    <!-- comma-separated list of MultiLength -->
```

```
<!ENTITY % Pixels "CDATA">
    <!-- integer representing length in pixels -->
<!-- these are used for image maps -->
<!ENTITY % Shape "(rect|circle|poly|default)">
<!ENTITY % Coords "CDATA">
    <!-- comma separated list of lengths -->
<!-- used for object, applet, img, input and iframe -->
<!ENTITY % ImgAlign "(top|middle|bottom|left|right)">
<!-- a color using sRGB: #RRGGBB as Hex values -->
<!ENTITY % Color "CDATA">
<!-- There are also 16 widely known color names with their sRGB values:
    Black  = #000000    Green  = #008000
    Silver = #C0C0C0    Lime   = #00FF00
    Gray   = #808080    Olive  = #808000
    White  = #FFFFFF    Yellow = #FFFF00
    Maroon = #800000    Navy   = #000080
    Red    = #FF0000    Blue   = #0000FF
    Purple = #800080    Teal   = #008080
    Fuchsia= #FF00FF    Aqua   = #00FFFF
-->
<!--================== Generic Attributes ===============================-->
<!-- core attributes common to most elements
  id       document-wide unique id
  class    space separated list of classes
  style    associated style info
  title    advisory title/amplification
-->
<!ENTITY % coreattrs
 "id          ID             #IMPLIED
  class       CDATA          #IMPLIED
  style       %StyleSheet;   #IMPLIED
  title       %Text;         #IMPLIED"
  >
<!-- internationalization attributes
  lang        language code (backwards compatible)
  xml:lang    language code (as per XML 1.0 spec)
  dir         direction for weak/neutral text
-->
<!ENTITY % i18n
 "lang        %LanguageCode; #IMPLIED
  xml:lang    %LanguageCode; #IMPLIED
  dir         (ltr|rtl)      #IMPLIED"
  >
<!-- attributes for common UI events
  onclick     a pointer button was clicked
  ondblclick  a pointer button was double clicked
  onmousedown a pointer button was pressed down
  onmouseup   a pointer button was released
  onmousemove a pointer was moved onto the element
  onmouseout  a pointer was moved away from the element
  onkeypress  a key was pressed and released
  onkeydown   a key was pressed down
  onkeyup     a key was released
-->
<!ENTITY % events
 "onclick     %Script;       #IMPLIED
  ondblclick  %Script;       #IMPLIED
  onmousedown %Script;       #IMPLIED
  onmouseup   %Script;       #IMPLIED
  onmouseover %Script;       #IMPLIED
  onmousemove %Script;       #IMPLIED
  onmouseout  %Script;       #IMPLIED
  onkeypress  %Script;       #IMPLIED
  onkeydown   %Script;       #IMPLIED
  onkeyup     %Script;       #IMPLIED"
  >
<!-- attributes for elements that can get the focus
  accesskey   accessibility key character
  tabindex    position in tabbing order
  onfocus     the element got the focus
  onblur      the element lost the focus
-->
<!ENTITY % focus
 "accesskey   %Character;    #IMPLIED
  tabindex    %Number;       #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED"
  >
<!ENTITY % attrs "%coreattrs; %i18n; %events;">
<!-- text alignment for p, div, h1-h6. The default is
     align="left" for ltr headings, "right" for rtl -->
<!ENTITY % TextAlign "align (left|center|right) #IMPLIED">
<!--================== Text Elements ====================================-->
<!ENTITY % special
   "br | span | bdo | object | applet | img | map | iframe">
<!ENTITY % fontstyle "tt | i | b | big | small | u
                      | s | strike |font | basefont">
<!ENTITY % phrase "em | strong | dfn | code | q | sub | sup |
                   samp | kbd | var | cite | abbr | acronym">
```

```
<!ENTITY % inline.forms "input | select | textarea | label | button">
<!-- these can occur at block or inline level -->
<!ENTITY % misc "ins | del | script | noscript">
<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; | %inline.forms;">
<!-- %Inline; covers inline or "text-level" elements -->
<!ENTITY % Inline "(#PCDATA | %inline; | %misc;)*">
<!--================== Block level elements ==============================-->
<!ENTITY % heading "h1|h2|h3|h4|h5|h6">
<!ENTITY % lists "ul | ol | dl | menu | dir">
<!ENTITY % blocktext "pre | hr | blockquote | address | center | noframes">
<!ENTITY % block
    "p | %heading; | div | %lists; | %blocktext; | isindex |fieldset | table">
<!ENTITY % Block "(%block; | form | %misc;)*">
<!-- %Flow; mixes Block and Inline and is used for list items etc. -->
<!ENTITY % Flow "(#PCDATA | %block; | form | %inline; | %misc;)*">
<!--================== Content models for exclusions =====================-->
<!-- a elements use %Inline; excluding a -->
<!ENTITY % a.content
   "(#PCDATA | %special; | %fontstyle; | %phrase; | %inline.forms; | %misc;)*">
<!-- pre uses %Inline excluding img, object, applet, big, small,
     sub, sup, font, or basefont -->
<!ENTITY % pre.content
   "(#PCDATA | a | br | span | bdo | map | tt | i | b | u | s |
      %phrase; | %inline.forms;)*">
<!-- form uses %Flow; excluding form -->
<!ENTITY % form.content "(#PCDATA | %block; | %inline; | %misc;)*">
<!-- button uses %Flow; but excludes a, form, form controls, iframe -->
<!ENTITY % button.content
   "(#PCDATA | p | %heading; | div | %lists; | %blocktext; |
      table | br | span | bdo | object | applet | img | map |
      %fontstyle; | %phrase; | %misc;)*">
<!--================ Document Structure ==================================-->
<!-- the namespace URI designates the document profile -->
<!ELEMENT html (head, body)>
<!ATTLIST html
  %i18n;
  xmlns       %URI;          #FIXED 'http://www.w3.org/1999/xhtml'
  >
<!--================ Document Head =======================================-->
<!ENTITY % head.misc "(script|style|meta|link|object|isindex)*">
<!-- content model is %head.misc; combined with a single
     title and an optional base element in any order -->
<!ELEMENT head (%head.misc;,
     ((title, %head.misc;, (base, %head.misc;)?) |
      (base, %head.misc;, (title, %head.misc;))))>
<!ATTLIST head
  %i18n;
  profile     %URI;          #IMPLIED
  >
<!-- The title element is not considered part of the flow of text.
       It should be displayed, for example as the page header or
       window title. Exactly one title is required per document.
    -->
<!ELEMENT title (#PCDATA)>
<!ATTLIST title %i18n;>
<!-- document base URI -->
<!ELEMENT base EMPTY>
<!ATTLIST base
  href        %URI;          #IMPLIED
  target      %FrameTarget;  #IMPLIED
  >
<!-- generic metainformation -->
<!ELEMENT meta EMPTY>
<!ATTLIST meta
  %i18n;
  http-equiv  CDATA          #IMPLIED
  name        CDATA          #IMPLIED
  content     CDATA          #REQUIRED
  scheme      CDATA          #IMPLIED
  >
<!--
  Relationship values can be used in principle:
   a) for document specific toolbars/menus when used
      with the link element in document head e.g.
      start, contents, previous, next, index, end, help
   b) to link to a separate style sheet (rel="stylesheet")
   c) to make a link to a script (rel="script")
   d) by stylesheets to control how collections of
      html nodes are rendered into printed documents
   e) to make a link to a printable version of this document
      e.g. a PostScript or PDF version (rel="alternate" media="print")
-->
```

```
<!ELEMENT link EMPTY>
<!ATTLIST link
  %attrs;
  charset     %Charset;      #IMPLIED
  href        %URI;          #IMPLIED
  hreflang    %LanguageCode; #IMPLIED
  type        %ContentType;  #IMPLIED
  rel         %LinkTypes;    #IMPLIED
  rev         %LinkTypes;    #IMPLIED
  media       %MediaDesc;    #IMPLIED
  target      %FrameTarget;  #IMPLIED
  >
<!-- style info, which may include CDATA sections -->
<!ELEMENT style (#PCDATA)>
<!ATTLIST style
  %i18n;
  type        %ContentType;  #REQUIRED
  media       %MediaDesc;    #IMPLIED
  title       %Text;         #IMPLIED
  xml:space   (preserve)     #FIXED 'preserve'
  >
<!-- script statements, which may include CDATA sections -->
<!ELEMENT script (#PCDATA)>
<!ATTLIST script
  charset     %Charset;      #IMPLIED
  type        %ContentType;  #REQUIRED
  language    CDATA          #IMPLIED
  src         %URI;          #IMPLIED
  defer       (defer)        #IMPLIED
  xml:space   (preserve)     #FIXED 'preserve'
  >
<!-- alternate content container for non script-based rendering -->
<!ELEMENT noscript %Flow;>
<!ATTLIST noscript
  %attrs;
  >
<!--======================== Frames ========================================-->
<!-- inline subwindow -->
<!ELEMENT iframe %Flow;>
<!ATTLIST iframe
  %coreattrs;
  longdesc    %URI;          #IMPLIED
  name        NMTOKEN        #IMPLIED
  src         %URI;          #IMPLIED
  frameborder (1|0)          "1"
  marginwidth %Pixels;       #IMPLIED
  marginheight %Pixels;      #IMPLIED
  scrolling   (yes|no|auto)  "auto"
  align       %ImgAlign;     #IMPLIED
  height      %Length;       #IMPLIED
  width       %Length;       #IMPLIED
  >
<!-- alternate content container for non frame-based rendering -->
<!ELEMENT noframes %Flow;>
<!ATTLIST noframes
  %attrs;
  >
<!--=================== Document Body ======================================-->
<!ELEMENT body %Flow;>
<!ATTLIST body
  %attrs;
  onload      %Script;       #IMPLIED
  onunload    %Script;       #IMPLIED
  background  %URI;          #IMPLIED
  bgcolor     %Color;        #IMPLIED
  text        %Color;        #IMPLIED
  link        %Color;        #IMPLIED
  vlink       %Color;        #IMPLIED
  alink       %Color;        #IMPLIED
  >
<!ELEMENT div %Flow;>  <!-- generic language/style container -->
<!ATTLIST div
  %attrs;
  %TextAlign;
  >
<!--=================== Paragraphs =========================================-->
<!ELEMENT p %Inline;>
<!ATTLIST p
  %attrs;
  %TextAlign;
  >
<!--=================== Headings ===========================================-->
<!--
  There are six levels of headings from h1 (the most important)
  to h6 (the least important).
-->
<!ELEMENT h1  %Inline;>
```

```
<!ATTLIST h1
  %attrs;
  %TextAlign;
  >
<!ELEMENT h2 %Inline;>
<!ATTLIST h2
  %attrs;
  %TextAlign;
  >
<!ELEMENT h3 %Inline;>
<!ATTLIST h3
  %attrs;
  %TextAlign;
  >
<!ELEMENT h4 %Inline;>
<!ATTLIST h4
  %attrs;
  %TextAlign;
  >
<!ELEMENT h5 %Inline;>
<!ATTLIST h5
  %attrs;
  %TextAlign;
  >
<!ELEMENT h6 %Inline;>
<!ATTLIST h6
  %attrs;
  %TextAlign;
  >
<!--=================== Lists ===================================================-->
<!-- Unordered list bullet styles -->
<!ENTITY % ULStyle "(disc|square|circle)">
<!-- Unordered list -->
<!ELEMENT ul (li)+>
<!ATTLIST ul
  %attrs;
  type        %ULStyle;     #IMPLIED
  compact     (compact)     #IMPLIED
  >
<!-- Ordered list numbering style
    1   arabic numbers      1, 2, 3, ...
    a   lower alpha         a, b, c, ...
    A   upper alpha         A, B, C, ...
    i   lower roman         i, ii, iii, ...
    I   upper roman         I, II, III, ...
    The style is applied to the sequence number which by default
    is reset to 1 for the first list item in an ordered list.
-->
<!ENTITY % OLStyle "CDATA">
<!-- Ordered (numbered) list -->
<!ELEMENT ol (li)+>
<!ATTLIST ol
  %attrs;
  type        %OLStyle;      #IMPLIED
  compact     (compact)      #IMPLIED
  start       %Number;       #IMPLIED
  >
<!-- single column list (DEPRECATED) -->
<!ELEMENT menu (li)+>
<!ATTLIST menu
  %attrs;
  compact     (compact)     #IMPLIED
  >
<!-- multiple column list (DEPRECATED) -->
<!ELEMENT dir (li)+>
<!ATTLIST dir
  %attrs;
  compact     (compact)     #IMPLIED
  >
<!-- LIStyle is constrained to: "(%ULStyle;|%OLStyle;)" -->
<!ENTITY % LIStyle "CDATA">
<!-- list item -->
<!ELEMENT li %Flow;>
<!ATTLIST li
  %attrs;
  type        %LIStyle;      #IMPLIED
  value       %Number;       #IMPLIED
  >
<!-- definition lists - dt for term, dd for its definition -->
<!ELEMENT dl (dt|dd)+>
<!ATTLIST dl
  %attrs;
  compact     (compact)      #IMPLIED
  >
<!ELEMENT dt %Inline;>
<!ATTLIST dt
  %attrs;
  >
<!ELEMENT dd %Flow;>
```

```
<!ATTLIST dd
  %attrs;
  >
<!--=================== Address ===============================================-->
<!-- information on author -->
<!ELEMENT address %Inline;>
<!ATTLIST address
  %attrs;
  >
<!--=================== Horizontal Rule =======================================-->
<!ELEMENT hr EMPTY>
<!ATTLIST hr
  %attrs;
  align       (left|center|right) #IMPLIED
  noshade     (noshade)      #IMPLIED
  size        %Pixels;       #IMPLIED
  width       %Length;       #IMPLIED
  >
<!--=================== Preformatted Text =====================================-->
<!-- content is %Inline; excluding
        "img|object|applet|big|small|sub|sup|font|basefont" -->
<!ELEMENT pre %pre.content;>
<!ATTLIST pre
  %attrs;
  width       %Number;      #IMPLIED
  xml:space   (preserve)    #FIXED 'preserve'
  >
<!--=================== Block-like Quotes =====================================-->
<!ELEMENT blockquote %Flow;>
<!ATTLIST blockquote
  %attrs;
  cite        %URI;          #IMPLIED
  >
<!--=================== Text alignment ========================================-->
<!-- center content -->
<!ELEMENT center %Flow;>
<!ATTLIST center
  %attrs;
  >
<!--=================== Inserted/Deleted Text =================================-->
<!--
  ins/del are allowed in block and inline content, but its
  inappropriate to include block content within an ins element
  occurring in inline content.
-->
<!ELEMENT ins %Flow;>
<!ATTLIST ins
  %attrs;
  cite        %URI;          #IMPLIED
  datetime    %Datetime;     #IMPLIED
  >
<!ELEMENT del %Flow;>
<!ATTLIST del
  %attrs;
  cite        %URI;          #IMPLIED
  datetime    %Datetime;     #IMPLIED
  >
<!--=================== The Anchor Element ====================================-->
<!-- content is %Inline; except that anchors shouldn't be nested -->
<!ELEMENT a %a.content;>
<!ATTLIST a
  %attrs;
  charset     %Charset;      #IMPLIED
  type        %ContentType;  #IMPLIED
  name        NMTOKEN        #IMPLIED
  href        %URI;          #IMPLIED
  hreflang    %LanguageCode; #IMPLIED
  rel         %LinkTypes;    #IMPLIED
  rev         %LinkTypes;    #IMPLIED
  accesskey   %Character;    #IMPLIED
  shape       %Shape;        "rect"
  coords      %Coords;       #IMPLIED
  tabindex    %Number;       #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  target      %FrameTarget;  #IMPLIED
  >
<!--=================== Inline Elements ====================================-->
<!ELEMENT span %Inline;> <!-- generic language/style container -->
<!ATTLIST span
  %attrs;
  >
```

```
<!ELEMENT bdo %Inline;>  <!-- I18N BiDi over-ride -->
<!ATTLIST bdo
  %coreattrs;
  %events;
  lang        %LanguageCode; #IMPLIED
  xml:lang    %LanguageCode; #IMPLIED
  dir         (ltr|rtl)      #REQUIRED
  >
<!ELEMENT br EMPTY>   <!-- forced line break -->
<!ATTLIST br
  %coreattrs;
  clear       (left|all|right|none) "none"
  >
<!ELEMENT em %Inline;>   <!-- emphasis -->
<!ATTLIST em %attrs;>
<!ELEMENT strong %Inline;>   <!-- strong emphasis -->
<!ATTLIST strong %attrs;>
<!ELEMENT dfn %Inline;>   <!-- definitional -->
<!ATTLIST dfn %attrs;>
<!ELEMENT code %Inline;>   <!-- program code -->
<!ATTLIST code %attrs;>
<!ELEMENT samp %Inline;>   <!-- sample -->
<!ATTLIST samp %attrs;>
<!ELEMENT kbd %Inline;>  <!-- something user would type -->
<!ATTLIST kbd %attrs;>
<!ELEMENT var %Inline;>   <!-- variable -->
<!ATTLIST var %attrs;>
<!ELEMENT cite %Inline;>   <!-- citation -->
<!ATTLIST cite %attrs;>
<!ELEMENT abbr %Inline;>   <!-- abbreviation -->
<!ATTLIST abbr %attrs;>
<!ELEMENT acronym %Inline;>   <!-- acronym -->
<!ATTLIST acronym %attrs;>
<!ELEMENT q %Inline;>   <!-- inlined quote -->
<!ATTLIST q
  %attrs;
  cite        %URI;          #IMPLIED
  >
<!ELEMENT sub %Inline;> <!-- subscript -->
<!ATTLIST sub %attrs;>
<!ELEMENT sup %Inline;> <!-- superscript -->
<!ATTLIST sup %attrs;>
<!ELEMENT tt %Inline;>   <!-- fixed pitch font -->
<!ATTLIST tt %attrs;>
<!ELEMENT i %Inline;>   <!-- italic font -->
<!ATTLIST i %attrs;>
<!ELEMENT b %Inline;>   <!-- bold font -->
<!ATTLIST b %attrs;>
<!ELEMENT big %Inline;>   <!-- bigger font -->
<!ATTLIST big %attrs;>
<!ELEMENT small %Inline;>   <!-- smaller font -->
<!ATTLIST small %attrs;>
<!ELEMENT u %Inline;>   <!-- underline -->
<!ATTLIST u %attrs;>
<!ELEMENT s %Inline;>   <!-- strike-through -->
<!ATTLIST s %attrs;>
<!ELEMENT strike %Inline;>   <!-- strike-through -->
<!ATTLIST strike %attrs;>
<!ELEMENT basefont EMPTY>  <!-- base font size -->
<!ATTLIST basefont
  id          ID             #IMPLIED
  size        CDATA          #REQUIRED
  color       %Color;        #IMPLIED
  face        CDATA          #IMPLIED
  >
<!ELEMENT font %Inline;> <!-- local change to font -->
<!ATTLIST font
  %coreattrs;
  %i18n;
  size        CDATA          #IMPLIED
  color       %Color;        #IMPLIED
  face        CDATA          #IMPLIED
  >
<!--====================== Object ======================================-->
<!--
  object is used to embed objects as part of HTML pages.
  param elements should precede other content. Parameters
  can also be expressed as attribute/value pairs on the
  object element itself when brevity is desired.
-->
```

```
<!ELEMENT object (#PCDATA | param | %block; | form | %inline; | %misc;)*>
<!ATTLIST object
  %attrs;
  declare     (declare)      #IMPLIED
  classid     %URI;          #IMPLIED
  codebase    %URI;          #IMPLIED
  data        %URI;          #IMPLIED
  type        %ContentType;  #IMPLIED
  codetype    %ContentType;  #IMPLIED
  archive     %UriList;      #IMPLIED
  standby     %Text;         #IMPLIED
  height      %Length;       #IMPLIED
  width       %Length;       #IMPLIED
  usemap      %URI;          #IMPLIED
  name        NMTOKEN        #IMPLIED
  tabindex    %Number;       #IMPLIED
  align       %ImgAlign;     #IMPLIED
  border      %Pixels;       #IMPLIED
  hspace      %Pixels;       #IMPLIED
  vspace      %Pixels;       #IMPLIED
  >
<!--
  param is used to supply a named property value.
  In XML it would seem natural to follow RDF and support an
  abbreviated syntax where the param elements are replaced
  by attribute value pairs on the object start tag.
-->
<!ELEMENT param EMPTY>
<!ATTLIST param
  id          ID             #IMPLIED
  name        CDATA          #REQUIRED
  value       CDATA          #IMPLIED
  valuetype   (data|ref|object) "data"
  type        %ContentType;  #IMPLIED
  >
<!--=================== Java applet ====================================-->
<!--
  One of code or object attributes must be present.
  Place param elements before other content.
-->
<!ELEMENT applet (#PCDATA | param | %block; | form | %inline; | %misc;)*>
<!ATTLIST applet
  %coreattrs;
  codebase    %URI;          #IMPLIED
  archive     CDATA          #IMPLIED
  code        CDATA          #IMPLIED
  object      CDATA          #IMPLIED
  alt         %Text;         #IMPLIED
  name        NMTOKEN        #IMPLIED
  width       %Length;       #REQUIRED
  height      %Length;       #REQUIRED
  align       %ImgAlign;     #IMPLIED
  hspace      %Pixels;       #IMPLIED
  vspace      %Pixels;       #IMPLIED
  >
<!--=================== Images ============================================-->
<!--
   To avoid accessibility problems for people who aren't
   able to see the image, you should provide a text
   description using the alt and longdesc attributes.
   In addition, avoid the use of server-side image maps.
-->
<!ELEMENT img EMPTY>
<!ATTLIST img
  %attrs;
  src         %URI;          #REQUIRED
  alt         %Text;         #REQUIRED
  name        NMTOKEN        #IMPLIED
  longdesc    %URI;          #IMPLIED
  height      %Length;       #IMPLIED
  width       %Length;       #IMPLIED
  usemap      %URI;          #IMPLIED
  ismap       (ismap)        #IMPLIED
  align       %ImgAlign;     #IMPLIED
  border      %Length;       #IMPLIED
  hspace      %Pixels;       #IMPLIED
  vspace      %Pixels;       #IMPLIED
  >
<!-- usemap points to a map element which may be in this document
  or an external document, although the latter is not widely supported -->
<!--================== Client-side image maps ============================-->
<!-- These can be placed in the same document or grouped in a
     separate document although this isn't yet widely supported -->
<!ELEMENT map ((%block; | form | %misc;)+ | area+)>
```

```
<!ATTLIST map
  %i18n;
  %events;
  id          ID             #REQUIRED
  class       CDATA          #IMPLIED
  style       %StyleSheet;   #IMPLIED
  title       %Text;         #IMPLIED
  name        CDATA          #IMPLIED
  >
<!ELEMENT area EMPTY>
<!ATTLIST area
  %attrs;
  shape       %Shape;        "rect"
  coords      %Coords;       #IMPLIED
  href        %URI;          #IMPLIED
  nohref      (nohref)       #IMPLIED
  alt         %Text;         #REQUIRED
  tabindex    %Number;       #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  target      %FrameTarget;  #IMPLIED
  >
<!--================ Forms ===============================================-->
<!ELEMENT form %form.content;>   <!-- forms shouldn't be nested -->
<!ATTLIST form
  %attrs;
  action      %URI;          #REQUIRED
  method      (get|post)     "get"
  name        NMTOKEN        #IMPLIED
  enctype     %ContentType;  "application/x-www-form-urlencoded"
  onsubmit    %Script;       #IMPLIED
  onreset     %Script;       #IMPLIED
  accept      %ContentTypes; #IMPLIED
  accept-charset %Charsets;  #IMPLIED
  target      %FrameTarget;  #IMPLIED
  >
<!--
  Each label must not contain more than ONE field
  Label elements shouldn't be nested.
-->
<!ELEMENT label %Inline;>
<!ATTLIST label
  %attrs;
  for         IDREF          #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  >
<!ENTITY % InputType
  "(text | password | checkbox |
    radio | submit | reset |
    file | hidden | image | button)"
   >
<!-- the name attribute is required for all but submit & reset -->
<!ELEMENT input EMPTY>     <!-- form control -->
<!ATTLIST input
  %attrs;
  type        %InputType;    "text"
  name        CDATA          #IMPLIED
  value       CDATA          #IMPLIED
  checked     (checked)      #IMPLIED
  disabled    (disabled)     #IMPLIED
  readonly    (readonly)     #IMPLIED
  size        CDATA          #IMPLIED
  maxlength   %Number;       #IMPLIED
  src         %URI;          #IMPLIED
  alt         CDATA          #IMPLIED
  usemap      %URI;          #IMPLIED
  tabindex    %Number;       #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  onselect    %Script;       #IMPLIED
  onchange    %Script;       #IMPLIED
  accept      %ContentTypes; #IMPLIED
  align       %ImgAlign;     #IMPLIED
  >
<!ELEMENT select (optgroup|option)+>  <!-- option selector -->
```

```
<!ATTLIST select
  %attrs;
  name        CDATA          #IMPLIED
  size        %Number;       #IMPLIED
  multiple    (multiple)     #IMPLIED
  disabled    (disabled)     #IMPLIED
  tabindex    %Number;       #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  onchange    %Script;       #IMPLIED
  >
<!ELEMENT optgroup (option)+>   <!-- option group -->
<!ATTLIST optgroup
  %attrs;
  disabled    (disabled)     #IMPLIED
  label       %Text;         #REQUIRED
  >
<!ELEMENT option (#PCDATA)>     <!-- selectable choice -->
<!ATTLIST option
  %attrs;
  selected    (selected)     #IMPLIED
  disabled    (disabled)     #IMPLIED
  label       %Text;         #IMPLIED
  value       CDATA          #IMPLIED
  >
<!ELEMENT textarea (#PCDATA)>     <!-- multi-line text field -->
<!ATTLIST textarea
  %attrs;
  name        CDATA          #IMPLIED
  rows        %Number;       #REQUIRED
  cols        %Number;       #REQUIRED
  disabled    (disabled)     #IMPLIED
  readonly    (readonly)     #IMPLIED
  tabindex    %Number;       #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  onselect    %Script;       #IMPLIED
  onchange    %Script;       #IMPLIED
  >
<!--
  The fieldset element is used to group form fields.
  Only one legend element should occur in the content
  and if present should only be preceded by whitespace.
-->
<!ELEMENT fieldset (#PCDATA | legend | %block; | form | %inline; | %misc;)*>
<!ATTLIST fieldset
  %attrs;
  >
<!ENTITY % LAlign "(top|bottom|left|right)">
<!ELEMENT legend %Inline;>     <!-- fieldset label -->
<!ATTLIST legend
  %attrs;
  accesskey   %Character;    #IMPLIED
  align       %LAlign;       #IMPLIED
  >
<!--
 Content is %Flow; excluding a, form, form controls, iframe
-->
<!ELEMENT button %button.content;>  <!-- push button -->
<!ATTLIST button
  %attrs;
  name        CDATA          #IMPLIED
  value       CDATA          #IMPLIED
  type        (button|submit|reset) "submit"
  disabled    (disabled)     #IMPLIED
  tabindex    %Number;       #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  >
<!-- single-line text input control (DEPRECATED) -->
<!ELEMENT isindex EMPTY>
<!ATTLIST isindex
  %coreattrs;
  %i18n;
  prompt      %Text;         #IMPLIED
  >
<!--======================= Tables ===================================-->
<!-- Derived from IETF HTML table standard, see [RFC1942] -->
<!--
 The border attribute sets the thickness of the frame around the
 table. The default units are screen pixels.
 The frame attribute specifies which parts of the frame around
 the table should be rendered. The values are not the same as
 CALS to avoid a name clash with the valign attribute.
-->
```

```
<!ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vsides|box|border)">
<!--
 The rules attribute defines which rules to draw between cells:
 If rules is absent then assume:
     "none" if border is absent or border="0" otherwise "all"
-->
<!ENTITY % TRules "(none | groups | rows | cols | all)">

<!-- horizontal placement of table relative to document -->
<!ENTITY % TAlign "(left|center|right)">
<!-- horizontal alignment attributes for cell contents
  char        alignment char, e.g. char=':'
  charoff     offset for alignment char
-->
<!ENTITY % cellhalign
  "align      (left|center|right|justify|char) #IMPLIED
   char       %Character;    #IMPLIED
   charoff    %Length;       #IMPLIED"
  >
<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cellvalign
  "valign     (top|middle|bottom|baseline) #IMPLIED"
  >
<!ELEMENT table
     (caption?, (col*|colgroup*), thead?, tfoot?, (tbody+|tr+))>
<!ELEMENT caption  %Inline;>
<!ELEMENT thead    (tr)+>
<!ELEMENT tfoot    (tr)+>
<!ELEMENT tbody    (tr)+>
<!ELEMENT colgroup (col)*>
<!ELEMENT col      EMPTY>
<!ELEMENT tr       (th|td)+>
<!ELEMENT th       %Flow;>
<!ELEMENT td       %Flow;>
<!ATTLIST table
  %attrs;
  summary     %Text;         #IMPLIED
  width       %Length;       #IMPLIED
  border      %Pixels;       #IMPLIED
  frame       %TFrame;       #IMPLIED
  rules       %TRules;       #IMPLIED
  cellspacing %Length;       #IMPLIED
  cellpadding %Length;       #IMPLIED
  align       %TAlign;       #IMPLIED
  bgcolor     %Color;        #IMPLIED
  >
<!ENTITY % CAlign "(top|bottom|left|right)">
<!ATTLIST caption
  %attrs;
  align       %CAlign;       #IMPLIED
  >
<!--
colgroup groups a set of col elements. It allows you to group
several semantically related columns together.
-->
<!ATTLIST colgroup
  %attrs;
  span        %Number;       "1"
  width       %MultiLength;  #IMPLIED
  %cellhalign;
  %cellvalign;
  >
<!--
 col elements define the alignment properties for cells in
 one or more columns.
 The width attribute specifies the width of the columns, e.g.
     width=64        width in screen pixels
     width=0.5*      relative width of 0.5
 The span attribute causes the attributes of one
 col element to apply to more than one column.
-->
<!ATTLIST col
  %attrs;
  span        %Number;       "1"
  width       %MultiLength;  #IMPLIED
  %cellhalign;
  %cellvalign;
  >
<!--
    Use thead to duplicate headers when breaking table
    across page boundaries, or for static headers when
    tbody sections are rendered in scrolling panel.
    Use tfoot to duplicate footers when breaking table
    across page boundaries, or for static footers when
    tbody sections are rendered in scrolling panel.
    Use multiple tbody sections when rules are needed
    between groups of table rows.
-->
```

```
<!ATTLIST thead
  %attrs;
  %cellhalign;
  %cellvalign;
  >
<!ATTLIST tfoot
  %attrs;
  %cellhalign;
  %cellvalign;
  >
<!ATTLIST tbody
  %attrs;
  %cellhalign;
  %cellvalign;
  >
<!ATTLIST tr
  %attrs;
  %cellhalign;
  %cellvalign;
  bgcolor     %Color;        #IMPLIED
  >
<!-- Scope is simpler than headers attribute for common tables -->
<!ENTITY % Scope "(row|col|rowgroup|colgroup)">
<!-- th is for headers, td for data and for cells acting as both -->
<!ATTLIST th
  %attrs;
  abbr        %Text;         #IMPLIED
  axis        CDATA          #IMPLIED
  headers     IDREFS         #IMPLIED
  scope       %Scope;        #IMPLIED
  rowspan     %Number;       "1"
  colspan     %Number;       "1"
  %cellhalign;
  %cellvalign;
  nowrap      (nowrap)       #IMPLIED
  bgcolor     %Color;        #IMPLIED
  width       %Pixels;       #IMPLIED
  height      %Pixels;       #IMPLIED
  >
<!ATTLIST td
  %attrs;
  abbr        %Text;         #IMPLIED
  axis        CDATA          #IMPLIED
  headers     IDREFS         #IMPLIED
  scope       %Scope;        #IMPLIED
  rowspan     %Number;       "1"
  colspan     %Number;       "1"
  %cellhalign;
  %cellvalign;
  nowrap      (nowrap)       #IMPLIED
  bgcolor     %Color;        #IMPLIED
  width       %Pixels;       #IMPLIED
  height      %Pixels;       #IMPLIED
  >
```

Appendix F. Character Entities

The following table collects the defined standard, proposed, and several nonstandard, but generally supported, character entities for HTML and XHTML.

Entity names, if defined, appear for their respective characters and can be used in the character-entity sequence &name; to define any character for display by the browser. Otherwise, or alternatively for named characters, use the character's three-digit numeral value in the sequence &#nnn; to specially define a character entity. Actual characters, however, may or may not be displayed by the browser depending on the computer platform and user-selected font for display.

Not all 256 characters in the ISO character set appear in the table. Missing ones are not recognized by the browser as either named or numeric entities.

To be sure that your documents are fully compliant with the HTML 4.0 and XHTML 1.0 standards, use only those named character entities whose conformance column is blank. Characters whose conformance column contains "!!!" are not formally defined by the standards; use them at your own risk.

Numeric Entity	Named Entity	Symbol	Description	Conformance
			Horizontal tab	
		Line feed	
		Carriage return	
			Space	
!		!	Exclamation point	
"	"	"	Quotation mark	
#		#	Hash mark	
$		$	Dollar sign	
%		%	Percent sign	
&	&	&	Ampersand	
'		'	Apostrophe	
(		(	Left parenthesis	
)		)	Right parenthesis	
*		*	Asterisk	
+		+	Plus sign	
,		,	Comma	
-		-	Hyphen	

Numeric Entity	Named Entity	Symbol	Description	Conformance
.		.	Period	
/		/	Slash	
0 - 9		0-9	Digits 0-9	
:		:	Colon	
;		;	Semicolon	
<	<	<	Less than	
=		=	Equals sign	
>	>	>	Greater than	
?		?	Question mark	
@		@	Commercial at sign	
A - Z		A-Z	Letters A-Z	
[		[	Left square bracket	
\		\	Backslash	
]		]	Right square bracket	
^		^	Caret	
_		_	Underscore	
`		`	Grave accent	
a - z		a-z	Letters a-z	
{		{	Left curly brace	
			|	Vertical bar	
}		}	Right curly brace	
~		~	Tilde	
‚		‚	Low left single quote	!!!
ƒ		*ƒ*	Florin	!!!

Numeric Entity	Named Entity	Symbol	Description	Conformance
„		„	Low left double quote	!!!
…		…	Ellipsis	!!!
†		†	Dagger	!!!
‡		‡	Double dagger	!!!
ˆ		^	Circumflex	!!!
‰		‰	Permil	!!!
Š		Š	Capital S, caron	!!!
‹		<	Less than sign	!!!
Œ		Œ	Capital OE ligature	!!!
Ž		Ž	Capital Z, caron	!!!
‘		`	Left single quote	!!!
’		'	Right single quote	!!!
“		"	Left double quote	!!!
”		"	Right double quote	!!!
•		•	Bullet	!!!
–		–	En dash	!!!
—		—	Em dash	!!!
˜		~	Tilde	!!!
™		™	Trademark	!!!
š		š	Small s, caron	!!!
›		>	Greater than sign	!!!
œ		œ	Small oe ligature	!!!
ž		ž	Small z, caron	!!!
Ÿ		Ÿ	Capital Y, umlaut	!!!

Numeric Entity	Named Entity	Symbol	Description	Conformance
			Nonbreaking space	
¡	¡	¡	Inverted exclamation point	
¢	¢	¢	Cent sign	
£	£	£	Pound sign	
¤	¤	¤	General currency sign	
¥	¥	¥	Yen sign	
¦	¦	¦	Broken vertical bar	
§	§	§	Section sign	
¨	¨	¨	Umlaut	
©	©	©	Copyright	
ª	ª	ª	Feminine ordinal	
«	«	«	Left angle quote	
¬	¬	¬	Not sign	
		-	Soft hyphen	
®	®	®	Registered trademark	
¯	¯	¯	Macron accent	
°	°	º	Degree sign	
±	±	±	Plus or minus	
²	²	²	Superscript 2	
³	³	³	Superscript 3	
´	´	´	Acute accent	
µ	µ	µ	Micro sign (Greek mu)	
¶	¶	¶	Paragraph sign	
·	·	·	Middle dot	

Numeric Entity	Named Entity	Symbol	Description	Conformance
¸	¸	¸	Cedilla	
¹	¹	¹	Superscript 1	
º	º	º	Masculine ordinal	
»	»	»	Right angle quote	
¼	¼	¼	Fraction one-fourth	
½	½	½	Fraction one-half	
¾	¾	¾	Fraction three-fourths	
¿	¿	¿	Inverted question mark	
À	À	À	Capital A, grave accent	
Á	Á	Á	Capital A, acute accent	
Â	Â	Â	Capital A, circumflex accent	
Ã	Ã	Ã	Capital A, tilde	
Ä	Ä	Ä	Capital A, umlaut	
Å	Å	Å	Capital A, ring	
Æ	Æ	Æ	Capital AE ligature	
Ç	Ç	Ç	Capital C, cedilla	
È	È	È	Capital E, grave accent	
É	É	É	Capital E, acute accent	
Ê	Ê	Ê	Capital E, circumflex accent	
Ë	Ë	Ë	Capital E, umlaut	
Ì	Ì	Ì	Capital I, grave accent	
Í	Í	Í	Capital I, acute accent	
Î	Î	Î	Capital I, circumflex accent	
Ï	Ï	Ï	Capital I, umlaut	

Numeric Entity	Named Entity	Symbol	Description	Conformance
Ð	Ð	Đ	Capital eth, Icelandic	
Ñ	Ñ	Ñ	Capital N, tilde	
Ò	Ò	Ò	Capital O, grave accent	
Ó	Ó	Ó	Capital O, acute accent	
Ô	Ô	Ô	Capital O, circumflex accent	
Õ	Õ	Õ	Capital O, tilde	
Ö	Ö	Ö	Capital O, umlaut	
×	×	x	Multiply sign	
Ø	Ø	Ø	Capital O, slash	
Ù	Ù	Ù	Capital U, grave accent	
Ú	Ú	Ú	Capital U, acute accent	
Û	Û	Û	Capital U, circumflex accent	
Ü	Ü	Ü	Capital U, umlaut	
Ý	Ý	Ý	Capital Y, acute accent	
Þ	Þ	Þ	Capital thorn, Icelandic	
ß	ß	ß	Small sz ligature, German	
à	à	à	Small a, grave accent	
á	á	á	Small a, acute accent	
â	â	â	Small a, circumflex accent	
ã	ã	ã	Small a, tilde	
ä	ä	ä	Small a, umlaut	
å	å	å	Small a, ring	
æ	æ	æ	Small ae ligature	
ç	ç	ç	Small c, cedilla	

Numeric Entity	Named Entity	Symbol	Description	Conformance
è	è	è	Small e, grave accent	
é	é	é	Small e, acute accent	
ê	ê	ê	Small e, circumflex accent	
ë	ë	ë	Small e, umlaut	
ì	ì	ì	Small i, grave accent	
í	í	í	Small i, acute accent	
î	î	î	Small i, circumflex accent	
ï	ï	ï	Small i, umlaut	
ð	ð	ð	Small eth, Icelandic	
ñ	ñ	ñ	Small n, tilde	
ò	ò	ò	Small o, grave accent	
ó	ó	ó	Small o, acute accent	
ô	ô	ô	Small o, circumflex accent	
õ	õ	õ	Small o, tilde	
ö	ö	ö	Small o, umlaut	
÷	÷	÷	Division sign	
ø	ø	ø	Small o, slash	
ù	ù	ù	Small u, grave accent	
ú	ú	ú	Small u, acute accent	
û	û	û	Small u, circumflex accent	
ü	ü	ü	Small u, umlaut	
ý	ý	ý	Small y, acute accent	
þ	þ	þ	Small thorn, Icelandic	
ÿ	ÿ	ÿ	Small y, umlaut	

Appendix G. Color Names and Values

With the popular browsers and according to the Cascading Style Sheets (CSS) standard, you may prescribe the display color for various elements in your documents. You do so by specifying a color value or a standard name. The user may override these color specifications through their browser preferences.

G.1 Color Values

In all cases, you may set the color value for an HTML element, such as `<body>` text, `<table>` background, and so on, as a six-digit hexadecimal number that represents the red, green, and blue (RGB) components of the color. The first two digits correspond to the red component of the color, the next two are the green component, and the last two are the blue component. A value of 00 corresponds to a component being completely off; the hexadecimal value of FF (decimal 255) corresponds to the component being completely on. Thus, bright red is FF0000, bright green is 00FF00, and bright blue is 0000FF. Other primary colors are mixtures of the components, such as yellow (FFFF00), magenta (FF00FF), and cyan (00FFFF). White (FFFFFF) and black (000000) are also easy to figure out.

You use these values in a tag by replacing the color with the RGB triple, preceded by a pound sign (#). Thus, to make all visited links display as magenta, use this body tag:

```
<body vlink="#FF00FF">
```

G.2 Color Names

Determining the RGB-triple value for other than the simplest colors (you try figuring out esoteric colors like "papaya whip" or "navajo white") is not easy. You can go crazy trying to adjust the RGB triple for a color to get the shade just right, especially when each adjustment requires loading a document into your browser to view the result.

To make life easier, the standards define sixteen standard color names that can be used anywhere a numeric color value can be used. For example, you can make all visited links in the display magenta with the following attribute and value for the body tag:

```
<body vlink="magenta">
```

The color names and RGB values defined in the HTML/XHTML standards are:

aqua (#00FFFF)	gray (#808080)	navy (#000080)	silver (#C0C0C0)
black (#000000)	green (#008000)	olive (#808000)	teal (#008080)
blue (#0000FF)	lime (#00FF00)	purple (#800080)	yellow (#FFFF00)
fuchsia (#FF00FF)	maroon (#800000)	red (#FF0000)	white (#FFFFFF)

The popular browsers go well beyond the standard and support the several hundred color names defined for use in the X Window System. Note that these color names may contain no spaces; also, the word "gray" may be spelled "grey" in any color name.

Those colors marked with an asterisk (*) actually represent a family of colors numbered one through four. Thus, there are actually four variants of blue, named "blue1," "blue2," "blue3," and "blue4," along with plain old "blue." Blue1 is the lightest of the four; blue4 the darkest. The unnumbered color name is the same color as the first; thus, blue and blue1 are identical.

Finally, if all that isn't enough, there are one hundred variants of gray (and grey) numbered 1 through 100. "Gray1" is the darkest, "gray100" is the lightest, and "gray" is very close to "gray75."

The extended color names are:

aliceblue	darkturquoise	lightseagreen	palevioletred*
antiquewhite*	darkviolet	lightskyblue*	papayawhip
aquamarine*	deeppink*	lightslateblue	peachpuff*
azure*	deepskyblue*	lightslategray	peru
beige	dimgray	lightsteelblue*	pink*
bisque*	dodgerblue*	lightyellow*	plum*
black	firebrick*	limegreen	powderblue
blanchedalmond	floralwhite	linen	purple*
blue*	forestgreen	magenta*	red*
blueviolet	gainsboro	maroon*	rosybrown*
brown*	ghostwhite	mediumaquamarine	royalblue*
burlywood*	gold*	mediumblue	saddlebrown
cadetblue*	goldenrod*	mediumorchid*	salmon*
chartreuse*	gray	mediumpurple*	sandybrown
chocolate*	green*	mediumseagreen	seagreen*
coral*	greenyellow	mediumslateblue	seashell*
cornflowerblue	honeydew*	mediumspringgreen	sienna*
cornsilk*	hotpink*	mediumturquoise	skyblue*
cyan*	indianred*	mediumvioletred	slateblue*
darkblue	ivory*	midnightblue	slategray*
darkcyan	khaki*	mintcream	snow*
darkgoldenrod*	lavender	mistyrose*	springgreen*
darkgray	lavenderblush*	moccasin	steelblue*
darkgreen	lawngreen	navajowhite*	tan*

darkkhaki	lemonchiffon*	navy	thistle*
darkmagenta	lightblue*	navyblue	tomato*
darkolivegreen*	lightcoral	oldlace	turquoise*
darkorange*	lightcyan*	olivedrab*	violet
darkorchid*	lightgoldenrod*	orange*	violetred*
darkred	lightgoldenrodyellow	orangered*	wheat*
darksalmon	lightgray	orchid*	white
darkseagreen*	lightgreen	palegoldenrod	whitesmoke
darkslateblue	lightpink*	palegreen*	yellow*
darkslategray*	lightsalmon*	paleturquoise*	yellowgreen

G.3 The Standard Color Map

Supporting hundreds of color names and millions of RGB triples is nice, but the reality is that a large (albeit shrinking) population of users can display only 256 colors on their system. When confronted with a color not defined in this set of 256, the browser has two choices: convert the color to one of the existing colors, or dither the color using the available colors in the color map.

Conversion is easy; the color is compared to all the other colors in the color map and is replaced by the closest color found. Dithering is more difficult. Using two or more colors in the color map, the errant color is approximated by mixing different ratios of the available colors. Viewed up close, you'll see a pattern of alternating pixels using the available colors. At a distance, the pixels blend to form a color close to the original color.

In general, your images will look best if you can avoid both conversion and dithering. Conversion will make your colors appear "off"; dithering makes them look fuzzy. How to avoid these problems? Easy: use colors in the standard color map when creating your images.

The standard color map actually has 216 values in it. There are six variants of red, six of green, and six of blue that are combined in all possible ways to create these 216 (6 x 6 x 6) colors. These variants have decimal brightness values of 0, 51, 102, 153, 204, and 255, corresponding to hexadecimal values of 00, 33, 66, 99, CC, and FF. Colors like 003333 (dark cyan) and 999999 (medium gray) exist directly in the color map and won't be converted or dithered.

Keep in mind that many of the extended color names are not in the standard color map and will be converted or dithered to a (hopefully) similar color. Using color names, while convenient, does not guarantee that the desired color will be used by the browser.

When creating images, try to use colors in the standard color map. When selecting colors for text, links, or backgrounds, make sure you select colors in the standard color map. Your pages will look better and will be more consistent when viewed with different browsers.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal featured on the cover of *HTML & XHTML: The Definitive Guide, Fourth Edition* is a koala. The koala is an Australian marsupial, the only member of the *Phascolarctidae* family.

Koalas are tiny, approximately one-half of a gram, when they are born. Young koalas stay in their mother's pouch for approximately seven months. Unlike most marsupials, the koala's pouch opens near the rear, not near the head. Koalas have a high mortality rate and face extinction in Australia due to epidemics in 1887–1889 and 1900–1903 and unrestrained hunting throughout the 20th century. They are a protected species and populations are rebuilding, but at present they survive only in eastern Australia.

Colleen Gorman was the production editor and proofreader, and Emily Quill was the copyeditor for *HTML & XHTML: The Definitive Guide, Fourth Edition*. Madeleine Newell and Melanie Wang provided quality control. Matt Hutchinson and Nancy Williams provided production support. Seth Maislin and Ellen Troutman-Zaig wrote the index. Edie Freedman designed the cover of this book, using a 19th-century engraving from the Dover Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font. Alicia Cech and David Futato designed the interior layout based on a series design by Nancy Priest. Mike Sierra implemented the design in FrameMaker 5.5.6. The text and heading fonts are ITC Garamond Light and Garamond Book. The illustrations that appear in the book were produced by Robert Romano and Rhon Porter using Macromedia FreeHand 8 and Adobe Photoshop 5. This colophon was written by Clairemarie Fisher O'Leary.

Article - XHTML: Bridging HTML & XML

It is only fitting that HTML, the language that drives the web, is in a constant state of flux. Just as the web is continually growing, changing, and maturing, HTML continues to evolve. Even with the best of intentions from standards groups and browser manufacturers, it seems HTML cannot stand still.

The early years of the web were wild and woolly, with new tags being defined by every new browser version that came along. Standards were de facto only, and early attempts to create an all-encompassing HTML standard collapsed under the sheer weight of all the features under consideration.

HTML 3.2 brought a bit of sanity to the web author's world, bringing together the most common tags and their attributes. The general, if grudging, acceptance of HTML 3.2 led to the creation of HTML 4.0, the first good, clean, well-supported HTML standard. Not only does HTML 4.0 tell you which tags are correct, it also tells you which tags are on their way out, so you can plan accordingly.

Only a year ago, the HTML world had settled comfortably into version 4.0. However, just minutes after the World Wide Web Consortium (http://www.w3.org/) put a bow on HTML 4.0, it decided that more modularity, more flexibility, and broader capabilities were needed. So, although HTML 4.0 may be the end of one long road to standardization, it has launched a new journey for web authors that begins with XHTML 1.0.

When HTML was originally conceived, no one had any idea it would be so successful or be asked to handle so many kinds of documents, browsers, and media. While it has beared up admirably under the demands of web users, HTML 4.0 has stretched as far as it can to accommodate new technology. While HTML 4.0 is petering out, XHTML 1.0 stands ready to step in, designed to handle almost anything web authors can dream up.

While HTML is a static markup language with a fixed set of tags, XHTML is just one set of tags defined with XML, the Extensible Markup Language. Using XML, you can define tags to represent almost any kind of data or document, not just what the HTML designers had in mind when they crafted HTML. Need to capture musical notation? Chemical formulae? Integrated circuits? XML can handle all this and let you integrate your new markup tags with XHTML, creating hybrid documents that will still work with existing browsers. By adding XSL (Extensible Style Sheets), you can even teach browsers how to display all your new tags.

XML is a platform for growth and expansion of the web. XHTML is a reformulation of HTML 4.0 using XML, allowing XHTML to be integrated with all the new tools that will be XML-based. Unfortunately, fluency in HTML does not guarantee your expertise in XHTML. While XHTML made every effort to be identical to HTML, there are enough differences to cause headaches for the unsuspecting web author. The only way to stay current, capable, and fluent in XHTML is to stay atop the latest HTML 4.01 standard and augment that knowledge with all the details of XHTML. And that's exactly why we wrote the fourth edition of *HTML & XHTML: The Definitive Guide*.