View Source

WEB DESIGN HAS COME very far very fast. Web designers have had to literally view source, work through trial and error, and push limits. How we've done this has been a fascinating and often spontaneous process.

Over time, however, the importance of Web standards and best practices has begun to clarify. This process has been linked directly to the most important piece of software we work with: the Web browser. Here, you'll learn everything you need to know in order to best master the ideologies, processes, and techniques of contemporary Web design.
THE WEB OF THE LATE 1990s was a wild and dangerous new frontier. Anything went, and often did. None of us were sure back then what the Web was supposed to be.

As is so often the case with a brand new toy, “pushing the limits” became our mantra.

Genesis

And push the limits we did. HTML began its life as a simple language for tagging research papers. Browser vendors and graphic designers took one look at its lack of layout capabilities and gave it a great big thumbs-down. Tricks and hacks like \texttt{<FONT>} tags and the misappropriation of the table element were developed to overcome these limitations—hacks that undermined the very structural purposes for which HTML was developed in the first place.

Basic elements were misused for layout effects completely unrelated to their original intended purposes; elements with specific functions were ignored because they didn’t look very nice. Proper structure was ignored in favor of a site’s visuals.

Of course, the major browser manufacturers at the time weren’t helping. Before the situation got better, it seemed that it was going to have to get worse. The two major players—Microsoft and Netscape—kept heaping proprietary extension on top of proprietary extension to
keep ahead of each other as the famed Browser Wars waged. While competition
keeps innovation alive, it can also stifle; Web authors were forced to deal with
multiple versions of the same site because HTML coded for one browser rarely
worked as expected in another, if at all.

In short, it was a big mess. The sacks of money carelessly thrown around
during the dot-com boom aided the Web's continued growth despite the problems.
A budget for two versions of the same site was easy to justify when the money
flowed. However, by the turn of the millennium, both content authors and in-
vestors had finally had enough.

THE BEGINNING OF A CHANGE
Early in the new century, Web budgets shriveled up. While the
economy took a nosedive, those who still had jobs quickly realized that the
extra work of coding for broken and nonstandard browsers was too much to
sustain; something had to be done.

A grassroots organization of Web designers who called themselves the Web
Standards Project (or WaSP; www.webstandards.org) (FIGURE 1) worked with
the major browser manufacturers of the time to bring their wildly varying soft-
ware into line with the “Recommendations” being generated by the World Wide
Web Consortium (or W3C). Standardizing on the specifications produced by
the W3C injected a sense of consistency into Web authoring. Over time it be-
came possible for Web designers to create a complex page in HTML and be
assured that it would render reasonably consistently across many browsers and
operating systems.

Many of the specifications released by the W3C deal with advanced coding and
architecture problems that the average Web designer will rarely need to worry
about. Although, beginning with a standardized version of HTML in the mid
1990s, the W3C produced a series of specifications for code sent to the browser
that attempted to solve the compatibility problems in an intelligent, usable, and
accessible manner.

FIGURE 1 The Web Standards Project.

note
The W3C is interested in advancing the state of the Web. Producing recommendations
since 1996, they have sought to provide solutions to the wide variety of technological
challenges the Web presents.
WHY THESE STANDARDS?

The W3C working groups involve individuals from a wide variety of backgrounds and specialties who meet to address ahead of time the issues an average Web designer should never need to worry about. The goal is to provide a series of recommendations so well thought-out that simply by following the specification properly, developers of authoring software and browsers have a clear guideline to follow, and content authors and designers can be assured that their sites are viewable and usable by the widest variety of user agents.

By “user agents” we mean Web browsers, of course; but the average desktop browser is only the tip of the iceberg. As portable devices like mobile phones take off, more and more users will browse the Web free of the shackles of the desktop. Not every user will come in using your browser of choice; not every user can. For example, those with special accessibility requirements may use assistive devices called screen readers, or special Braille displays or magnification software.

Because the design goal of these W3C-created specifications was to clear up the incompatible mess of the Web of the 1990s, it would hardly make sense to serve different versions of the same site to all these different user agents. So the recommendations—which include HTML 4.01, XHTML, Cascading Style Sheets (CSS), and the DOM—were designed with all this in mind.

These were the technologies that designers and coders within the Web Standards Project embraced. It was logical to support specifications that would guarantee the widest content accessibility, while allowing for precise visual control. CSS fit the bill, but it was a completely new way of building Web sites. Convincing other designers that it was in their best interest to learn CSS proved to be the challenge, and the first few years of the new millennium were spent figuring out exactly how to use it.

This was the climate of the Web when the idea for the Zen Garden was planted. The people who were working with CSS at the time tended to be coders and programmers; they were highly adept at figuring out the technical issues when implementing CSS, but the layouts they produced were often considered minimal, bland, and uninspiring. Graphic designers hadn't latched on to the idea of using CSS, because there weren't any exceptional examples of CSS design.
PLANTING THE SEED
In late 2002, Zen Garden creator Dave Shea started thinking about this problem. How could those who were capable of producing real beauty with CSS be inspired by examples that were less than beautiful? Because he had a background in both coding and visual arts, Shea recognized the potential of CSS as a design language. And he realized that the people who should be using it weren’t.

An idea began germinating: If only there were a central repository of great CSS design work. Simply aggregating what existed clearly wasn’t going to work; not much did exist. Encouraging designers to create new, great-looking CSS work was going to be the key to growing this idea. The seed was planted for a new project.

Influences
In an article written in 2002, Web Standards Project cofounder Jeffrey Zeldman implored those who understood the benefits of designing with Web standards to quit spending their time selling others on the benefits, and instead to just start using them. This was the best way to demonstrate the advantages, and “Show, don’t sell” became the mantra of the project. Zeldman went on to write the definitive guide on the subject, Designing with Web Standards (New Riders, 2003).

A project called Daily CSS Fun was created by Web developer Chris Casciano in 2002 (FIGURE 2). The idea was that over the course of a month, Casciano would release a new style sheet every day that would modify the underlying HTML in wildly different ways. He made no claim of being a graphic designer, but the results were compelling for their unique insight into what was possible by relying on CSS alone for layout purposes.

Due to the continued popularity of Netscape Navigator 4, a browser with insufficient CSS support, these lessons were nice in theory but far from practical for everyday Web use. All that started going away as the market share of Netscape Navigator 4 plummeted, and by 2003 that browser barely registered on the radar anymore. It was time to move on.

Early Thinking
The ideas for the Zen Garden coalesced, and in May 2003 Shea launched the result on his personal site, www.mezzoblue.com, and then quickly moved it to the domain where it now lives, www.csszen花园.com.
In late 2002, Shea had already begun creating preliminary prototypes for the Zen Garden that switched between four style sheets, each themed for one of the four ancient elements—wood, water, fire, and wind. The idea hadn't been fully developed at the time, and images weren't a large part of the equation. As pictured in FIGURE 2.7, the first Zen Garden drafts were far from inspiring.

The designs were boring because they lacked much imagery. Background images were added by making use of the background-image property. Foreground images, on the other hand, were not possible. If the img tag were used, there wouldn't be any flexibility: you'd be confined to a single set of images and continuously have to work around those. And background images alone wouldn't cut it, because design work often relies on imagery that is emphasized alongside text, rather than behind it within a piece. Clearly, background images were only a half-solution.

The workaround for the imagery problem clicked when Douglas Bowman published his now-infamous article “Using Background-Image to Replace Text”. After this method was publicized, the final key had clicked into place: Foreground images could easily be placed by hooking into the many text elements
within a page and swapping them with graphical equivalents. CSS and a little bit of extra, customizable markup allowed the replacement of inline textual items with background images—similarly to how an <img> tag with proper alt text behaves.

In April 2003, planning for the Zen Garden began in earnest. A single HTML file was built that would serve as a master, and five initial designs were created to launch with the site (FIGURES 7–11).

![FIGURE 7 #001, Tranquille.](image)

![FIGURE 8 #002, Salmon Cream Cheese.](image)

![FIGURE 9 #003, Stormweather.](image)

![FIGURE 10 #004, arch4.20.](image)

![FIGURE 11 #005, Blood Lust.](image)

**note**

If you read Bowman's original article at Stopdesign (www.stopdesign.com/articles/replace_text), you’ll notice it mentions that the technique it advocates has some basic accessibility problems. This is the same technique used for many Zen Garden designs. We’ll cover this technique in more detail in Chapter 4.

**note**

Since the Zen Garden's HTML file would be impossible to change once submissions started coming in, extra time was spent ensuring that the markup and wording of the written text was acceptable, and that there were enough unique class and id elements for the style to modify. After the launch, additional markup considerations became evident that would have been impossible to predict in advance—we’ll cover this a bit later in the chapter.
DEFAULT DESIGN—TRANQUILLE

Though the first five designs would prove important for the project's initial success, only the default design would retain a high visibility over the long term. The subtle and minimal Tranquille was always designed to be the default, though design #002, Salmon Cream Cheese, was considered for a while due to its more immediate impact; thankfully, the decision was made in favor of Tranquille.

All images and text within the default design, aside from the body type, were digitally hand-painted in Adobe Photoshop. The intended rendering of the title text was a series of hand-painted calligraphic brushstrokes, but after spending an hour with a brush and India ink, Shea decided that the Calligraphic 421 font from Bitstream was a much more desirable option (FIGURE 12).

Because Chinese characters are intricate and beautiful, many designers appreciate the extra detail they can add to a piece. Trying to avoid the typical Western method of joining a bunch of random Chinese characters together for mystique and detail, Shea spent some time researching the characters to add some meaning to the scroll hanging down from the top-left corner of the design. Well intentioned as this approach was, the result still ended up largely nonsensical anyway: The symbols chosen roughly represent a beginning, complete or whole, and skill in English.

Remember the project in grade school where you dumped some India ink onto a sheet of white paper, then blew on it with a drinking straw to create a black treelike shape, and finally pasted on crumpled pieces of tissue paper as "blossoms"? The tree in the bottom-right corner of the design grew out of fond memories of that project. As you can see from the screen shot, it went through a few revisions (FIGURE 13). The process involved sketching a rough outline with simple colors and then building up layered detail over top of it.

IMPLICATIONS

User-contributed designs began appearing within days; traffic increased to tens of thousands of visitors a day within the first week. People volunteered to translate the site into dozens of other languages, and it was immediately clear that the Zen Garden satisfied a pent-up demand that nothing else on the Web had managed to address.

Since launch, the Zen Garden has appeared in dozens of magazines around the globe and a host of other books, cementing its canonical status in the history
of Web design. Thank-you letters pour in from around the globe, coming from
designers and developers who have used the site to convince their employers,
clients, and coworkers of the value of CSS design. And most who have contrib-
uted design work have felt a nice boost to their careers due to the exposure it
has given them.

Though created and maintained by a single individual, the Zen Garden is
largely a collaborative and volunteer effort. Talented designers helped it suc-
cceed through their submissions. Everyone has a different reason for finding
value in the Zen Garden, and although it was built mostly as a demonstration
to prove the value of CSS design, it’s clearly also valuable for other reasons that
weren’t anticipated.

☐ Designers who struggle with a layout problem have a place to turn to if
they get stuck; because of the hundreds of submissions, there’s an excellent
chance someone has solved that exact problem before. Using the CSS on
the site as a method to learn how to solve layout problems is allowed and
encouraged.

☐ Similarly, new layout techniques and CSS effects are there for discovery.
Chances are good that you will find something new among the hundreds
of submissions.

☐ It’s a great place to test a Web browser’s rendering capabilities. All files are
written in valid, standards-based code, so in theory they should render the
same across all browsers. (In practice, though, the latter isn’t yet the case.
Due to bugs and unsupported elements, most designers have needed to
account for major rendering differences through various CSS filters and
hacks.)

☐ It’s a common place to point employers and clients to when making the
case for standards-based design.

☐ It’s a great cure for the creative blues. Feeling stuck? Noodle around for
inspiration.

☐ It provides wide exposure to seasoned pros and up-and-coming designers
alike. Employers and prospective clients regularly contact the designers of
the submissions they like, for both contracts and jobs. The work often ap-
ppears in international books and magazines.

☐ It provides teachers and instructors with a useful tool for teaching contem-
porary Web design practices.
Laying the Foundation

Before you dive into CSS, your Web page needs markup structure to work with. Since you’re reading this book, you’ve likely heard of the separation of structure and presentation. For those unfamiliar with the idea, one of the main advantages of CSS-based design is that it enables you to hook in to a well-written document structure and add an extra layer of styling to make it attractive.

Formatting a Web page has long meant choosing basic HTML elements according to how they appear. If you wanted an indented paragraph, you might have wrapped the text with a `<blockquote>` tag to add a margin on either side. The `blockquote` element has a purpose, however—it specifically designates that the content within it is a block of text, quoted from another source—hence the name. Using it instead for generic text that isn’t actually a quote, simply for the sake of the indents, means using the wrong tool for the job; CSS provides the ability to indent any element on a page.

Having a structural document basically means you’ve used the proper elements for the right job, and chosen them according to what they mean as opposed to how they look.

Now that CSS design is practical, the goal becomes offloading all formatting and design to the CSS file, and marking up the HTML for the purposes of structure. CSS is the design language that hooks into existing structure and applies visuals over top of it, whereas HTML dictates the structure and doesn’t need to be used for design any longer.

So what do we have to gain from paying attention to proper structure? What exactly is “structure” in this context, anyway?

**SEMANTIC MARKUP**

The difference in philosophy between coding for appearance and coding for proper semantics can be subtle, so here is a code example to help illustrate. This is considered presentational markup:

```
<br><br>
<b><font size="2">Our Family</font></b>
<br><br>
<font size="1">Pictured are Matt and Jeremy. As usual, Matt is making a funny face. We don’t have many photos where he isn’t.</font>
```
Whereas this is semantic:

```html
<div id="family">
  <h3>Our Family</h3>
  <p>Pictured are Matt and Jeremy. As usual, Matt is making a funny face. We don't have many photos where he isn't.</p>
</div>
```

In the first example, all tags were chosen strictly for formatting purposes. The `<br>` tags were used to force line breaks, the `<b>` tags were used to make the text bold, and the `<font>` tags set the text size.

In the second example, absolutely no concern was given to the way the page looks; attention was focused on how well the `h3` and `p` elements describe the function of each piece of content within the page. The `p` element contains an actual paragraph of text, and the `h3` describes a header that may be considered third level within the rest of the page.

The key is that in the second case, it doesn't matter what the elements look like, because we will override them with CSS. They were chosen because they best describe the content. This is what defines HTML semantics: elements that are chosen for their purpose and not their appearance.

There are genuine benefits to formatting a document semantically, which we'll discuss just a little bit later in this chapter. For now, keep in mind that a major design goal when building with CSS is to start out with a valid, well-structured, and semantic HTML document, and then apply style on top of that solid foundation.

**BUILDING GREAT MARKUP**

There is no one-size-fits-all solution for building a document because every page is different and will therefore have different structural needs. But there are some general guidelines to keep in mind when constructing your own HTML.

**Use a DOCTYPE**

Beginning your markup with a DOCTYPE (short for *Document Type*) identifies the markup language you're using to browsers and other user agents. No matter which language you choose, starting with a DOCTYPE is the first step toward healthy markup.
Example:

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

Specify a Language/Character Set

Just as important as including a DOCTYPE is tagging your document with a human language. If you've ever stumbled across a page full of gibberish that was actually meant to be Japanese, Greek, Swahili, or otherwise, you've probably experienced what happens when the document author has forgotten to set the encoding. Because Google and other search engines will filter results differently based on the language of a search request, proper encoding pays off if you want search engine traffic. We'll explore character encoding in more depth later in this chapter.

Examples:

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

Sets the document's XML language, in this case the ISO code for English, en.
```  

```html
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

Assigns a character set to the document, in this case UTF-8.
```

Title It

Every page needs a descriptive `<title>` element. They're saved as link descriptions in a viewer's browser when your page is bookmarked, and search engines consider keywords within a page title more authoritative—and therefore give them a nice ranking boost in many instances. Each page within a site deserves its own unique title, if possible. They should be descriptive and relevant to the individual page, instead of generic across the whole site.

Example:

```
<title>css Zen Garden: The Beauty in CSS Design</title>
```

Use the Proper Elements

We've already touched on this, but it bears repeating: Mark up your document with HTML elements that describe the structure of your content, not how it looks. Use a `p` for paragraphs, not for line breaks. Use a `blockquote` for blocks of quoted text, not for indents. And so on.
Of course, not every HTML element has survived in XHTML 1.0 Strict. In fact, the list of elements you can use in the latter (and XHTML 1.1) is smaller but more focused. If there is no element that matches the structure you're attempting to create, it's time to think about using the all-purpose div or span elements. But there's a caveat... keep reading.

**Example:**

```html
<h3>The Road to Enlightenment</h3>
```

and not:

```html
<code style="font-size: 1.5em;">The Road to Enlightenment</code>
```

### Avoid div-itis

The danger when using div and span elements is in going overboard. A few of them placed in strategic spots can greatly enhance the inherent structure of a document; too many, and you may wish to question whether there's a more appropriate HTML element you should be using. For example, a div should never be used where an h3 would make more sense, and a span should never replace a label.

But that doesn't mean they should be avoided. A few divs in logical spots throughout your document will provide extra styling control and logical separation of sections. Think of a div as a reusable container: You don't want to bury your content in too many containers, but a few well-placed divs can keep content sorted well.

**Example:**

```html
<div id="pageTitle">
  <h1>CSS Zen Garden</h1>
  <h2>The Beauty of <acronym title="Cascading Style Sheets">CSS</acronym> Design</h2>
</div>
```

### Minimize Markup

Implied in the previous two tips is that less markup is better—it is. Assuming that you've built a solid structure, reducing markup bloat should be a universal goal. Use only the elements that are needed, and trim the rest. Not only will file sizes be smaller—and downloads therefore quicker—but a browser will spend less time interpreting the final file and get it onscreen faster.
Example:

\<p\>The Zen Garden aims to excite, inspire, and encourage participation.<\p\>

and not

\<div\>
  \<p\>\<span class="text">The Zen Garden aims to excite, inspire, and encourage participation.</span></p\></div>

**Use class and id Appropriately**
Adding identifying attributes to various elements allows you to hook into them later with CSS and JavaScript. A *class* is a reusable attribute that may be applied to any element on a page, while an *id* is a unique attribute that may be used only once per page.

You can apply more than one class to an element, and apply a class more than once across a page. You can apply both a class and an *id* to a single element, but remember that using more than one instance of the same *id* within a page is invalid. Alphanumeric characters (a-z, A-Z, 0-9) can be used in both class and *id* names, but neither may start with a number. And even though some browsers are forgiving, it's wise to consider both case-sensitive, for ease of debugging.

**Three examples of valid identifiers:**

\<body class="homepage">

\<p id="introduction">

\<div id="section5" class="top corner solid">

**Two examples of invalid identifiers:**

\<span class="15pxhigh">

\<div id="footer">\<div id="footer">


THE TEST OF TIME

If you’ve ever written a document that was meant to last a considerable time into the future, you will recognize that a lot of attention must be paid to getting things just right.

Consider the publishing industry, for example. This book has been through edits and reviews by multiple people to tighten the language and check the facts. Because books tend to stay on people’s shelves much longer than a Web page stays in a browser’s cache, all this work has to be done in advance to make sure the final result is a manuscript that is accurate. (Knock on wood!)

Simultaneously convenient and problematic, the Web has no such limitations. A Web site is uniquely served from a computer every time a viewer accesses it. Generally this means that you can go in and make content or design changes after the publishing date. Every returning visitor will see the changes, unlike a book, which doesn’t change until it is updated or a second edition is printed. Of course, the temptation to continue tweaking and refining often proves great, much to the chagrin of designers who have signed a fixed contract.

Though it lives on the Web, the Zen Garden presented some unique challenges that made writing the underlying HTML more akin to writing a book than a Web site. Modifying the markup after designs started coming in would break the older submissions and unfairly change the goals of the site. Editing the text would alter the size of the document and perhaps modify the layout of certain designs in an unpredictable manner. Making any structural or visual changes would be detrimental, so the HTML needed to remain precisely as written.

ZEN GARDEN SOURCE HTML

Consider the Zen Garden HTML structure as permanent as a book. It won’t change, even though there are countless ways it could improve. The content itself may undergo some tweaking, but the basic HTML elements are effectively set in stone.

The next few pages are a complete listing of the Zen Garden source HTML for the sake of reference later on in the book. You can also get a copy of this by visiting the site—www.csszengarden.com—and viewing the page source in your browser. This is accomplished the same way in most major browsers: You choose View > View Source or the nearest equivalent. Or even just find the link to the sample HTML file on the Zen Garden site itself for an easy download.
A DOCTYPE is essential for identifying the type of document you've written. See "Use a DOCTYPE," earlier in this chapter.

Specifying the character encoding of the document you are writing is incredibly important in a global environment like the Web. See "Character Encoding," later in this chapter, for more; and for a broader overview of character-encoding issues see Joel Spolsky's article "The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)" at Joel on Software (www.joelonsoftware.com/articles/Unicode.html).

This empty script element is for the sake of avoiding a styling glitch in Microsoft Internet Explorer for Windows. See "Flash of Unstyled Content" at Blue Robot (www.bluerobot.com/web/css/fouc.asp).

Though not as necessary anymore, this method of importing CSS ensures that Netscape Navigator 4 doesn't try to apply the external CSS file, instead displaying simple, unstyled markup. See "Tricking Browsers and Hiding Styles," from Eric Meyer on CSS (www.ericmeyerontcss.com/bonus/trick-hide.html).

Comments that remain in the original source on the site have been removed from this listing, in favor of notes in the sidebar to further explain some sections and offer reasoning for various items. Again, you can always view the comments by visiting the site.
An id applied to the body element is called a CSS Signature. A user style sheet may over-ride any or all style on a page by tapping into the CSS Signature. See "[css-d] CSS signatures" (http://archivist.incutio.com/viewlist/css-discuss/13291).

By separating logical groups of elements into their own div elements, layout styling is easier and more flexible.

Always make things easier than they need to be. Even though those with an interest in contributing to the Zen Garden should know how to download the sample files from these links (right-click the link and choose Save As), not everyone with an interest in studying the code does. Email still comes in asking how to save the files, which just goes to show that your target audience will always have unexpected members.

In hindsight, it would have made more sense to end the #intro div here and place #preamble within #supportingText. The length of #preamble means that it more logically fits with the larger block of text that makes up the bulk of the page than within the introduction. For positioning and stylistic considerations, it might have been more flexible that way, too.
Grammar and markup pedants will tell you that W3C is technically an abbreviation, not an acronym. This is true, and there is an abbr element defined in HTML for abbreviations of this type; it’s unsupported in Internet Explorer, though, so many use acronym instead, which is supported.

No, we don’t know what a “structurist” is. At the time of writing there seemed to be a terribly pressing need to differentiate between those who code and those who are concerned with HTML structure. What that need was, though, we really can’t remember.

What is &amp;#8217;? It’s the character code for a proper typographical curvy quote (‘), used instead of the single prime marks (‘), which usually serve as replacement characters. The Web supports a wide range of typographical characters and conventions, which should be used wherever possible. See “The Trouble with EM’s and Other Shady Characters” at A List Apart (www.alistapart.com/articles/emen).

We must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the acronym title=“World Wide Web Consortunm=“W3C”<acronym><acronym title=“Web Standards Project”>W3SPIacronym>&amp; the major browser creators.</acronym></p>

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.</p>

No, we don’t know what a “structurist” is. At the time of writing there seemed to be a terribly pressing need to differentiate between those who code and those who are concerned with HTML structure. What that need was, though, we really can’t remember.

What is &amp;#8217;? It’s the character code for a proper typographical curvy quote (‘), used instead of the single prime marks (‘), which usually serve as replacement characters. The Web supports a wide range of typographical characters and conventions, which should be used wherever possible. See “The Trouble with EM’s and Other Shady Characters” at A List Apart (www.alistapart.com/articles/emen).

We must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the acronym title=“World Wide Web Consortunm=“W3C”<acronym><acronym title=“Web Standards Project”>W3SPIacronym>&amp; the major browser creators.</acronym></p>

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.</p>

No, we don’t know what a “structurist” is. At the time of writing there seemed to be a terribly pressing need to differentiate between those who code and those who are concerned with HTML structure. What that need was, though, we really can’t remember.

What is &amp;#8217;? It’s the character code for a proper typographical curvy quote (‘), used instead of the single prime marks (‘), which usually serve as replacement characters. The Web supports a wide range of typographical characters and conventions, which should be used wherever possible. See “The Trouble with EM’s and Other Shady Characters” at A List Apart (www.alistapart.com/articles/emen).
The requirements for submission have changed in response to the overwhelming volume. See "Submission Guidelines" at mezzoblue.com (www.mezzoblue.com/zengarden/submit/guidelines) for the latest information before submitting your work.

As you'll see in the Chapter 7, there are now ways to experiment with advanced CSS effects while providing a slightly downscaled version of the style for older browsers. Some designs have explored this idea, so the requirements about browser support might be changed to instead read "Use your best judgment."
The list of links within footer is a list ... why wasn’t a ul used to structure this group of links? That’s a rhetorical question—they should have been contained within a list element of some sort.

As the Zen Garden markup was originally written, the 1i elements surrounding each link were actually spans, which triggered an error when checking accessibility through automated services like Bobby (http://bobby.watchfire.com). The extra &nbsp; at the end of each link was a misguided attempt to circumvent the problem; a list quickly replaced the spans before it was too late, but the legacy &nbsp; entities never quite disappeared.
Accesskeys are an assistive technology for easier navigation of a Web site, but they are not without their flaws. See "Using Accesskeys—Is it worth it?" at WATS.ca (www.wats.ca/articles/accesskeys/19).

This method of browsing through the designs is laughably inadequate, now that there are more than 100 of them. The link to an external list of all the designs is much more useful (www.mezzoblue.com/zengarden/alldesigns). Still, it's often overlooked. Many people assume the Zen Garden only has eight designs, and they never find the archives. Don't make the same mistake!

Though the "view source" method is a given among those who have been working with the Web for any length of time, enough new-comers emailed asking for a way to view each design's CSS file that a link was added shortly after launch.
As you'll see in some designs, these completely empty divs and spans are for the sake of adding imagery where it wasn't practical within the regular set of elements. Think of them as reusable img elements.

**VISUAL STRUCTURE**

Designer Andy Clarke created a visual guide to the Zen Garden's markup structure (FIGURE 14). It does a great job demonstrating how the various elements nest inside each other, and serves as a good starting point for studying and adapting the Zen Garden's markup. See Clarke's accompanying write-up, "3D CSS Zen Garden" (www.stuffandnonsense.co.uk/archives/3d_css_zen_garden.html).

**DESIGNING FOR FLEXIBILITY**

From the beginning, one of the primary goals of the markup was maximum flexibility to allow for design variation. It was essential for the Zen Garden to provide as many extra markup hooks as possible so that each individual element on the page could be modified and adjusted as necessary. Without the extra bulk, selecting specific elements of the page is impossible. Descendant selectors allow generic selections like so:

```css
#quickSummary p {
    color: red;
}
```

But with that style, all p elements within #quickSummary turn red. What if you wanted to modify only the second paragraph within #quickSummary? Without a unique class on each paragraph inside it, the only way to accomplish that is with advanced CSS2 selectors:

```css
#quickSummary > p + p {
    color: red;
}
#quickSummary > p + p + p {
    color: inherit;
}
```
This is about to get confusing: The first rule is a combination of child and adjacent selectors that says, roughly, "Apply this rule to any paragraph element that is immediately preceded by another paragraph element that is also a child of quickSummary. Which means all paragraphs within quickSummary except the first one in this case, so the second rule overrides it by going an extra step and undoing the rule to any paragraphs that follow two other preceding paragraphs in a row, the first of which must be a child of quickSummary, which means all paragraphs are reset except for the first two.

Whew! That’s a mess of confusing logic for the simple sake of element selection; thankfully, CSS3 introduces a much easier way to do it:

```css
#quickSummary p:nth-child(2) {
  color: red;
}
```

But after all that fun, there’s one more thing you need to know: Neither of these methods works in Internet Explorer! You can pretty much forget they exist for now, unless you have a pressing need to use a style that much of your audience can’t see; Internet Explorer use may change over the next few years, but at the time of publication its market share was hovering around 90 percent.

This talk of advanced CSS selectors has been a rather long way of making a simple point. It should be obvious why the extra classes were necessary, if maximum flexibility was a major design goal. The reasoning for the extra span elements was similar: Providing interior elements for each paragraph on the page means extra elements available for styling.

**Image Replacement**

The spans had a further purpose: image replacement. When the Zen Garden was built, a brand new method floating around the CSS design world was the idea that if you had a header (say an h3) and you wanted to replace the text in it with a typographically rich image, then you would need two elements surrounding the text. One would be the h3, the other a span.

By hiding the text within the span using a style rule like `display: none;`, you were left with an h3 that you could apply a background image to, completing the illusion of a graphical header. Since each element on the Zen Garden has an inner span, the possibilities are wide open for this type of experimentation.
Why so much trouble when an `<img>` element would do just the same? Once again: flexibility. Swapping style sheets to dramatically alter the page just wouldn’t be possible if images were hard-coded into the HTML. Image replacement has its own set of problems, though, which we’ll talk about later in the book.

**Lighten Up**

Some decisions were made according to the best knowledge available at the time. They required more markup than newer techniques that have come to light since, and it’s not hard to see places where the bulk could be reduced in retrospect.

Since the launch of the Zen Garden, image-replacement techniques that don’t require a `span` have become available, so most `span` elements in the markup are no longer necessary. And the extra classes aren’t used in 98 percent of the designs submitted, so they’re mostly redundant and unneeded. The list of empty `<div>`s and `<span>`s at the end of the document are useless except for the extra visual hooks they give a designer.

It’s probable that these elements wouldn’t make sense for other Web sites. They have a specific task on the Zen Garden: to allow for maximum flexibility. It’s not necessary to account for such wildly varying designs on every site.

If you use the Zen Garden’s HTML as a starting point in your own work (which you are allowed to do, with our blessing), keep in mind that a large goal of standards-based design is to eliminate unnecessary elements. Using the occasional `<div>` element to surround a logical group of elements is appropriate use; wrapping a `span` around every single paragraph and header on a page is overkill. We’ll discuss structuring a document properly later in this chapter.

**Lessons Taught**

The primary mission of the Zen Garden is to showcase what’s possible with standards-based design, and browsing the various designs will provide a powerful display of the flexibility CSS offers. On its own, CSS is a design language meant to enhance a base markup language like HTML, XHTML, or XML. CSS1 was first introduced in 1996, with CSS2 following up soon after in 1998. Browsers began incrementally supporting both, and from 1998 on it became a standard practice for Web designers to control all of a site’s typography through an external CSS file. That’s about all that anyone did with it, until four or five years later.
There was a good reason why fonts were all you’d consider using CSS for—they were about the only thing you could rely on working across a variety of browsers. They didn’t even work that well, but there was a distinct advantage to doing it this way that made a lot of designers use CSS anyway.

The advantage is that a single CSS file may be referenced by all documents on a site. If you set your typeface through a CSS declaration, all pages will apply it. If you change the font or the type size in that one file, all documents on the site will instantly recognize the change and your job is done.

This was a vast improvement over the prior method of working; the notorious <font> tag that came before was a document-level presentational element that had to be applied to every single instance across a site. You might end up with seven or eight <font> tags per page, across hundreds of pages. Changing the font was a time-consuming and painful process.

This type of presentational separation offered a small glimpse of the potential that CSS offers. A redesign was made orders of magnitude easier, as moving text from the old design didn’t require hunting down and replacing thousands of <font> tags across multiple files; you just needed to copy and paste the content, then let your CSS do the rest.

But this is where common use rested for a long time. Older browsers with poor support for more advanced styling stayed around for years past their expiry date, and doing any advanced CSS work was simply impractical.

Finally, in the early years of the new millennium, the baseline support had improved to the point where full CSS layouts could be considered a valid choice. Browser support wasn’t perfect (and it still isn’t), but it was good enough to start considering CSS for more than just fonts. A few years of intense collaborative effort followed, with Web designers across the globe producing reliable methods for working with the style options available, techniques for building layouts with floats and the positioning model, and workarounds for browsers that weren’t quite up to par.

And that’s where we are today. There are browsers even now in 2005 that have problems with CSS2, which was finalized in 1998. But support is strong enough in general that the benefits of using CSS and standards-based design are tangible and paying off today for those doing so.

Let’s take a brief look at those benefits. Chances are you’re already aware of them since you’re reading this book, but if you’re interested in more in-depth detail, please consult the reading list in the Closing Thoughts section of this book.
BENEFITS OF WEB STANDARDS

Using Web standards means considering the whole picture, including both CSS and structural HTML. It's difficult to summarize the benefits of either without mentioning the other. The real payoff happens when both are used in concert, with a document's content and structure controlled by the base HTML, and its design controlled by the CSS.

The Zen Garden designs are proof that an HTML file can be modified in any number of ways by dropping a style sheet on top of it. CSS is a method of presentation that hooks into the underlying document structure of an HTML file, and though it relies on that structure, it operates independently to provide a layer of visuals on top.

What is important about the base structure is that it be well formed and valid; an invalid document will render unpredictably and cause more trouble than anyone deserves. The first rule in diagnosing a CSS layout problem, before anything else, is to make sure your markup validates.

Smaller File Sizes and Quicker Downloads

When you move your presentation out of the markup and into CSS, your file sizes usually decrease as well. What might have once taken 30 Kbits of HTML for a site's design framework might be accomplished with 10 Kbits of HTML and an additional 10Kbits or less of CSS.

In addition to initially resulting in smaller file sizes, CSS gets cached in a browser's memory. Consider that for a second—with 30 Kbits of presentational HTML to create your page's design, every new page load on your site requires pulling that 30 Kbits, each time, in addition to whatever content is actually within the page. Browsing ten pages of a site built this way requires downloading 300 Kbits almost needlessly.

But when the design is moved to an external CSS file, not only is the final page weight smaller, but less of the design needs to be downloaded on each page load—only 10 Kbits of extra HTML is required per page, since the 10 Kbits CSS file is pulled from the local browser memory each time. The 300 Kbits of downloading from the previous example might be reduced to as little as 110 Kbits in this case.

This sort of piecemeal file-size reduction can really add up, for both end users and the server. When your traffic is high, the extra bytes trimmed have huge
implications for your total bandwidth savings. Sports network ESPN's Web site went CSS in early 2003, and its numbers are almost unreal. As stated in an interview originally published on Netscape's now-defunct DevEdge (now available at www.mikeindustries.com/blog/archive/2003/06/espn-interview), the developers of ESPN.com estimate they're now saving about 2 terabytes each and every single day. This means they can serve more users, and serve them quicker, without changing anything else. That's a substantial return on a one-time investment.

**Increased Portability**

A properly structured page doesn't even need the CSS to render, actually. In some scenarios this makes perfect sense; when you design for a large, 1024-by-768 monitor, you have plenty of space to fill; the same can't be said for a small, 240-by-320 Pocket PC.

Without CSS, the bare content remains perfectly readable, links still work, and form elements are functional. Without doing anything else, using structural markup means that you have an automatic mobile version of your site.

But CSS offers support for various media types, enabling you to more specifically target your style for the media it's presented on. Computer screens, mobile devices like cell phones, and printers all have their own media types, and even more are defined or under development.

Print-media style sheets spell the death of the little "Print This Page" link. Every page on a site can be printable without much more than a simple style sheet that formats certain items, hides others, and lays out the page more precisely for the benefit of the printer. Since printers have different restrictions than monitors, the control that a print style sheet offers is welcome.

The theory is that one day every device will rely on a media type appropriate for the challenges it faces when displaying a Web page; the reality is that right now, barely anyone makes use of anything beyond screen media (page authors and device manufacturers alike). CSS display options for cell phones rarely go beyond basic text size and color, and a browser that makes use of the spoken "aural" media type has yet to appear.

It's a chicken-and-egg problem, since better support isn't likely until content is available, and no one is producing content for devices that don't yet exist. The situation is slowly improving, though, so perhaps one day a mobile CSS Zen Garden targeting cell phone and PDA styling will be possible.

---

**note**

CSS media types are outlined on W3C's "7 Media Types" page (www.w3.org/TR/CSS3-media.html).
note
The W3C's Web Content Accessibility Guidelines (WCAG) are available on the W3C site (www.w3.org/TR/WAI-WEBCONTENT).

Better Accessibility
Properly structured HTML was designed to be accessible; you can create a version of your site that conforms to 90 percent of the W3C's content-accessibility guidelines simply by writing valid markup.

Say good-bye to text-only equivalents of your content and multiple versions of a site that are painfully difficult to keep current; say hello to one-size-fits-all Web authoring. In most cases, even visually impaired users who run screen readers—which audibly read a Web page back to them—can use your site, without changing anything else.

There are caveats to this, of course, and a little later in this chapter we'll look at some of the accessibility problems that CSS might introduce.

Precise Control
As a styling language, CSS offers many rich new options that HTML formatting could never have hoped to match: precise control of leading (the space between the lines of text) with the line-height property, spacing between elements with the margin and padding properties, positioning of elements on a page through absolute and relative positioning, and more.

If you've ever felt constrained by the grid that a table-based layout imposes, you might be delighted to learn that CSS positioning allows you to smash out of it and place elements wherever you like on a page. If you've ever wished for more precise background control than the choice between a color and a single repeating image, the background properties of CSS could be a minor revelation.

Lessons Learned
No site is perfect, the Zen Garden included. It's too late to go in and change things now, but discussion over the past few years has revealed extra knowledge that is worth sharing.

ACCESSIBILITY CHECKING
After the markup was written, a quick check from Bobby confirmed that it passed most major accessibility checkpoints. A few quick changes were needed before launch to fix the few glitches that Bobby noticed. A link labeled "AAA" was added to the footer of the Zen Garden to signify that accessibility had been taken care of.
Or had it? It turns out that Bobby is not the final word when it comes to accessibility. If you familiarize yourself with the Web Content Accessibility Guidelines published by the W3C (see “Better Accessibility” earlier in this chapter you’ll soon realize there are guidelines that Bobby simply can’t check. Checkpoint 2.1, for example, states that all information conveyed through color needs to be available to the viewer without color as well—software like Bobby has no way of distinguishing this, especially if the information and color are embedded in an image.

In fact, if you do some digging on the Bobby site, you’ll also find this disclaimer:

“Accessibility is ultimately a human endeavor. It is determined by whether or not a diverse group of people with a variety of abilities and disabilities can access information efficiently. Bobby is just one step in helping to make web pages more accessible, but cannot guarantee total accessibility.”

—Frequently Asked Questions

So the Zen Garden’s HTML theoretically passed all accessibility checkpoints related to markup but there are further checkpoints that go beyond HTML. A few of them even apply to CSS, and it became evident over time that some designs weren’t taking these into account. And when you consider the accessibility implications of Fahrner Image Replacement—which we’ll discuss more in Chapter 4—it’s easy to see how the CSS can quickly cause problems that no automated checker can diagnose.

The lesson learned is that automated tools like Bobby may serve as a useful starting point for building accessible Web sites, but WCAG provides many more checkpoints that are equally as important, which it cannot check for you.

**TEXT SCALABILITY**

One particular CSS accessibility problem has to do with text resizing. It’s tempting to choose a smaller font size for your work to fit more information on the screen at once. We all know how little space we’re given to work with onscreen anyway, so there’s a justified rationale for going smaller.

The problem is that many people just can’t read text below a certain size. For some, 12-pixel-high text is unreadable, and in extreme cases even 24 pixels is too small. Of course, no one is going to start designing with 24-pixel-high body copy on today’s low-resolution monitors, so there has to be a compromise somewhere.

**note**

Bobby (http://bobby.watchfire.com/) is an online accessibility checker. There are others, such as Cynthia Says (www.cynthiasays.com) and WebAIM’s Wave (http://wave.webaim.org/).
Fortunately all modern browsers have some form of font-resizing mechanism built in. Not all users know where to find this tool in their browsers, but for those who do it’s invaluable. The problem comes in when font-size values are specified with the px units. Internet Explorer will scale fonts that are sized using em or %, but it won’t scale the px unit.

So any help that users might have received from their browsers is of no use, and the problem persists. If the designer avoids the px unit, everything should be fine, right? Probably, but Chapter 5 covers some entirely different problems that might be encountered when using em or % units.

Even if text is scalable in all browsers, though, there’s something else to consider: The design must also allow for scaling (Figure 15). That sounds easy, but if you blend fixed-size elements (like a 20px-high background image) with elements that are variable (like text that the viewer has resized from 14px to 24px), you’ll quickly notice that a design that relies on fixed sizing can break down very quickly when the font is different from what the designer expected. Most Zen Garden designs have been tested for this condition and can be resized quite a bit larger than their default.

The lesson learned is that whatever size you choose for your text, chances are that someone is going to want it larger. It’s important to be aware of the potential problems they may face when trying to make the text larger, and accommodate them when reasonable to do so.

XHTML AND MIME TYPES
The Zen Garden launched as an XHTML 1.1 document. After all, the markup passed XHTML 1.1 validation, so why not use that instead of the older XHTML 1.0 Strict?

As it turns out, there’s more to XHTML than just closing tags properly. Even if you validate your XHTML, and it does turn out that it’s completely valid, it’s not technically XHTML unless you’re serving it up with the correct MIME type.

MIME is a standard for describing information, and the only reason you need to know about it is because the correct MIME type for XHTML is application/xhtml+xml. You can use the far more common text/html, which is also the default for HTML, but only if you’re using XHTML 1.0 Transitional or if your XHTML 1.0 Strict is meant to be backward-compatible. You may never serve XHTML 1.1 as text/html; it’s always supposed to be served with an XML MIME type.
Serving XML documents with the proper MIME type involves programmatically hooking into the HTTP headers your server sends and altering those, potentially through .htaccess or httpd.conf files. If you have no idea what we’re talking about, it’s not your fault. This can be really low-level, technical stuff for the uninitiated, as it falls in the realm of server administration.

But once you’ve figured out how to serve the correct MIME type, the problems with using it are far from over. Only the most recent browsers support application/xhtml+xml, so a switching method is necessary to selectively serve it to them, while serving a more compatible MIME type to older browsers.

Further, once you turn on the proper MIME type, you trigger XML-parsing in your browser. XML has a rule that says if an error has occurred—for example, a missing </p> tag—the data must stop being parsed and return an error message. You won’t even get partial rendering in your browser; you simply won’t see your Web page (FIGURE 16). An XML document must be well formed, and that’s that—there’s no room for error.

Is XHTML really worth all this? The whole point of XHTML was to reformulate HTML as an application of XML, after all. So it seems odd to some that you would bother using XHTML without serving it as genuine XML. The benefits of serving it this way are almost nonexistent, but pedantically speaking it’s necessary if you’re going to call it XHTML. If you choose not to worry about this, you’re not alone. Many of the world’s leading Web designers use XHTML and the text/html MIME type without a moment’s pause. You’re certainly in good company; just make sure to avoid using XHTML 1.1, and use only the 1.0 variants. You might also choose to avoid the issue completely and simply stick with HTML 4.01, which is properly served as the easier text/html and may be written according to similar rules as those for XHTML.

The lesson learned is that XHTML is simple on the surface, but once you start digging into the real implications, there’s a whole tangled mess of issues awaiting you.

**FOREIGN LANGUAGES**

Once upon a time in the computer industry, if you wanted to write a document in a language other than English (assuming you were in an English-speaking country), you had just about no chance of accomplishing it without special software support.
Thankfully, today's software is far more internationally aware. The World Wide Web is the most logical place to look for evidence of this, and indeed the number of non-English sites is growing rapidly.

To reach a wider audience, the Zen Garden solicited volunteer translations (www.mezzoblue.com/zengarden/translations). As they came in, a few problems became abundantly clear: Working with foreign languages requires at least a rudimentary knowledge of character encoding, and translation is much more of an art than a science.

**Character Encoding**

Modern operating systems are quite good about working with non-English characters. A base installation of Windows XP or Mac OS X won't always include full support for the wide range of possible human languages, but language packs are available on the install CDs that enable proper rendering of many foreign characters.

Being able to see the fonts is only half the battle; copying and pasting text from a source into an HTML document only works if the character encoding matches. Text that uses shift_jis (Japanese) as its base encoding will not display properly in a document encoded with utf-8 (Unicode), for example. It's important to ensure that your HTML document's character encoding matches the text you're working with.

Encoding may be set at the server level, as a default for all pages on your site. But even if done so, it's important to specify the character encoding within your HTML using a `meta` tag:

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

The Zen Garden uses a value of `iso-8859-1`, which refers to a standardized encoding for most major European languages. Various translations use different encodings, but in hindsight the best choice would have been to use the value `UTF-8` for each.

UTF-8 is a variant of Unicode, the popular international encoding language. The advantage of using it is that multiple languages with differing character sets, like French, Japanese, Arabic, and Greek, can all potentially coexist within the same document.
The only real caveat against its widespread use on the Web right now is that some older, non-internationally aware server-side software and authoring tools won't support it. Modern browsers have no problems that prevent its use, so the user side isn't the problem; it's the server side. As well, UTF-8 creates larger files than necessary when working with Asian languages, due to the complex character sets they require, so documents written primarily in Chinese might benefit from a more specific character encoding.

The lesson learned is that due to its powerful ability to handle multiple languages, UTF-8 is the system of choice for dealing with language on the Web if the tools support it.

**Translation Discrepancies**

Because the Zen Garden relies on volunteers for translations, the quality of each varies. Native speakers of each language currently available often send comments with suggested fixes; obvious grammar and spelling flaws are easily resolved, but suggestions for alternate wording are much more difficult to implement.

Even if a professional translation service is used, human languages aren't directly interchangeable. A phrase may be interpreted in different ways, each shaded by the interpreter's experience and cultural context. For example, does the phrase "That will never fly" refer to a bad idea or a faulty airplane? Is "enlightenment" a Zen concept or a weight-loss technique? Context might clear up the ambiguity, but only if the context is properly understood.

Human language is more imprecise than computer language, so the problem persists. The best that can be hoped for is a consensus among multiple speakers of the language; factor in regional dialects and variations that have evolved over time (are you going to spell it *color* or use the British *colour*?), and it's apparent that even consensus is hard to accurately form. Especially if you don't speak the language in question!

The lesson learned here is that translating a document is no easy task, and perfection is perhaps unattainable. As a result, the Zen Garden now has a lighthearted disclaimer stating that errors are to be expected, and we're willing to live with that.
COPYRIGHT AND THEFT

Beyond demonstrational purposes, much of the value of the Zen Garden comes from inspiration and sharing of knowledge. This is a double-edged sword, since sharing ultimately means that sooner or later, someone is going to expect more from you than you wish to give.

From the beginning, all designs submitted were done so under the agreement that the CSS would be available for limited use by others, governed by a Creative Commons license. However, since the images were largely responsible for the uniqueness of the work, full copyright control of them was reserved for the original designer. Anyone wishing to base work on the designs was permitted to copy and modify the CSS, but using any of the GIF, JPG, or PNG files, or reusing them wholesale without explicit permission from the designer was prohibited.

In theory, this was a good way of retaining enough individual ownership of the work for the original designer, while allowing for sampling and reuse of parts of the CSS file to encourage learning efforts. The point of the site was to share the knowledge, so it seemed like the best of both worlds. In practice, it turned out to be less than ideal.

Not long after the Zen Garden launched, the first copycat sites started appearing. Some would rip off the entire design, images and all. These are clear cases of copyright violation, as the images are protected.

However, some would use and modify both images and CSS, or sample bits and pieces of each, or reuse design elements in a manner that felt a little bit too much like reuse. In certain cases, the final result was barely perceptible as originating from a Zen Garden design. In others the CSS carried enough of the design elements that with or without images, the result mimicked the original in ways the design’s creator wasn’t comfortable with.

What could be done about the latter? In most cases, a quick email was all that was necessary, as most people are willing to respect the rights and wishes of the person they took their inspiration from. But in cases where the violator wasn’t as courteous, the problem became enforcement. Unless the original images were involved, the Creative Commons license was open enough that there was no recourse. This type of reuse was technically allowed, although many contributors had a problem with it.
Creative copyright infringement has long been a gray area. How much of an original work do you need to change before it becomes a new work? Are you allowed to start with someone else’s copyrighted work, if the final result is completely different? Since these are questions that even the courts struggle with, there were no easy answers.

For the Zen Garden, the ideal is a solution that leaves the CSS open enough to allow for sampling but discourages wholesale or even partial reuse of the designs in commercial work (or otherwise) without the designer’s permission. A fine balance is necessary—too open and designers wouldn’t feel comfortable submitting; too closed and much of the value of the Zen Garden would be lost in overprotection.

Without shutting down the site, there will never be any guarantees that the work will not be reused against the wishes of the original designers. License or not, this is simply the nature of publishing creative work: It will get ripped off. It happens all the time—just look around pirated-sites.com.

The Web makes it slightly easier to do, thanks to the View Source option built into every browser. But it’s also slightly easier to catch, thanks to referral logs and search engines.

At the time of writing, the issue still is far from resolved. A decision has been made to continue allowing limited use of the CSS files under a Creative Commons license, one that discourages commercial reuse. It’s not a perfect solution, but it works in a way that acknowledges that most have good intentions.

The lesson learned is that a few bad eggs will do what they want no matter which license is slapped on the work, so it’s simply a matter of choosing a method that notifies those with good intentions of what they’re allowed to do.

**BUILD IT BIGGER, BUILD IT BETTER!**

Various viewers of the site have offered suggestions for making it bigger, better, different, louder, vitamin C enriched, and just about anything else you can imagine. Here is a sampling of some ideas that have been proposed:

**CMS Zen Garden**

Styling one page is nice, but what about a whole site? Why not extend the concept to different pages across a real, updated site managed by a content management system (CMS) of some sort to better reflect real-world situations?
E-Commerce Zen Garden
Similar to the CMS Zen Garden, what about styling an e-commerce site filled with genuine and constantly changing products? Options for styling could include shopping-cart widgets, sale items, featured products, and so on.

JavaScript Zen Garden
The separation of structure and presentation is a lofty goal; why not take it a step further and enhance the Zen Garden with a behavior layer that's also separate from the other two?

A great-looking proof of concept (FIGURE 17) was submitted by Rares Portan, one of the originators of this idea (www.csszengarden.com/javascript/), and a discussion about the pros and cons of adding this method to the Zen Garden ensued (www.mezzoblue.com/archives/2004/05/04/javascript_b).

XML/XSLT/RSS/Whatever Zen Garden
Insert base markup language of choice, then style it using CSS. Why not show the potential of CSS when combined with XML-based languages other than XHTML?

So Where Are They?
Without a doubt, these ideas are intriguing, and many of them would provide additional useful resources to the Web design community. In most cases the originators of the ideas have received positive feedback and the suggestion that off-site demonstrations of their concepts would be welcome.

The truth is, the Zen Garden has a very specific set of goals, and it's not all things to everyone. At this point in time, a very large base of prior work means that changing the rules would alter the purpose of the site and be unfair to those who have contributed.

New sites based on these ideas could work, though. If you can build a better mousetrap, we'd encourage trying. One or two have even been built, but the results haven't lasted long. Everyone will have a differing theory for why this is, but here's what we think.

First, the Zen Garden is easy to contribute to. The barrier for entry involves having design skill and CSS coding knowledge, but an entry doesn't require weeks' worth of time. Styling a whole site is much harder, especially when the
content changes frequently. When the work is compensated, it’s easy to justify the time; when it’s not, the time requirement is a barrier to entry.

Second, the Zen Garden’s primary goal has been accomplished. It set out to change the perception of CSS-based design, and that it has done. While the specific styles involved apply to one single page, it’s not hard to extrapolate and apply the principle to multiple-paged sites.

If someone absolutely can’t imagine how the lessons apply to more than one page, there are plenty of real-world sites making use of standards-based design that would serve as better examples. Many large corporate sites have now switched to standards-based design: Wired News (www.wired.com), ESPN.com, Chevrolet (www.chevrolet.com), AOL.com, and many more.

The lesson learned is that the demand is high for standards across a wide variety of sites, but real-world examples illustrate the value of standards even more effectively than a Zen Garden–like demonstration site.