P A R T 1

Technology

CSS is emerging as the true language of web design. Being a great web designer means being fluent in the language, especially as we embark on a time where web browsers can more effectively bring to life the power and elegance of CSS. With CSS, we can bridge the gap between science and art. To create a sophisticated web page means to understand the visual and also to understand how the visual is created through the technology.

To learn and use CSS effectively, you must have an understanding of the underlying technical structure of both web markup and CSS itself. To that end, the following chapters focus on demonstrating structured markup, how CSS integrates with markup, and how CSS works *as a language* to enable you to express innovative and progressive ideas through your work.

COPYRIGHTED MATERIAL

One cannot help but be in awe when he contemplates the mysteries of eternity, of life, of the marvelous structure of reality.

—Albert Einstein

CHAPTER one

Understanding Structured Markup

Just as a schooled artist must study anatomy, so must a web designer study the structures that are the infrastructure of a design. It is imperative for web designers who want to write great CSS to understand how web markup has evolved, grown, and changed. It's also critical to have a detailed understanding of how markup works if you want to achieve excellence when writing CSS.

To prepare for that excellence, you'll learn about the historical emergence of markup and the current need to return to a structure temporarily lost in the frenzy of web design and development in the late '90s. The web design profession has matured, so best practices and standard methods are emerging to help us create very clean, very powerful means of delivering great sites for use today and well into the future.

In this chapter you will learn:

- The significance of standards
- The meaning of structure
- Ideas central to markup
- In-depth information about HTML and XHTML
- How HTML and XHTML relate to CSS
- How to create structured HTML and XHTML documents

What Are Web Standards?

Chances are you fit into one of several categories when it comes to authoring documents. You might be a person who uses a visual editor such as Microsoft FrontPage, Dreamweaver MX, or Adobe GoLive exclusively, letting the visual environment guide the markup. Or, you might be specifically a markup author, pounding out your tags and attributes in a text editor such as Emacs, vi, Notepad, or SimpleText. Maybe you like to write your own documents but prefer working with an HTML-style editor such as HomeSite or BBEdit.

As with more and more web professionals, a hybrid methodology may exist for you: you might work using a visual editor and do some hand-authoring, too.

No matter your method, you probably know something about HTML, even if you've learned it by viewing source or reading books and have never studied it formally. And, while you might have heard the term *web standards*, it's not a very clear term and may have confused you as to what it means and why it's important.

The term *web standard* is a confusing one not just because it's inaccurate, but because it also suggests conformity. I use the term to refer to what is actually *a series of specifications and recommendations created by the World Wide Web Consortium* (W3C). The W3C is not an authoritative standards body per se. They're not going to drop by your office and give you a ticket for noncompliance. The primary functions of the W3C are to research, develop, and publish information on technologies and activities related to the Web (see Table 1.1).

Table 1.1: A Sample of Technologies and Activities of the W3C

Technology or Activity	Purpose
Accessibility	To ensure documents are accessible to all people
Cascading Style Sheets (CSS)	To provide a style language for presentation of documents
Document Object Model (DOM)	To provide consistent object models within browsers
Hypertext Markup Language (HTML)	To author web documents
Internationalization	To facilitate proper encoding and display of multilingual and international documents
Synchronized Multimedia Integration Language (SMIL)	An XML-based language to facilitate multimedia synchronization: video, audio, text
Scalable Vector Graphics (SVG)	An XML-based language to facilitate scalable vector-based graphics and animation
Extensible Hypertext Markup Language (XHTML)	To author documents for the web and alternative devices
Extensible Markup Language (XML)	To provide a universal format for structured documents and data

For more specific information related to these technologies, as well as descriptions of the many other issues the W3C concerns itself with, visit www.w3.org/.

When you design your website, no matter which method you use, you are working in part with W3C technologies. And, while the W3C is not an authoritative organization that you *must* follow, it certainly provides a base from which designers can learn consistent and intelligent practices.

Following a W3C specification—a *web standard*—does *not* mean that design options are limited. I suggest the opposite. Innovation requires intermittent periods of stability and chaos. Currently, we are in a transitional time as web professionals. We're trying to make sense out of the chaos of the past years and achieve some stability in our designs, tools, and methods. It makes sense: we're looking to find some conventions to make our jobs easier.

Building the next level of the Web's infrastructure as cleanly as possible provides the matrix for new levels of innovation. I think people are sensing this, and this is why many people are starting to get interested in standards. Web professionals are starting to realize it's a cool—and necessary—thing to pay attention to in order to keep the technology moving forward in a mature way.

A standards-based site also does not suggest that the site will be unattractive, use text-only, or have few visual elements. In fact, this book exists in part to prove that standards-based sites can not only be attractive, but also downright innovative.

Back to the Future

Most readers are aware of HTML, and many are also aware of its successor in web markup, XHTML.

But where did these languages come from, and how come we're going through such a radical shift in the way we as designers and developers work?

Let's begin with the Standardized General Markup Language (SGML). It is what is known as a *meta-language*. SGML is essentially a massive *document type definition* (DTD) used to create other markup languages.

NOTE DTDs are plain-text documents that describe the elements, attributes, and other allowed components of a given markup language such as HTML or XHTML, along with its version and type, such as HTML 4.01 Strict. You'll read more about these distinctions and how they influence your design options later in this chapter.

HTML is derived from SGML. The early use of HTML led to some great and innovative approaches, bending the language to allow for increasingly more complex visual sites. The best—and most problematic—example of this is HTML tables, which were originally added to HTML to manage data tables more effectively than the preformatted text element `pre`. By turning off table borders, suddenly a de facto grid system for laying out even the most popular site content was born, as you can see in Figure 1.1. As the example shows, the primary site design method became tables.

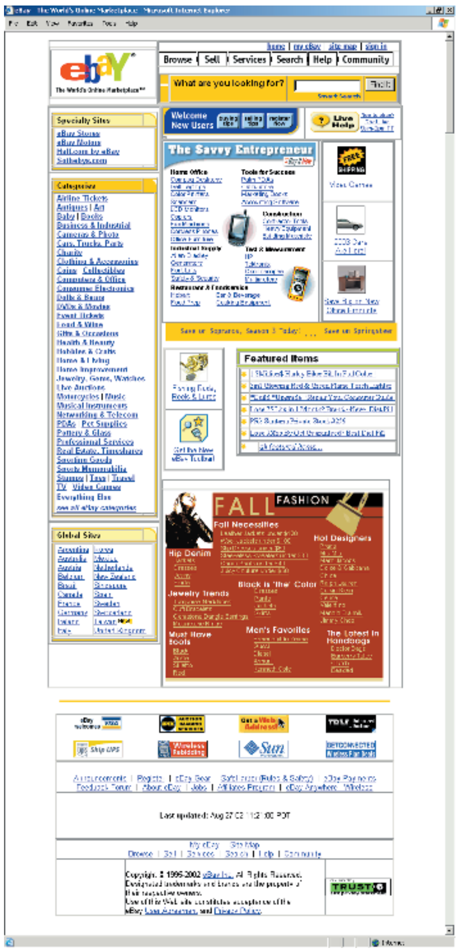


Figure 1.1: eBay's Home Page with all table borders turned on

As the examples show, the primary site design method became tables. But tables were problematic for several reasons, including:

- Tables, especially when particularly complex or deeply nested, create overhead in terms of size and speed.
- Complex tables can be difficult to manage, especially across web teams.
- Tables of this nature are often inaccessible to those with visual or mobility-related impairments.

Other problems came to the forefront as various browsers introduced proprietary markup not related to work being done at the W3C. The so-called "browser wars" began in the early 1990s and still exist to a large degree. Browser manufacturers sometimes let corporate agendas get in the way of the greater good. The irony is that the browser manufacturers were (and are now still) members of working groups for standards and yet continue to only slowly implement W3C recommendations while aggressively trying out new technologies in an effort to dominate the browser space.

Not all aspects of the browser wars was bad. In fact, early on, this competition allowed for a time of great innovation. Everyone broke rules: designers, developers, browser and tools vendors. It was an important exercise. However, now the time has come to clean up our practices if we want to move forward in the expanding potential of the web and create a strong, next-level infrastructure.

While these difficult issues in HTML and browser support were being examined, Extensible Markup Language (XML) was coming to light. XML, also derived from SGML, is a more streamlined meta-language that is especially useful for sharing documents and information on the Internet and related networks.

HTML was re-evaluated in the context of XML—with its emphasis on rigor, conformance, and extensibility. This evaluation resulted in the publication of a new markup language, XHTML 1. Introduced in January 2000, XHTML 1 officially replaced HTML 4.01 as the most contemporary available specification. XHTML is now in its 1.1 version, with version 2 on the horizon.

NOTE Just because a specification is currently recommended does not necessarily mean it is the specification with which you must work. In other words, it’s perfectly acceptable to be writing HTML 4.01 instead of XHTML 1.1. The goal is to know the specifications well enough to be able to make decisions about which method will work best for your circumstances.

XHTML is considered by the W3C as a *reformulation of HTML as an XML application*. XHTML at its most basic is HTML vocabulary in a well-structured XML document. At its most complex, XHTML is extensible and customizable. You’ll read more about this and see examples as you read further in the chapter.

But Why Standards?

So why should you follow standards? A lot of people say, “Hey, I can use nonstandard markup that works just fine.”

There are several reasons why understanding and following standards makes sense. Here are a few:

- You will save time. If your documents follow standards, you achieve a level of efficient work practices. Troubleshooting becomes easier because of document consistency. Team members will work more efficiently in an environment where documents follow structure and logic.
- Saving time means saving money. If you are able to save time by ensuring that your documents are standards compliant, stable, and use CSS for style, you will be able to both profit from the process *and* pass the resulting savings on to your clients.
- You’ll reduce complicated pages so browsers will interpret and display a page quickly and accessibility concerns will be addressed. This means a happier end user.
- You’ll have better job opportunities. If you are still creating web pages in a visual editor without understanding the underlying markup and have not spent any time studying standards, you are restricting yourself in terms of advancement within the profession.

- You will become part of the solution, not the problem, as the infrastructure of the web becomes increasingly more complex.
- Standards set the stage for extending content beyond the limits of the Web to wireless devices such as smart phones, pagers, and PDAs; alternative devices such as MSNTV (formerly WebTV); and a range of devices yet to come.

To sum up, early case studies suggest that compliance probably saves money for everyone in the website food chain—from site owner to developer to ISP. Those are the immediate advantages. Longer term, working with standards addresses many technical, creative, and even social concerns. Technically, websites will be more easily maintained and also readily available for many platforms beyond the Web. Creatively, you can apply style sheets that will easily make a site look good on a computer screen, on a PDA screen, even in print. Socially, you remove barriers to access by cleaning up your hacked markup and paying attention to accessibility concerns.

Making a Case for Standards: The Web Standards Project

One of the difficulties of becoming familiar with standards is that the W3C tends to focus on the development and creation of technologies rather than the dissemination of educational material related to their work.

The documents at the W3C are often prohibitive for busy people who don't want to read through the minutia—much less translate clumsy academic writing into useful guidance. The need for more interesting documentation is currently being addressed by the W3C, which has several committees that are working to make their findings more readily understandable and available to the public at large.

In a desire to see the long-term benefits of standards be implemented by browsers, software developers, and designers and developers themselves, a grassroots, volunteer-driven organization called The Web Standards Project emerged. Members of The WaSP (as most people refer to it) work to evangelize standards and educate others about them. The group has recently expanded to include educational initiatives, and its website is an example of an attractive, usable site that adheres to W3C standards and professionally acquired best practices.

The WaSP, however, is a truly independent organization. While understanding the long-term benefits that W3C work does, WaSP members have (and will continue if necessary) to openly criticized certain activities of the W3C. Part of The WaSP mission is to encourage the W3C to live up to its potential.

For more information about The WaSP and its activities, see `www.webstandards.org/`.

Exploring HTML Concepts

By the time HTML 4 emerged in 1998 as the recommended specification for web markup, a rigorous evaluation had been given to the state of affairs occurring with browsers and language. With HTML 4 came several directives that have shaped the languages that have since evolved and the practices related to those languages. The most critical concerns voiced when HTML 4 was introduced include the following:

- The need to separate document structure and style in order to return documents to a more accessible, cross-platform state. As CSS Level 1 had been completed in 1996, this was a means of encouraging designers and developers to use CSS.
- A desire to improve document rendering.
- A highly motivated need to encourage the authoring of accessible documents.
- The need to encourage web professionals, web development tools manufacturers, and browser developers to adhere to a common base of guidelines.

These critical concepts have followed through to the final version of HTML—HTML 4.01—and beyond, into XHTML.

NOTE HTML 4.01 contains only minor editorial changes from HTML 4. It is canonically important, however, because the HTML 4.01 DTDs were used as the basis for XHTML 1. As this discussion progresses, you'll see examples of HTML 4.01. Most people authoring to standards (and using HTML) use an HTML 4.01 DTD.

The following sections will provide deeper insight into the concerns that HTML 4 raised; this will help you gain a more profound appreciation for how these ideas have influenced the growing interest in using CSS.

Separation of Document Structure and Style

One of HTML 4's prime directives is to separate document formatting from the presentation of that document. This is a critical issue, because most of the HTML in use today is misused—it's filled with errors that provide endless problems and make accessibility, document rendering speed, and cross-browser consistency nightmarish to achieve.

What Is Document Structure?

For the purposes of this book, document structure for the purposes of this book is the skeletal structure of a Web document. This skeletal structure includes:

- A `DOCTYPE` declaration
- An `html` element
- A `head` with `title`
- A document `body`
- Structural elements only, used in a logical manner for managing content. In this category you'll find such things as headings (`h1`, `h2`, `h3`, `h4`, `h5`, `h6`), paragraphs and breaks (`p`, `br`), and lists (`ul`, `ol`, `dl`).

NOTE A `DOCTYPE` declaration defines the document type and DTD version. `DOCTYPE` declarations appear at the top of a structured document. These declarations are described in detail in the "Learning About Document Type Definitions" section later in this chapter.

A structured document results in a tree. Listing 1.1 shows a valid HTML 4 document with the required structural components and some content marked up with basic elements.

Listing 1.1: An HTML 4 Document with Basic Structure

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
        "http://www.w3.org/TR/REC-html40/strict.dtd">

<html>
<head>

<title>Working with Structure</title>
```

```
</head>

<body>

<h1>Welcome</h1>

<p>Welcome to the site where structure matters!</p>

<h2>Getting Into Structure</h2>

<p>In order to begin working with structure, you need to be aware of:</p>

<ul>
<li>DOCTYPE declarations</li>
<li>Properly structured head and body components</li>
<li>Logical structures for content markup</li>
</ul>

</body>
</html>
```

Figure 1.2 demonstrates the document tree that results from Listing 1.1. Document trees become especially important when working with CSS because of the concept of inheritance—which means style features defined for an element will pass to its children—and how elements are related to one another, influencing the way your style sheets will be interpreted.

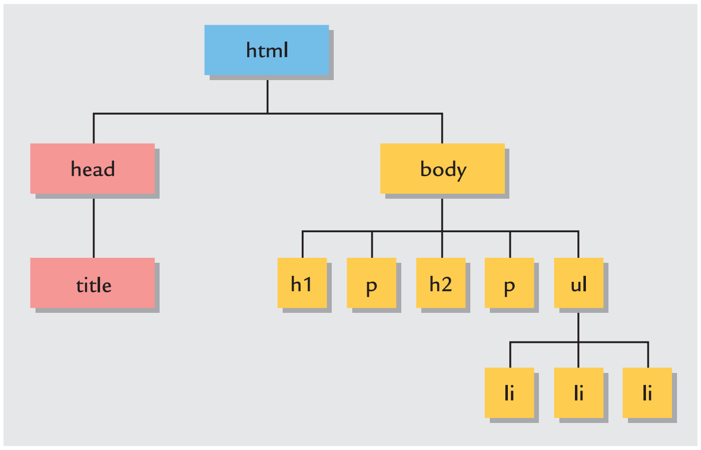


Figure 1.2: Document tree with parent, child, and leaf nodes

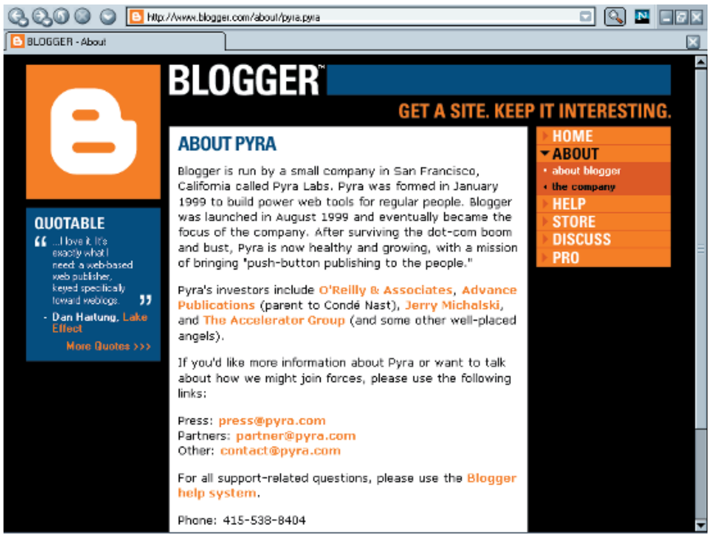
Documents are broken down into a document root (typically the `html` element) and any child relationships (the `head` and `body` are children to the root, and the body has four children, `h1`, `h2`, `p`, and `ul`). Finally, there are the leaf nodes: the three bullet items denoted by the `li` element. Each `li` element is, of course, a child to the `ul` element.

NOTE The concept of inheritance is explored in depth in Chapter 2, "Learning CSS Theory."

Document Presentation

Presentation mostly refers to anything that involves visual details. Examples of presentation include the following:

- Color (see Figure 1.3)
- Text formatting and typographic design (see Figure 1.4)
- Background graphics (see Figure 1.5)
- Borders, padding, spacing (see Figure 1.6)
- Layout of pages (see Figure 1.7)



QUOTABLE

“ ...I love it. It's exactly what I need: a web-based web publisher, keyed specifically toward weblogs. ”

- Dan Hartung, Lake Effect

ABOUT PYRA

Blogger is run by a small company in San Francisco, California called Pyra Labs. Pyra was formed in January 1999 to build power web tools for regular people. Blogger was launched in August 1999 and eventually became the focus of the company. After surviving the dot-com boom and bust, Pyra is now healthy and growing, with a mission of bringing "push-button publishing to the people."

Pyra's investors include O'Reilly & Associates, Advance Publications (parent to Condé Nast), Jerry Michalski, and The Accelerator Group (and some other well-placed angels).

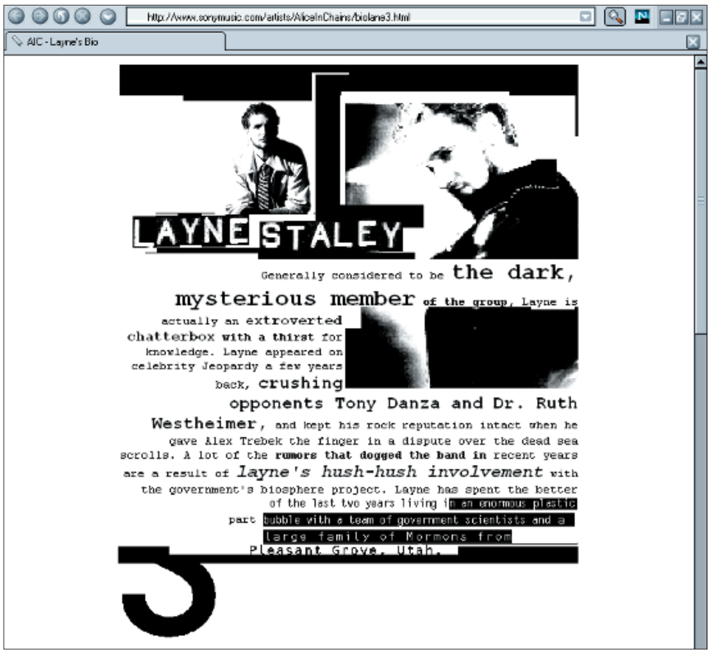
If you'd like more information about Pyra or want to talk about how we might join forces, please use the following links:

Press: press@pyra.com
Partners: partner@pyra.com
Other: contact@pyra.com

For all support-related questions, please use the Blogger help system.

Phone: 415-538-8404

Figure 1.3: The colors in this page are all generated using HTML color attributes.


http://www.sonymusic.com/artists/AliceInChains/biolane3.html
AIC - Layne's Bio
LAYNE STALEY

Generally considered to be **the dark, mysterious member** of the group, Layne is actually an extroverted chatterbox **with a thirst for** knowledge. Layne appeared on celebrity Jeopardy a few years back, crushing opponents Tony Danza and Dr. Ruth **Westheimer**, and kept his rock reputation intact when he gave Alex Trebek the finger in a dispute over the dead sea scrolls. A lot of the **rumors that dogged the band in** recent years are a result of *Layne's hush-hush involvement* with the government's biosphere project. Layne has spent the better part of the last two years living in an enormous plastic bubble with a team of government scientists and a large family of Mormons from Pleasant Grove, Utah.

Figure 1.4: The range of fonts, colors, and sizes in this image are generated by HTML.

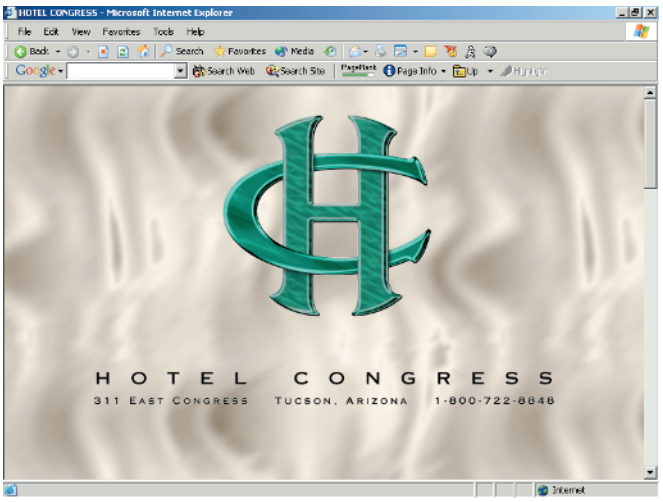
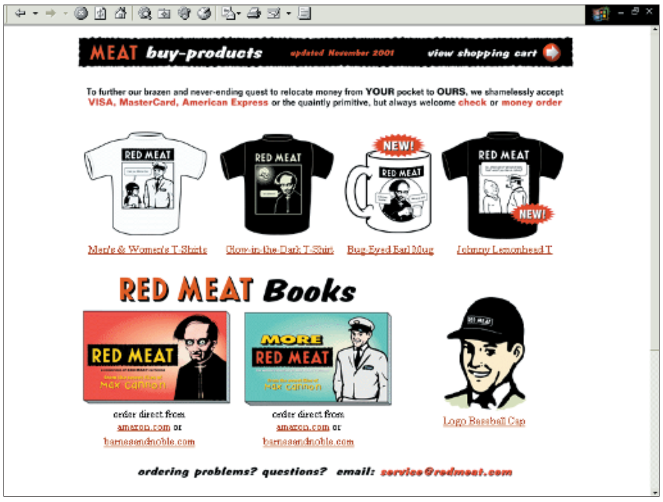

HOTEL CONGRESS - Microsoft Internet Explorer
HOTEL CONGRESS
311 EAST CONGRESS TUCSON, ARIZONA 1-800-722-8848

Figure 1.5: This early web page still looks great because of its background.



To further our brazen and never-ending quest to relocate money from YOUR pocket to OURS, we shamelessly accept VISA, MasterCard, American Express or the quaintly primitive, but always welcome check or money order

Figure 1.6: HTML defines the borders, padding, and spacing of page elements.

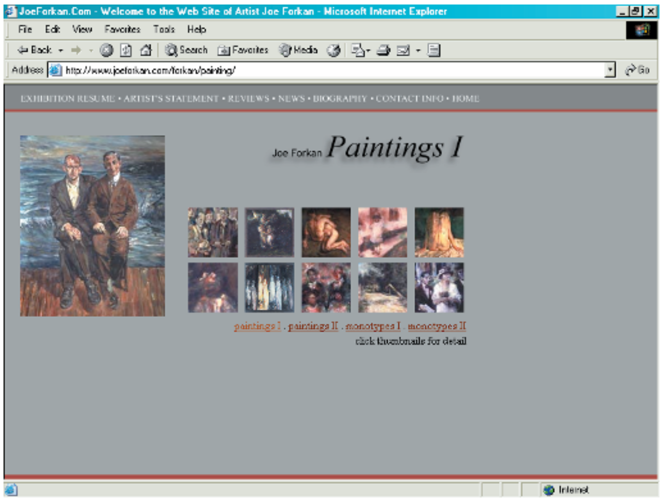

JoeForkan.Com - Welcome to the Web Site of Artist Joe Forkan - Microsoft Internet Explorer
http://www.joeforkan.com/forkan/painting/
EXHIBITION RESUME • ARTIST'S STATEMENT • REVIEWS • NEWS • BIOGRAPHY • CONTACT INFO • HOME
Joe Forkan Paintings I
paintings I . paintings II . monotypes I . monotypes II
click thumbnails for detail

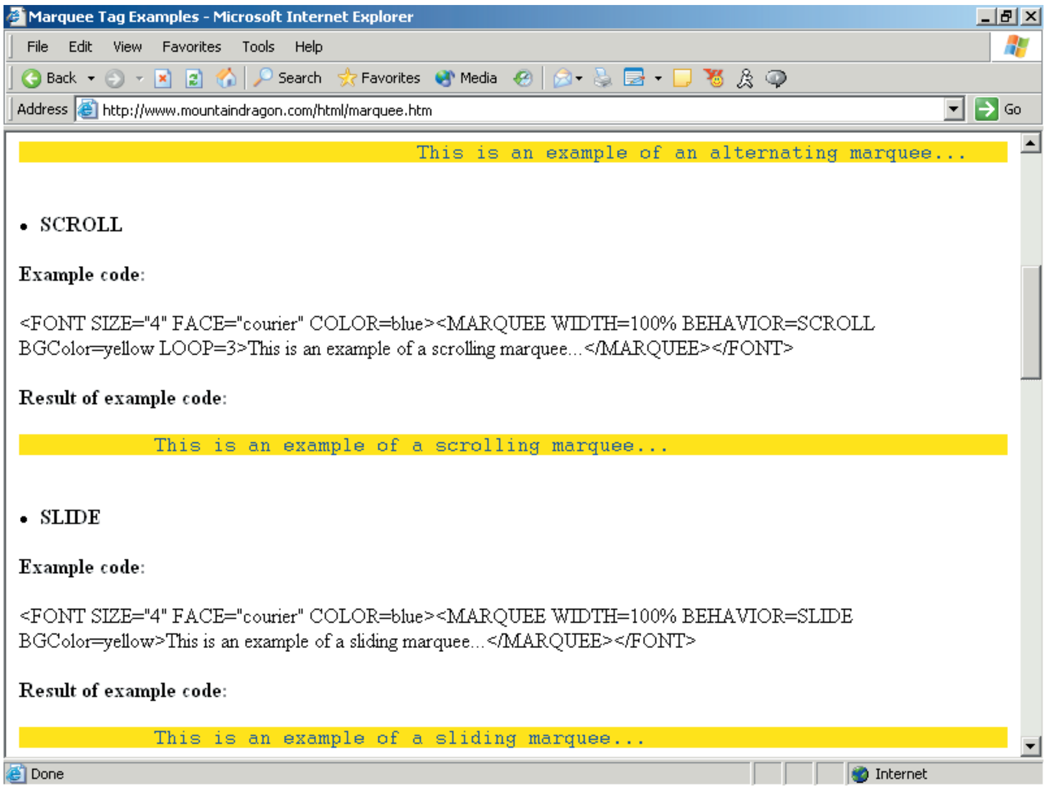
Figure 1.7: This site uses tables rather than CSS for layout, a common current practice.

HTML 4 was the first HTML version to formally recommend that in its most idealistic expression, authors should leave the HTML document empty of presentational detail and address presentational concerns using—you guessed it—Cascading Style Sheets (CSS).

NOTE One of the greatest features of CSS is that their presentation methods go beyond the screen. You can prepare style sheets so documents can be styled for many types of media, including print, aural devices, PDAs, and cell phones.

Improving Accessibility and Document Rendering

Originally, HTML was designed to be a language that could be easily and readily distributed across various platforms and read by anyone, regardless of their software. But innovation and competition within the browser sector quickly changed that reality. With Microsoft and Netscape Communications Corp. rushing hither and yon to create the coolest technology on the block, the consistency of HTML's semantic structure was disrupted when new, browser-based tags and attributes emerged—many unsupported by competing browsers. Most of these tags and attributes had to do with presentation or visual design, including Netscape's renowned `blink` tag and the dreaded Microsoft IE `marquee` tag (Figure 1.8).


Marquee Tag Examples - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.mountaindragon.com/html/marquee.htm
Go
Done
Internet

This is an example of an alternating marquee...

- SCROLL

Example code:

```
<FONT SIZE="4" FACE="courier" COLOR=blue><MARQUEE WIDTH=100% BEHAVIOR=SCROLL
BGColor=yellow LOOP=3>This is an example of a scrolling marquee...</MARQUEE></FONT>
```

Result of example code:

This is an example of a scrolling marquee...

- SLIDE

Example code:

```
<FONT SIZE="4" FACE="courier" COLOR=blue><MARQUEE WIDTH=100% BEHAVIOR=SLIDE
BGColor=yellow>This is an example of a sliding marquee...</MARQUEE></FONT>
```

Result of example code:

This is an example of a sliding marquee...

Figure 1.8: IE's `marquee` *tag—not only proprietary, but annoying, too.*

One of HTML 4's goals was to bring organization back to HTML. A related goal was to improve accessibility of documents. By using some new element and attribute options within HTML 4, document authors and page designers could now help individuals understand and negotiate pages no matter what their platform—or physical abilities.

Many people with impaired or no vision have tremendous difficulty accessing today's World Wide Web. This is largely because electronic screen readers that browse the screen and read the content aloud are significantly more challenged by complex graphical pages. However, with a little forethought, authors can make this process much easier. Individuals with other physical limitations are also assisted by devices—and whether it's a screen reader or special keyboard, the methodologies that HTML 4 proposed to aid access have begun to play a growing role in accessibility and improved rendering of documents.

Accessibility is becoming a hot topic due to recent legislation worldwide that certain kinds of sites must be made accessible to all people, including those with disabilities. In the United States, accessibility has become especially important for federal agency websites as well as those sites created by anyone receiving federal contracts to fund their sites. This is due to legislation regarding accessibility in the U.S.; specifically, it relates to a portion of legislation known as Section 508.

NOTE To learn more about Section 508 and what it details, see `www.section508.gov/`.

The rendering of documents in a web browser can be improved by adhering to common practices. At its most strict, HTML 4 suggests that the author leave tables behind as a means of presenting layout and instead use Cascading Style Sheets for the positioning of objects on a page. The use of CSS in this way not only improves the rendering of documents within supporting browsers, but it also helps you address accessibility concerns by ensuring that it is the structured document and its content, not the presentation of that content, that gets distributed to those accessing the pages using assistive methods.

The Web Accessibility Initiative (WAI)

The Web Accessibility Initiative (WAI) of the W3C is dedicated to promoting awareness worldwide about accessibility. The WAI provides the following:

- Accessibility guidelines and tips
- Accessibility tests
- Accessibility validation options
- News on accessibility activities

To learn more about the WAI, see `www.w3.org/WAI/`.

Going Global

Just as access is important to those with disabilities, it is equally important to ensure that multilingual, international, and localized sites can be developed.

To accommodate access for international users, the W3C makes a concerted effort in HTML 4 and beyond to address international issues for authors, user agents, and development tools.

Current areas of activity include the following:

- Increasing awareness among web and browser developers regarding international access issues
- Stressing the importance of Unicode as a mechanism for character encoding
- Creating study groups within the W3C to look at details of international concerns

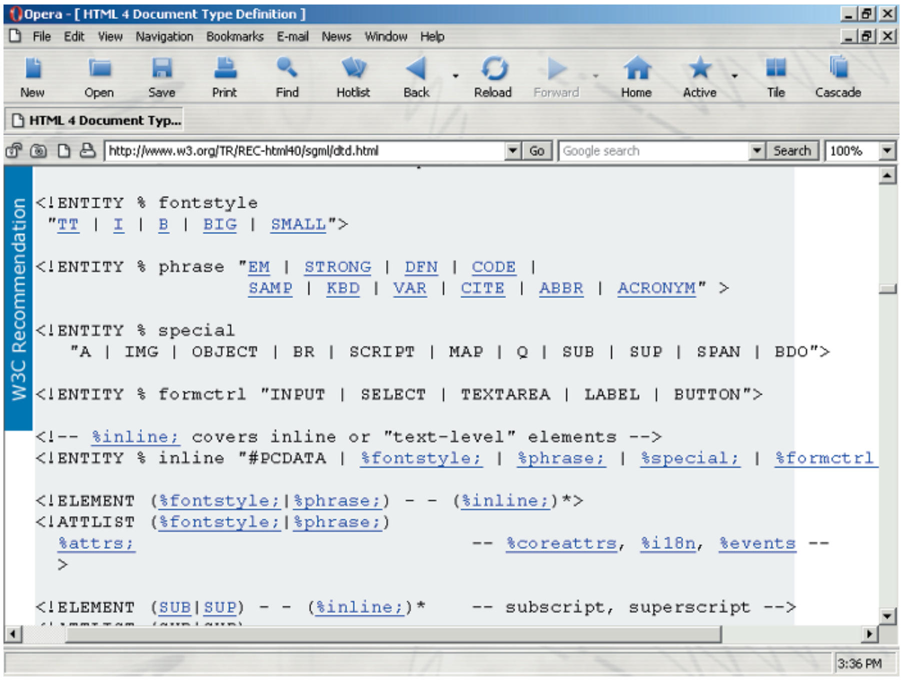
The importance of internationalization is easily seen with a visit to any multi-lingual website. If you'd like to see a variety of international sites, check out the BBC's international websites, each written and displayed in a different language, `www.bbc.co.uk/`.

NOTE For more information on internationalization concerns, visit `www.w3.org/International/`.

Learning About Document Type Definitions

Another piece of the puzzle that HTML 4 provided was a means by which web designers could move away from the haphazard practices of yesterday and embrace the longer-term ideals of standards. This came about with the creation of three distinct DTDs within HTML 4. Each of these definitions contains specific information about which elements and attributes are available in HTML and how they could be used, depending upon the goals of the designer.

A DTD is a plain-text document that contains rules about the way a given language works. In the case of HTML, the DTDs are publicly available at the W3C. Figure 1.9 shows a portion of a W3C DTD viewed within a web browser.


Opera - [HTML 4 Document Type Definition]
File Edit View Navigation Bookmarks E-mail News Window Help
New Open Save Print Find Hotlist Back Reload Forward Home Active Tile Cascade
HTML 4 Document Typ...
http://www.w3.org/TR/REC-html40/sgml/dtd.html
Go
Google search
Search
100%
W3C Recommendation
3:36 PM

```
<!ENTITY % fontstyle
 "TT | I | B | BIG | SMALL">

<!ENTITY % phrase "EM | STRONG | DFN | CODE |
                   SAMP | KBD | VAR | CITE | ABBR | ACRONYM" >

<!ENTITY % special
   "A | IMG | OBJECT | BR | SCRIPT | MAP | Q | SUB | SUP | SPAN | BDO">

<!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 -->
```

Figure 1.9: A portion of the HTML 4.01 Transitional DTD as seen in the Opera browser

The Three Flavors of DTDs in HTML 4.0/4.01

Three types of DTDs, referred to as *interpretations*, were first defined in HTML 4 and later influenced HTML 4.01 and XHTML 1. These DTD interpretations are as follows:

Strict The Strict DTD is the most optimistic of the three, with almost all presentational elements and attributes completely unavailable to use if you want a valid document. (See the next section, "Compliance and Validation.") The recommended means of addressing presentation at this level is, of course, CSS.

Transitional Also referred to as "loose," this DTD was developed with the understanding that CSS was not then (and is still not now) completely available in all cases. The Transitional DTD allows for simple presentational elements and attributes, including even those that had been deprecated in favor of a better technology, such as the `font` element (note that this is even true in the XHTML 1 Transitional DTD). Ideally, document authors choosing a Transitional DTD understand that they are working toward the goals the Strict DTD embodies. To that end, while you *can* use deprecated elements in a Transitional DTD, you are not particularly *encouraged* to do so.

Frameset The Frameset DTD is specific to the creation of framesets, which as most readers are aware, is a different type of document from a conventional HTML page. Framesets can be considered the control documents that restructure a browser's interface and give the designer control over how to do that. The Frameset DTD only includes information for frameset documents, so the only time you will use a Frameset DTD is when you are creating a frameset. Conventional pages within your frames can be marked up using any HTML language, version, and DTD.

No doubt you're wondering how DTDs influence the way a document is read by browsers. The truth is that only recently did the inclusion of a `DOCTYPE` citing a specific DTD influence the way a given browser interpreted a document. (See the "DOCTYPE Switching" sidebar later in this chapter.) For the most part, if you include something nonstandard in a document, the browser's going to display it if it knows the markup you're using. Following a DTD means following the language's rules, and this relates mostly to the creation of *compliant* and *valid* documents.

Compliance and Validation

Two key concepts when working with web standards include *compliance* and *validation*:

- A compliant document is one that conforms to the DTD that is referenced by its `DOCTYPE`.
- A valid document is one that is tested for compliance using a validator such as that provided by the W3C, `http://validator.w3.org/`.

NOTE With any application, bugs exist. So it is with validators, too. For a helpful guide on commonly encountered validation problems, see "Liberty! Equality! Validity!" at `http://devedge.netscape.com/viewsource/2001/validate/`.

There have been some arguments about the usefulness of validation. I consider validation to be an important process when working seriously with HTML, XHTML, and CSS. Validation aids in education by providing warnings and errors regarding document compliance.

The only consideration is that validation errors can be as confusing as grammar rules in a Microsoft Word document! You have to learn a bit of process when it comes to troubleshooting.

NOTE You'll step through the validation process with HTML and XHTML in this chapter, and you will learn to validate CSS later in the book. Interestingly, the W3C validator is the most used service of the W3C.

Comparing DTDs in HTML 4

A good way to learn a bit about the way DTDs work is to compare portions of them with each other. In this case, first examine the DTD portion in Listing 1.2.

Listing 1.2: Excerpt from HTML 4.01 Strict DTD—Paragraphs

```
<!--========= Paragraphs =========-->

<!ELEMENT P - O (%inline;)* -- paragraph -->
<!ATTLIST P
%attrs;                    -- %coreattrs, %i18n, %events --
>
```

Notice how this DTD portion shows how paragraphs are handled in HTML 4.01 Strict. The only attributes allowed are *core* attributes (`%coreattrs`) needed for style, scripting, and accessibility (`id`, `class`, `style`, `title`); *internationalization* attributes (`%i18n`) required for internationalization (`lang`, `dir`); and *event* attributes (`%events`), used for scripting as well (onclick, `onmouseup`, etc.).

Now examine Listing 1.3, which describes how paragraphs are handled in HTML 4.01 Transitional.

Listing 1.3: Excerpt from HTML 4.01 Transitional DTD—Paragraphs

```
<!--========= Paragraphs =========-->

<!ELEMENT P - O (%inline;)* -- paragraph -->
<!ATTLIST P
%attrs;                     -- %coreattrs, %i18n, events --
%align;                     -- align, text alignment --
>
```

The attributes in this DTD are a bit different. You'll notice that the `align` category of attributes is included as well (`align=`*x* where *x* is `left`, `right`, `center` or `justify`). Alignment isn't allowed in the Strict DTD because of the idea that presentation should be separated from the document structure and its content.

NOTE The HTML 4.01 Strict DTD can be found at `www.w3.org/TR/html401/sgml/dtd.html`. The HTML 4.01 Transitional DTD is available at `www.w3.org/TR/html401/sgml/loosedtd.html`. All of the DTDs in this chapter are public and online at the W3C website, `www.w3.org/`.

Enter XHTML

XHTML 1 is, again, the rewrite of HTML as an XML application. The primary concepts in HTML 4—especially the separation of document structure from presentation and issues concerning accessibility and internationalization—remain intact in XHTML 1. What's more, the three DTD offerings (Strict, Transitional, and Frameset), originally from HTML 4 and later refined by HTML 4.01, are essentially the same DTDs in XHTML 1.

Despite these similarities, there are quite a few differences of great importance from both theoretic and semantic standpoints.

XHTML in Theory

XML brings several important ideas and incentives to web designers and developers through XHTML:

Reintroduce structure back into the language Picking up on the SGML and XML idea that documents should be written in conformance with the rules set out within the languages, XHTML makes it clear to authors that structural and semantic rules should be adhered to and *must* be adhered to in order to create compliant pages.

Provide designers with incentives to validate documents Validation carries with it some controversy, but it's a powerful learning tool that helps you find your mistakes, fix them, and in the process, understand the way a specific DTD works. Validation, therefore, is an encouraged practice.

Accommodating new devices Part of the drive to accommodate XML in the web development environment has to do with the interest of delivering web-based content to other devices such as PDAs, cell phones, pagers, set-top boxes, WebTV (now known as MSNTV), and even television.

With XHTML 1.1, the concept of separation of structure and presentation is complete. XHTML 1.1 has only one public DTD, based on the Strict DTD in XHTML 1. Web authors also have the option to work with *modularization*.

Modularization breaks HTML down into discrete modules such as text, images, tables, frames, forms, and so forth. The author can choose which modules they want to use and then write a DTD combining those modules into a unique application.

This is the first time you really see the extensibility introduced by XML at work: instead of having only the public DTDs to choose from, authors can create their own applications.

NOTE An overview of XHTML modularization can be found at `www.w3.org/MarkUp/modularization`. The actual XHTML 1.1 recommendation is at `www.w3.org/TR/xhtml11/`. Modularization is a fascinating and rather dramatic change to the way we approach pages, but it is beyond the scope of this book to cover it in detail.

Semantic Changes from HTML

In practice, XHTML works a bit differently from HTML. XHTML is much more rigorous than HTML and demands close attention to details.

- It is recommended but not required that an XHTML 1 document be declared as an XML document using an XML declaration.
- It is required that an XHTML 1 document contain a `DOCTYPE` that denotes that it is an XHTML 1 document and also denotes the DTD being used by that document.
- An XHTML 1 document has a root element of `html`. The opening tag of the `html` element should contain the XML namespace `xmlns` and the appropriate value for the namespace.
- The syntax and structure of the document must follow the syntactical rules of XHTML.

XML Prolog, *DOCTYPE* Declaration, and Namespace

An XHTML document may contain several structural elements to be considered correct.

The XML Prolog

The XML Prolog is a declaration that can appear above your `DOCTYPE` declaration. The prolog is recommended but not required. Part of the reason it is not required is that some browsers (including IE 4.5 for Mac, IE 6 for Windows, and Netscape 4 for Windows) will display XHTML pages inappropriately if it is used.

So, most XHTML 1 authors interested in the best interoperability leave it out. However, because the encoding information is important in many instances—particularly when working with international documents—if you don't use the XML declaration, you are encouraged to be sure encoding is set on your server or in a `meta` tag. Here's an example of the XML prolog, which states the XML version of the document as well as the document's encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The DOCTYPE Declaration

There are only three DTDs available in XHTML 1: Strict, Transitional, and Frameset, all carrying over with some minor differences from HTML 4.01. The `DOCTYPE` declaration declares the language version, interpretation, and location of the related DTD.

The way a DOCTYPE declaration is written is important for the reasons described in this DOCTYPE Switching sidebar. The following shows the available DOCTYPE declarations for HTML 4.01, XHTML 1.0, and XHTML 1.1, as they should be written.

HTML 4.01 Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
```

HTML 4.01 Transitional

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
     "http://www.w3.org/TR/html4/loose.dtd">
```

HTML 4.01 Frameset

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
     "http://www.w3.org/TR/html4/frameset.dtd">
```

XHTML 1.0 Strict

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

XHTML 1.0 Transitional

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

XHTML 1.0 Frameset

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

XHTML 1.1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

DOCTYPE Switching

In many recent browsers, an implementation for managing standard versus nonstandard markup has emerged. Browsers with this feature, referred to as DOCTYPE Switching, will behave in different ways *depending upon the DTD* that is declared in your document, or if in fact the DTD is declared at all.

This behavior involves switching *modes* to best represent standard versus nonstandard markup. The two modes are *quirks mode*, which behaves just as any legacy browser would, and *strict rendering mode*, which follows the standard.

Those pages containing older or transitional HTML DOCTYPEs or no DOCTYPE at all are displayed using quirks mode. Documents with correct Strict or XHTML DOCTYPEs use strict rendering mode.

This switching becomes more important as you delve into CSS, because certain rendering modes create different results. I'll point these concerns out as you move through the CSS chapters in the next part of the book.

A switching table created by Eric A. Meyer (technical editor of this book) can be found at `www.meyerweb.com/eric/dom/dtype/dtype-grid.html`. Another table, by Matthias Gutfeldt, is available at `http://gutfeldt.ch/matthias/articles/doctypeswitch/table.html`. This table shows how various browsers will relate to given DOCTYPE declarations.

The XML Namespace for XHTML

An XML namespace is a collection of unique element and attribute names. In XHTML, the namespace points to the related document at the W3C. The namespace is placed in the root element of the document tree, `html`:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

XHTML Syntax

Once an XHTML document contains the necessary declarations and basic structural information, you can examine the syntax changes resulting from XML's influences on web markup, including:

- Heavy focus on logical markup
- Case sensitivity
- Well-formed syntax
- Specific management of empty and non-empty elements
- Quotation requirements
- Escaping of script characters
- Management of minimized attributes

Each of these changes brings a marked amount of rigor to your authoring practices. Whether you end up using HTML or XHTML to mark up the documents you'll be styling with CSS, the knowledge of these practices will greatly influence your ability to write your style sheets with equal logic and organization.

Logical Markup

It can't be expressed enough that anyone wishing to learn CSS must understand the value of logical markup. When you work with content, the proper use of headers, paragraphs, breaks, lists, and so on should follow a sensible tree.

If you've ever wondered why an `h1` is bigger than an `h6` instead of the other way around, consider that headers in a document are meant to be organized by level of topic importance, like in an outline.

Let's say you've got three important places to go today: Bank, Post Office, Grocery Store. If you were creating a document tree out of those three topics, they would all be level 1 headers. The main activities you want to do at each stop would comprise your level 2 headers, and so on.

Paragraphs of content should be structured properly, too. Other items, such as lists, can easily organize information in a logical way. As you build your page, keep the tree concept in mind because you can work off of the elements in your tree when creating your style sheets—as you will see in many examples throughout the course of this book.

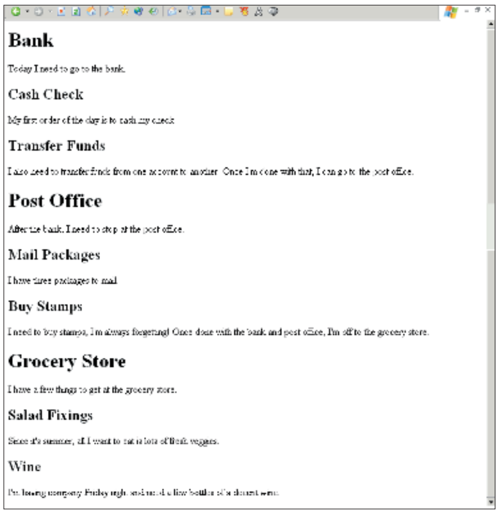
Using the Bank, Post Office, Grocery Store example, Listing 1.4 shows how this structure might pan out within a document.

Listing 1.4: Exploring the Logical Structure of Level 1 and 2 Headers

```
<h1>Bank</h1>
<p>Today I need to go to the bank.</p>
<h2>Cash Check</h2>
<p>My first order of the day is to cash my check.</p>
<h2>Transfer Funds</h2>
<p>I also need to transfer funds from one account to another. Once I'm done
with that, I can go to the post office.</p>

<h1>Post Office</h1>
<p>After the bank, I need to stop at the post office.</p>
<h2>Mail Packages</h2>
<p>I have three packages to mail.</p>
<h2>Buy Stamps</h2>
<p>I need to buy stamps, I'm always forgetting! Once done with the bank and
post office, I'm off to the grocery store.</p>

<h1>Grocery Store</h1>
<p>I have a few things to get at the grocery store.</p>
<h2> Salad Fixings</h2>
<p>Since it's summer, all I want to eat are lots of fresh veggies.</p>
<h2>Wine</h2>
<p>I'm having company Friday night and need a few bottles of a decent
wine.</p>
```

Of course, you can go on to develop your markup to use additional headers. The important lesson is tapping into using headers the way they were intended. The problem prior to style sheets was that designers were limited to what the default display was of their header (unless they also used a font tag and attributes to modify it). With CSS, the logic can be restored to the page while the separate presentation rules can leave the customized look of these components in your hands.

Figure 1.10 shows this sample page, unstyled. In Figure 1.11, I added a few simple CSS styles for the headers.



Bank

Today I need to go to the bank.

Cash Check

My first order of the day is to cash my check.

Transfer Funds

I also need to transfer funds from one account to another. Once I'm done with that, I can go to the post office.

Post Office

After the bank, I need to stop at the post office.

Mail Packages

I have three packages to mail.

Buy Stamps

I need to buy stamps, I'm always forgetting! Once done with the bank and post office, I'm off to the grocery store.

Grocery Store

I have a few things to get at the grocery store.

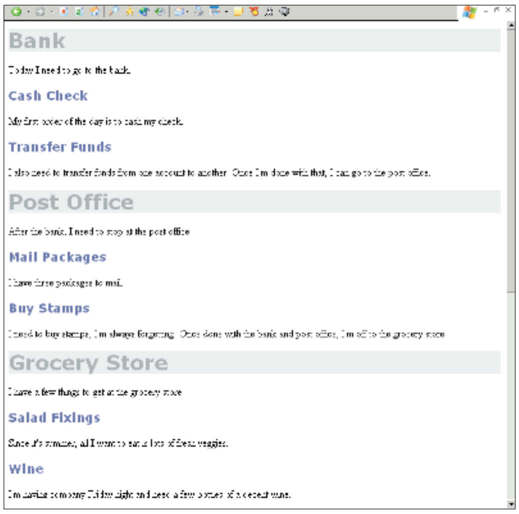
Salad Fixings

Since it's summer, all I want to eat is lots of fresh veggies.

Wine

I'm having company Friday night and need a few bottles of a decent wine.

Figure 1.10: Studying logical structure, unstyled



Bank

Today I need to go to the bank.

Cash Check

My first order of the day is to cash my check.

Transfer Funds

I also need to transfer funds from one account to another. Once I'm done with that, I can go to the post office.

Post Office

After the bank, I need to stop at the post office.

Mail Packages

I have three packages to mail.

Buy Stamps

I need to buy stamps, I'm always forgetting. Once done with the bank and post office, I'm off to the grocery store.

Grocery Store

I have a few things to get at the grocery store.

Salad Fixings

Since it's summer, all I want to eat is lots of fresh veggies.

Wine

I'm having company Friday night and need a few bottles of a decent wine.

Figure 1.11: The same document, styled

Case Sensitivity

HTML is not case sensitive. This means that HTML elements and attributes names can be in upper-, lower-, or mixed-case:

```
<body background="my.gif">
```
or
```
<BODY BACKGROUND="my.gif">
```
or even
```
<BoDy background="my.gif">
```

All of these examples mean the same thing in HTML.

On the other hand, XML *is* case sensitive. This means that XHTML is also case sensitive. In XHTML 1, all elements and attribute names *must* be written in lowercase:

```
<body background="my.gif">
```

NOTE Attribute values, such as “`my.gif`”, can be in mixed case. This is especially important in instances where the files are on servers with case-sensitive file systems.

Well-Formed Syntax

Many HTML browsers are quite forgiving of HTML errors and many HTML tools don't conform to standards. As such, some web designers have either inadvertently created poorly formed markup or learned bad habits.

The following example will work in many browsers:

```
<b><i>Welcome to MySite.Com</b></i>
```

It will display as both bold and italic in a forgiving browser. But, if you take a pencil and draw an arc from the opening bold tag to its closing companion, and then from the opening italic tag to its closing companion, you'll see that the lines of the arcs intersect. This demonstrates improper nesting of tags and is considered poorly formed.

NOTE In a conforming browser, assuming the content displayed at all, it would be italic but not boldface.

In XHTML 1, such poorly formed markup is unacceptable because the potential problems resulting from nonstandard methods are unacceptable. The concept of *well-formedness* must be adhered to in that every element must nest appropriately. The XHTML 1 equivalent of the prior sample is as follows:

```
<b><i>Welcome to MySite.Com</i></b>
```

Draw the arcs now, and you'll see that they do not intersect. These tags are placed in the proper sequence and are considered to be well-formed.

Management of Non-Empty and Empty Elements

A *non-empty element* is one that contains an element and some content:

```
<p>This is the content within a non-empty element.</p>
```

An *empty element* is one that has no content, just the element and any allowed attributes, such as `hr`, `br`, and `img`. XML says that empty and non-empty elements must be properly terminated. In HTML, non-empty elements often have optional closing tags.

In HTML, I could write the paragraph above as follows:

```
<p>This is the content within a non-empty element.
```

This is considered correct. XHTML 1 demands that non-empty elements be properly terminated with a closing tag, as in the first example.

Another example is the list item, `li`, element.

In HTML, you could have a list like this:

```
<ul>
<li>Bank
<li>Post Office
<li>Grocery Store
</ul>
```

or like this:

```
<ul>
<li>Bank</li>
```

```
<li>Post Office</li>
<li>Grocery Store</li>
</ul>
```

In XHTML 1, *only* the latter method is allowed.

Empty elements work a bit differently. They are terminated in XML with what is known as a *trailing slash:*

```
<br>
```

becomes:

```
<br/>
```

Due to problems some browsers accustomed to interpreting HTML have with this method, a workaround was introduced, adding a space before the slash: `br /`. You should always use the space prior to the trailing slash in XHTML documents.

Here's an XHTML example of the image element, which is an empty element:

```
<img src="my.gif" height="55" width="25" border="0" alt="picture of me" />
```

Other empty elements of note are `hr`, `meta`, and `link`.

Quotation Rules

Quotation marks in HTML are arbitrary in that you can use or not use them around attribute values without running into too much trouble. There's no rule that says that leaving values unquoted is illegal. The following is perfectly acceptable in HTML:

```
<table border=0 width="90%" cellpadding=10 cellspacing="10">
```

Despite the fact that some attribute values are quoted and others are not, browsers will render this markup just fine. However, if you want to conform to XHTML 1, you'll have to quote all of your attribute values:

```
<table border="0" width="90%" cellpadding="10" cellspacing="10">
```

TIP You can never go wrong when you quote your attribute values in HTML, so get in the practice of always quoting values!

Other Markup and Code Concerns in XHTML

There are two other important concerns of which to be aware when working with XHTML:

Escaping certain characters in any inline script Let's say you have a JavaScript within your document. Any ampersand (`&`) must be *escaped* properly (that is, coded as an entity, not input using the keyboard symbol) as `&amp;` for that document to be valid.

No attribute minimization is allowed *Attribute minimization* is a phenomenon that occurs in HTML, where an attribute is minimized to only the attribute name. An example of this is the `nowrap` attribute. In HTML, the attribute name can stand alone, with no value. However, in XHTML, minimization is not allowed—the attribute name is its value. Therefore, to be valid in XHTML, the HTML `nowrap` attribute must become `nowrap="nowrap"`.

As you can see, none of these changes are monumental. A bit different, yes, but if you begin to use XHTML, you'll find that your markup is a lot more consistent. That consistency is part of what makes XHTML so attractive—it provides a strong foundation upon which to build future constructs as well as to help you and your team members manage documents within a site more efficiently.

Creating Structured Documents

No doubt you're itching to get your hands into the work and actually create a structured document. You will now learn to create a structured HTML 4.01 document as well as an XHTML 1 document.

Authoring a Structured HTML Document

Open your favorite web design tool. You can use anything you like as long as it allows you to work by hand and, when you save your changes, your changes remain intact. You can also use any plain text editor such as Notepad or SimpleText.

1. Begin with the `DOCTYPE` declaration. In this case, I've chosen HTML 4.01 Strict because I don't intend to have any presentational elements or attributes within the document, just the structure and content:

   ```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">
   ```

2. Add the root element. In this case, it will be `html`. Note that I've added both the open and closing tags:

   ```
   <html>

   </html>
   ```

3. Within the `html` tags, add the `head` element, along with the `title`:

   ```
   <head>
   <title>Structured HTML Document</title>
   </head>
   ```

4. Add the `body` element below the closing `</head>` tag:

   ```
   <body>

   </body>
   ```

Listing 1.5 shows a complete HTML 4.01 Strict Document that contains all the necessary components to begin working with HTML 4.01.

Listing 1.5: A Conforming HTML 4.01 Strict Document Template

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
```

```
<title>Structured HTML Document</title>
</head>

<body>

</body>

</html>
```

Now that the necessary structural components are complete, you can add some content and structure according to the logical ideas described earlier in this chapter. Listing 1.6 shows my HTML 4.01 document with content. You can follow my lead, or be creative and add your own content.

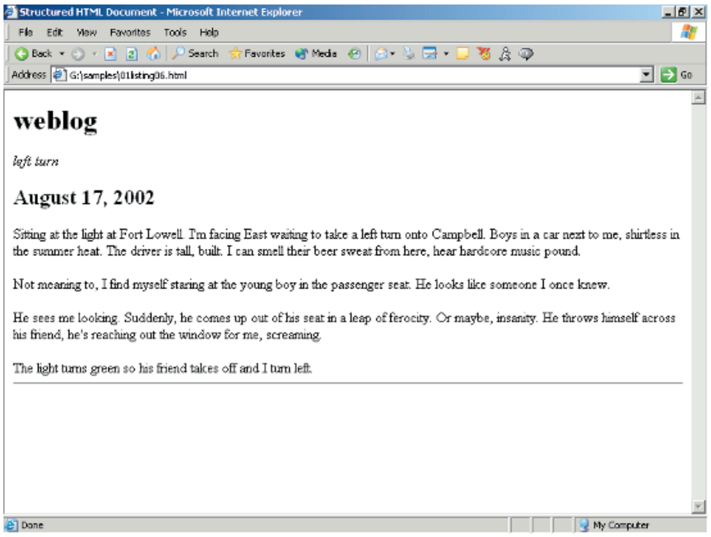
NOTE Be sure to use only structural markup when managing your content. Do not use visual presentation such as color, alignment, text styles, and so on. If you aren't sure about something, you can either try to look it up within the DTD, or better yet, wait till the end of this section, where you'll walk through validation. If you've got an error, the validator will let you know.

Listing 1.6: A Conforming HTML 4.01 Strict Document with Logical Styles

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Structured HTML Document</title>
</head>

<body>

<h1>weblog</h1>

<p><em>left turn</em>
<h2>August 17, 2002</h2>

<p>Sitting at the light at Fort Lowell.  I'm facing East waiting to take a
left turn onto Campbell. Boys in a car next to me, shirtless in the summer
heat. The driver is tall, built. I can smell their beer sweat from here,
hear hardcore music pound.

<p>Not meaning to, I find myself staring at the young boy in the passenger
seat. He looks like someone I once knew.

<p>He sees me looking.  Suddenly, he comes up out of his seat in a leap of
ferocity. Or maybe, insanity. He throws himself across his friend, he's
reaching out the window for me, screaming.

<p>The light turns green so his friend takes off and I turn left.
```

```
<hr>

</body>

</html>
```

Save the document, as you'll be validating it in just a bit. Figure 1.12 shows the unstyled page. In Figure 1.13, you can see my content here as viewed from within my website.



Structured HTML Document - Microsoft Internet Explorer

Address G:\samples\01listing06.html

weblog

left turn

August 17, 2002

Sitting at the light at Fort Lowell. I'm facing East waiting to take a left turn onto Campbell. Boys in a car next to me, shirtless in the summer heat. The driver is tall, built. I can smell their beer sweat from here, hear hardcore music pound.

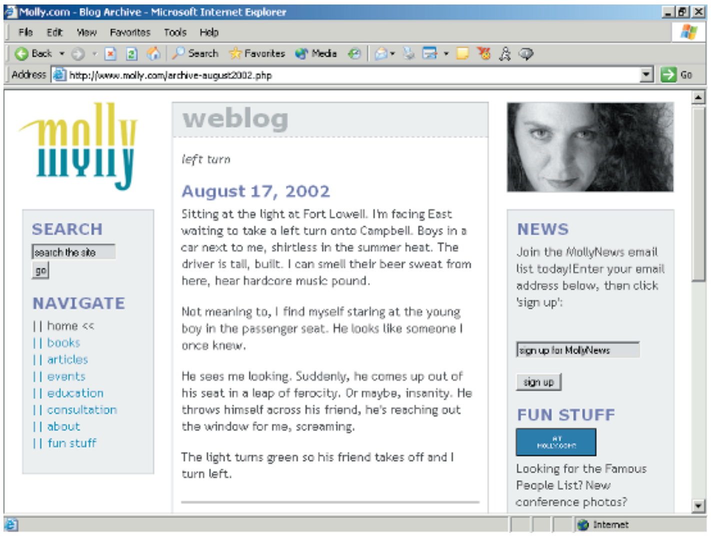
Not meaning to, I find myself staring at the young boy in the passenger seat. He looks like someone I once knew.

He sees me looking. Suddenly, he comes up out of his seat in a leap of ferocity. Or maybe, insanity. He throws himself across his friend, he's reaching out the window for me, screaming.

The light turns green so his friend takes off and I turn left.

Done — My Computer

Figure 1.12: Unstyled markup



Molly.com - Blog Archive - Microsoft Internet Explorer

Address http://www.molly.com/archive-august2002.php

SEARCH

search the site

go

NAVIGATE

|| home <<
|| books
|| articles
|| events
|| education
|| consultation
|| about
|| fun stuff

weblog

left turn

August 17, 2002

Sitting at the light at Fort Lowell. I'm facing East waiting to take a left turn onto Campbell. Boys in a car next to me, shirtless in the summer heat. The driver is tall, built. I can smell their beer sweat from here, hear hardcore music pound.

Not meaning to, I find myself staring at the young boy in the passenger seat. He looks like someone I once knew.

He sees me looking. Suddenly, he comes up out of his seat in a leap of ferocity. Or maybe, insanity. He throws himself across his friend, he's reaching out the window for me, screaming.

The light turns green so his friend takes off and I turn left.

NEWS

Join the MollyNews email list today! Enter your email address below, then click 'sign up':

sign up for MollyNews

sign up

FUN STUFF

Looking for the Famous People List? New conference photos?

Internet

Figure 1.13: The styled entry at Molly.Com

Creating a Structured XHTML Document

The process here is essentially the same, although there are specific differences as noted earlier in terms of the XML Prolog and the XHTML syntax in use.

As with the prior exercise, open your favorite editor and begin a new document.

1. This time, you'll begin by adding the XML Prolog. Remember, this is recommended but not required, and in most cases it's best to leave it out of a document to avoid browser rendering problems. However, here you'll learn to include it.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   ```

2. Now add the `DOCTYPE` declaration. In this case, I've chosen XHTML 1.1, which is based on the XHTML 1 Strict DTD:

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
   ```

3. Now, add the root element. Note that I've added both the open and closing tags:

   ```
   <html>

   </html>
   ```

4. Within the opening `html` tag, add the XML namespace for XHTML:

   ```
   <html xmlns="http://www.w3.org/1999/xhtml">
   ```

5. Within the `html` tags, add the head element, along with the `title`:

   ```
   <head>
   <title>Structured XHTML Document</title>
   </head>
   ```

6. Add the `body` element below the closing `/head` tag:

   ```
   <body>

   </body>
   ```

Save the document for validation. Listing 1.7 shows a complete XHTML 1.1 Strict Document that contains all the necessary components to begin working with XHTML 1.1.

Listing 1.7: A Conforming XHTML 1.1 Document Template (with XML Prolog)

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Structured XHTML 1.1 Document</title>
</head>

<body>

</body>

</html>
```

Now I'm going to add the same content to this document that I did to the HTML 4.01 document, but I will modify the document to be in conformance with XHTML. In this case, that means closing the non-empty paragraph elements and properly terminating the horizontal rule in accordance with XHTML.

Listing 1.8 shows the results.

Listing 1.8: XHTML 1.1 Document with Content

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Structured XHTML 1.1 Document</title>
</head>

<body>

<h1>weblog</h1>

<p><em>left turn</em></p>
<h2>August 17, 2002</h2>

<p>Sitting at the light at Fort Lowell.  I'm facing East waiting to take a
left turn onto Campbell. Boys in a car next to me, shirtless in the summer
heat. The driver is tall, built. I can smell their beer sweat from here,
hear hardcore music pound.</p>

<p>Not meaning to, I find myself staring at the young boy in the passenger
seat. He looks like someone I once knew.</p>

<p>He sees me looking.  Suddenly, he comes up out of his seat in a leap of
ferocity. Or maybe, insanity. He throws himself across his friend, he's
reaching out the window for me, screaming.</p>

<p>The light turns green so his friend takes off and I turn left.</p>

<hr />


</body>

</html>
```

Validating Your Documents

In this section, you'll use the W3C validator to test your documents. First, you'll test the HTML document, then the XHTML document. Then, you'll do some validation tests on your own.

Validating the Document

The W3C validator can validate a document online or by upload. To validate your document online, you'll first need to place it on a web server. To validate your document by upload, be sure you know the name and location of the document.

Then, follow these steps:

1. Point your web browser to `http://validator.w3.org/`.
2. If you are validating an online document, enter the address of your document in the Validate by URI address field. If you are uploading your file, click the Upload Files link and add the file from your hard drive. Leave all the other options as they are.
3. Select the Validate This Page or Validate This Document button. The validator will now compare your document to the DTD you described in the document.

First validate your HTML document, then repeat this step with your XHTML document. You may find that the validator returns errors as well as warnings. An error is a problem with the markup that must be fixed for the document to be valid. A warning provides you with information that might assist you in improving your document. Warnings will not affect your document's validity.

If any errors are reported, examine what they are, troubleshoot your document, make any necessary changes, and revalidate until your document passes the validation test.

Encoding Warnings

If you are uploading files, you will generate a warning with the HTML example here regarding proper encoding.

Encoding is ideally set on the server, so this error should not appear when you are validating from an online source, assuming your server is properly configured.

Note that a warning of this nature does not interfere with your document's validity; it's simply a means of alerting you to a potential problem.

Another means of adding encoding is to place the proper encoding information into a `meta` tag. The document using the XML prolog should not generate this warning on upload or online test because the prolog contains the encoding information. In this case, the encoding is the ISO character set for Latin-1 characters.

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

Setting your encoding in a `meta` tag will ensure that you do not receive this warning.

Validating Other Documents

At this point it will serve you well to begin validating other documents that you have been working on recently. Find a document that you know might be problematic (has font tags, uses nested tables, uses proprietary browser tags—anything like that will do). Then, try validating the document with a range of DTDs.

Next Steps

Now that you have a clear idea of the way markup works in a contemporary, standard fashion, it's time to dig into the real topic at hand: CSS.

In the next chapter, you will explore CSS principles, structure and syntax, and visual models.