chapter eight

XHTML by Example: A Hybrid Layout (Part I)

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

Benefits of Transitional Methods Used in These Chapters

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

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

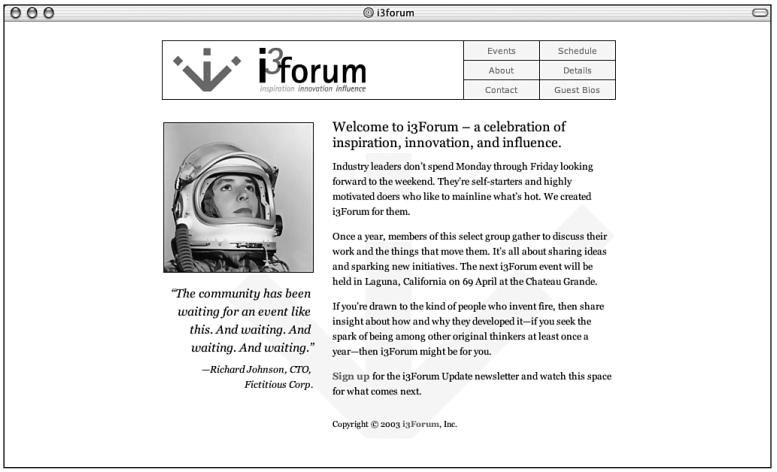
Style Sheets Instead of JavaScript

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

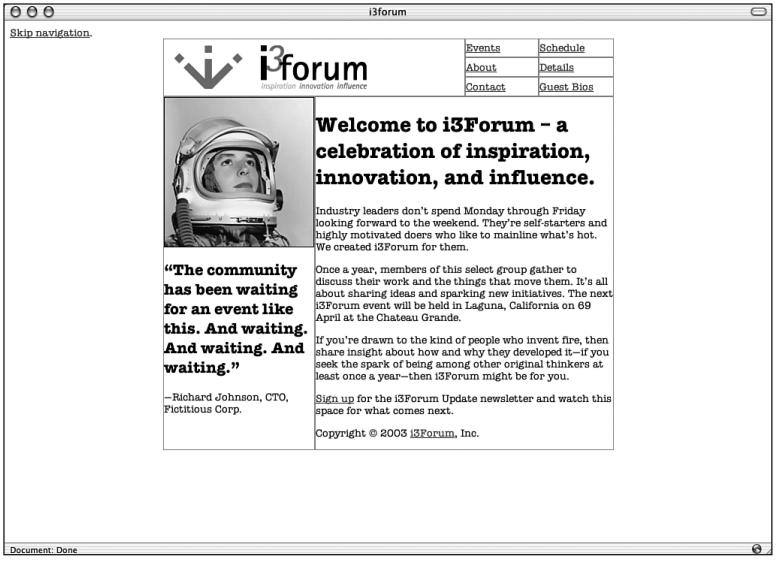
In this chapter, we'll nail down our markup. In Chapter 10, we'll add CSS to control the color, size, and relative positioning of elements. (We'll pause in Chapter 9 to learn CSS basics.) Among other things, the CSS we create will deliver menu rollover effects more commonly accomplished with images and JavaScript. There's nothing wrong with images or JavaScript, but by using XHTML text and CSS instead, we'll save bandwidth while making the site readily accessible to a wide variety of environments, including screen readers and text browsers, nondesktop-browsers (such as PDA and phone-based browsers), and non-CSS-capable browsers.

Basic Approach (Overview)

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



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



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

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

Separate Tables: CSS and Accessibility Advantages

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

- Streamline the CSS we'll create in Chapter 10
- Provide certain accessibility enhancements
- Structurally label each table according to the job it does, making it easier to some day revisit the layout and replace presentational XHTML tables with `div`s styled via CSS

The Table Summary Element

In addition, breaking the layout into two tables allows us to add a summary attribute to each:

```
<table id="nav" summary="Navigation elements" … etc. >
<table id="content" summary="Main content." … etc. >
```

The summary attribute is invisible to ordinary desktop browsers like IE and Netscape. But the screen-reading software used by non-sighted visitors understands the summary attribute and will read its value aloud. In our case, the screen reader will say "Navigation elements" and "Main content." Well-designed screen readers allow users to skip the table if they don't think it will interest them. Writing table summaries thus forms a good accessibility backup strategy to accommodate users who might miss the Skip Navigation link described two paragraphs from now.

Page Structure and *id*

We've assigned an `id` attribute value to each table according to the structural job it does—navigation or content. Doing so now allows us to later write compact CSS rules that apply to an entire table, avoiding classitis and divitis (defined and discussed in Chapter 7).

It also allows us to provide a Skip Navigation link in the top of our markup.

The What and Why of Skip Navigation

As its name implies, the Skip Navigation link allows visitors to bypass navigation and jump directly to the content table by means of an anchor link.

The id attribute whose value is "content" provides the anchor to which we link:

```
<div class="hide"><a href="#content" title="Skip navigation."
accesskey="2">Skip navigation</a>.</div>
```

Skipping navigation is not an urgent requirement for most sighted web users, who can focus their attention on particular parts of a web page simply by glancing at those parts and ignoring other parts that don't interest them.

But nonsighted visitors who are using screen readers experience the web in a linear fashion, one link at a time. It frustrates such users to endure a constant audio stream of menu links each time they load a page of your site. Skip Navigation lets these users avoid this problem.

Skip Navigation can also help sighted readers using non-CSS-capable PDA browsers and web phones avoid tediously scrolling through a fistful of links every time they load a new page. Finally, Skip Navigation can benefit sighted users who are physically impaired, although the method is not perfect. (See the later section titled "accesskey: Good News, Bad News.")

Skip Navigation and *accesskey*

Our Skip Navigation link enables visitors who are using nonvisual or non-CSS browsers to jump directly to content in the second table, whose id attribute name (and thus whose anchor link) is "content":

```
<table id="content" …> etc.
```

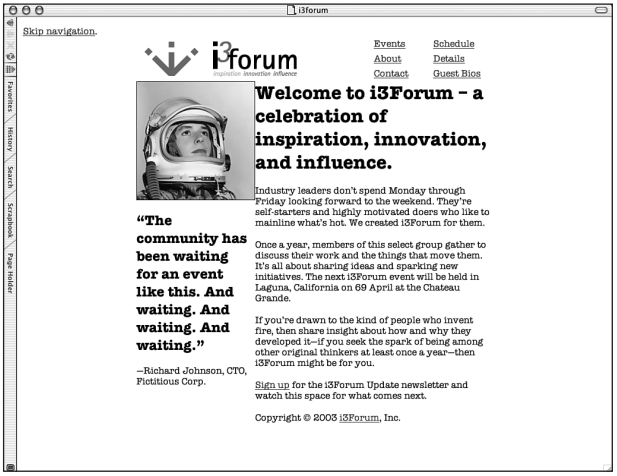
In these nonvisual or non-CSS environments, the link is readily available at the top of the page [**8.3**, **8.4**]. You'll create a rule to hide the Skip Navigation link in CSS-capable browsers in Chapter 10. (If you're the impatient type, we've also included it here.)

```
.hide    {
    display: none;
    }
```

Because of this CSS rule, visitors who are using modern browsers with CSS turned on will not see the Skip Navigation link—but most of them do not need to see it because they do not require Skip Navigation functionality for the reasons discussed in "The What and Why of Skip Navigation." Screen readers that ignore CSS will merrily read the content of the div, thus informing non-sighted visitors that they can avoid the boring recitation of the other links. (Alas, some screen readers obey CSS even though their users can't see it.)

8.3
In a non-CSS browser (or a CSS-capable browser with CSS turned off), the Skip Navigation link is clearly visible at the top of the page.

8.4
The visible Skip Navigation link in context—in our layout with CSS turned off.



There's an exception to every assumption, of course. A person who has impaired mobility, viewing the site via a CSS-capable browser, might desire to skip the navigation area and jump directly to content. Most web users who have impaired mobility can see an entire web page at once (the exception being users who are visually *and* physically impaired). But to navigate that page, impaired users employ the keyboard or an alternative, assistive input device. Tabbing their way past unwanted navigation links could be a nuisance, or worse.

How can we help these users skip navigation if they can't see the Skip Navigation link in their browser? We've provided that option via the `accesskey` attribute, which works even when the Skip Navigation link is invisible in the browser. Alas, the method is imperfect, as discussed next.

accesskey: Good News, Bad News

The `accesskey` attribute to HTML/XHTML enables people to navigate websites via the keyboard instead of a mouse. To assign an `accesskey` to an element, you simply declare it, as in the earlier XHTML excerpt, which we reprint here with the relevant attribute and value highlighted in bold:

```
<a href="#content" title="Skip navigation." accesskey="2">Skip
navigation</a>.
```

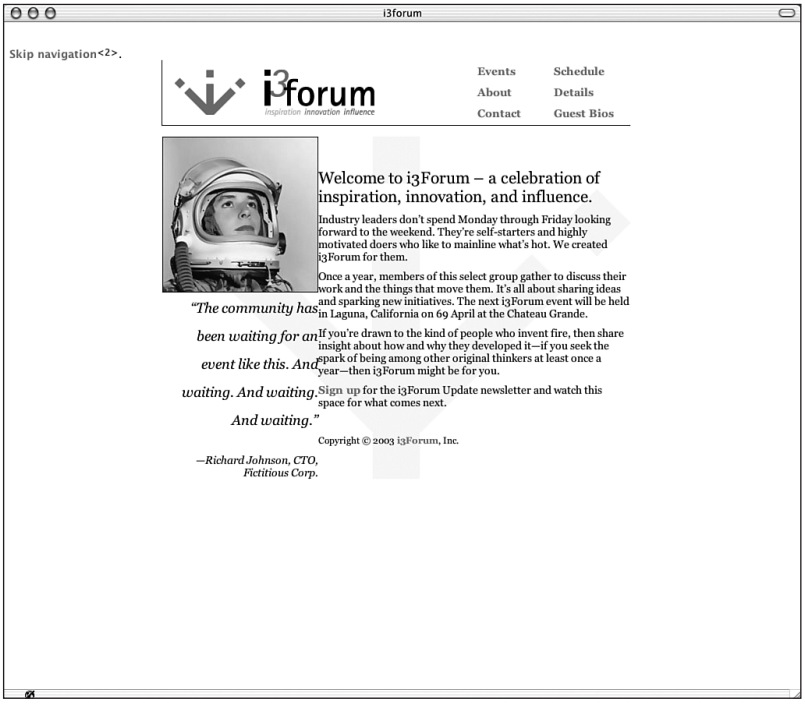
In our markup, we've assigned the Skip Navigation link an `accesskey` of 2. Therefore, to skip navigation, the visitor simply presses 2 on her keyboard. As is often true of accessibility enhancements, the required markup is easy to write and has no effect on the site's visual design. In this case, that's both good and bad.

For how does the visitor know to press 2 on her keyboard? No widely used browser displays `accesskey` letter assignments. Neither do most little-used browsers.

accesskey and iCab

As of this writing, only iCab [**8.5**], a Macintosh browser, visually displays `accesskey` letter assignments. Most web users are not Macintosh users, and most Macintosh users are not iCab users. Making matters worse, iCab cannot show the `accesskey` assignment when the Skip Navigation link is hidden via CSS. As this book goes to press, iCab still does not support much of CSS1, the W3C's first CSS recommendation, published way back in 1996. In short, although iCab is an interesting browser and its commitment to supporting HTML 4 is impressive (and no, we're not being facetious: iCab's HTML 4 support is superb), iCab is not going to solve the world's `accesskey` problem.

8.5
Of all the world's browsers, only iCab for Macintosh (`http://www.icab.de/`) displays our `accesskey` of 2, cuing the user that she can skip navigation by pressing 2 on her keyboard.



Two Utopian Possibilities for *accesskey*

Clearly, the majority of users who might benefit from `accesskey` have no way of knowing which `accesskey` letters or numbers to press; therefore, they cannot benefit from it. Because of that, including `accesskey` in your markup is somewhat idealistic.

If the W3C would recommend standard `accesskey` assignments for universal functions like "skip navigation" (and if designers and developers would follow those recommendations), users would always know which keys to press. That would be a good thing.

Alternatively, browser makers might decide to beef up their `accesskey` support by visually displaying `accesskey` values if the user decides to turn on this accessibility option in his preferences. IE for Windows provides an accessibility option allowing users to ignore font sizes on any web page. It might also add an option to Always Show Accesskey Values.

We must admit it feels rather Utopian to hope that the W3C will standardize `accesskey` shortcuts any time soon, and it also feels Utopian to hope that any

major browser vendor (let alone all of them) will devote engineering time and resources to an always-visible-`accesskey` option. Nevertheless, we continue to use `accesskey`. Some users might view source to see which `accesskey` values are in use on a page and thereafter use the appropriate keys to navigate. We hope things become easier for these users soon.

Additional *id* Attributes

In addition to the primary `id` attribute names (`nav` and `content`), in our first pass at the site's markup we also assign unique `id` attribute names to each cell of the navigation table. Two cells should suffice to make the method clear:

```
<td width="100" height="25" id="events"><a href=
"events.html">Events</a></td>
<td width="100" height="25" id="schedule"><a href=
"schedule.html">Schedule</a></td>
```

We also assign unique `id` attribute values to each of the two primary divisions of the content table, namely the sidebar (`id="sidebar"`) and primary content (`id="primarycontent"`) areas. Next, with much data removed for clarity, is the shell of the content table; `id` attributes and values have been highlighted in bold:

```
<table id="content" etc.>
<tr>
<td width="200" id="sidebar">
Sidebar content goes here.
</td>
<td width="400" id="primarycontent">
Primary content goes here.
</td>
```

For good measure, we slap an `id` attribute name on the secondary rows of the navigation bar. Thus, the *second* row of navigation "buttons" has the following `id` value:

```
<tr id="nav2">
```

And, as you might expect, the *third* row of navigation "buttons" has this `id` value:

```
<tr id="nav3">
```

How Much Is Too Much?

The latter two `id` attribute names (`nav2` and `nav3`) aren't required for this layout's purposes, but they might come in handy one day, in the event a redesign is required. Should we include them or not? Including them now adds a few bytes to our XHTML, and we might with equal merit have chosen not to do so.

If, on the final site, the navigation bar lives in a separate Server-Side Include file (or in a unique record managed by PHP, JSP, ColdFusion, or ASP), the client could easily edit that file at any point in the future, changing the entire site by adjusting a single file. If the client plans to use server-side technologies, it might be silly to include `nav2` and `nav3`. On the other hand, if no server-side technologies are used and the menu markup is manually repeated on every page, it might be safer to go ahead and include `nav2` and `nav3` to avoid potential search-and-replace errors in a future redesign. And that is what we've done.

First Pass Markup: Same as Last Pass Markup

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

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

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

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

Navigational Markup: The First Table

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

```
<body bgcolor="#ffffff">
<div class="hide"><a href="#content" title="Skip navigation."
accesskey="2">Skip navigation</a>.</div>
<table id="nav" summary="Navigation elements" width="600"
border="0" align="center" cellpadding="0" cellspacing="0">
<tr>
<td rowspan="3" id="home" width="400"><a href="/" title=
"i3Forum home page."><img src="images/logo.gif" width="400"
height="75" border=""0" alt="i3forum home" /></a></td>
<td width=""100" height="25" id="events"><a href="
"events.html">Events</a></td>
<td width="100" height="25" id="schedule"><a href=
"schedule.html">Schedule</a></td>
</tr>
<tr id="nav2">
<td width="100" height="25" id="about"><a href=
"about.html">About</a></td>
<td width="100" height="25" id="details"><a href=
"details.html">Details</a></td>
</tr>
<tr id="nav3">
<td width="100" height="25" id="contact"><a href=
"contact.html">Contact</a></td>
<td width="100" height="25" id="guestbios"><a href=
"guestbios.html">Guest Bios</a></td>
</tr>
</table>
```

Presentation, Semantics, Purity, and Sin

How big a standards geek are you? Did you spot the worst sin in our XHTML? The primary offense took place in the first line—namely the use of the outdated `bgcolor` (background color) attribute to the body element to specify, even in non-CSS browsers, that the page's background color should be white (`#ffffff`). Here it is again:

```
<body bgcolor="#ffffff">
```

Writing old-school markup like that could get us thrown out of the Pure Standards Academy faster than a greased meteor. After all, CSS lets us specify the body background color, and the W3C recommends that CSS, not HTML or XHTML, be used for this purpose. In the eyes of many standards fans, our use of `bgcolor` is a sin.

A Transitional Book for a Transitional Time

To the kind of standards geek who spends hours each week arguing about the evils of presentational markup on W3C mailing lists, what we've done here is evil and harmful. For that matter, we've also sinned by using tables as anything other than containers of tabular data, by specifying widths and heights in our table cells and by setting image margins to zero in markup. In fact, in the eyes of some, this entire chapter is sinful. Some standards geeks might not think much of this book, quite frankly. In their view, we should be telling you how to write semantic markup instead of letting you think it's okay to sometimes use tables for layout.

But the thing is, it is okay. Maybe it won't be okay some years from now, when designers use and browsers support purely semantic future versions of XHTML and rich future versions of CSS and SVG. But this is a transitional book for a transitional time. "Web standards" is not a set of immutable laws, but a path filled with options and decisions. In our view, people who insist on absolute purity in today's browser and standards environment do as much harm to the mainstream adoption of web standards as those who have never heard of or are downright hostile toward structural markup and CSS.

Making Allowances for Old Browsers

Why did we use the scarlet `bgcolor` attribute in spite of its shameful wickedness? The hybrid site we're producing makes no assumptions about the browsers used by its visitors. In an old, non-CSS-capable browser, if the default background color were set to any color other than white, the site's transparent GIF logo image would be afflicted by nonangelic halos caused by mismatched edge pixels. No client wants his logo to look shoddy in any browser, even if the rest of the site is just so-so in some old browsers.

One popular old browser that did not support CSS set medium gray as its default background color. Our logo is not antialiased against medium gray but against white. If we hadn't set the background color via the XHTML `bgcolor` attribute, our logo would look bad in such browsing environments.

In reality, you might not care what your site looks like in a 2.0 or 3.0 browser. For that matter, you might not care what it looks like in a 4.0 browser—neither might your boss or your client. The semantically impure techniques used in this chapter do not attempt to create the same visual experience in all browsers. In a non-CSS browser, our layout will not look any better than what you see in Figure 8.3. And that's okay.

We used tables for this site and included `bgcolor` to show you that such compromises can be made in XHTML 1.0 Transitional and the site will still validate. We also did this to suggest that any effort to include standards in your work—even a compromised (but realistic) effort that uses some presentational markup—is worth making.

Content Markup: The Second Table

The "content" table immediately follows the navigational one and should be self explanatory to anyone who's ever written HTML or XHTML. The two things worth noting are the compactness of this markup and its use of `id`:

```
<table id="content" summary="Main content." width="600"
border="0" align="center" cellpadding="0" cellspacing="0">
<tr>
<td width="200" id="sidebar" valign="top">
<img src="images/astro.jpg" width="200" height="200"
border="0" alt="i3Forum. Breeding leadership." title="i3Forum.
Breeding leadership." />
<h2>Subhead</h2>
<p>Text</p>
</td>
<td width="400" id="primarycontent">
<h1>Headline</h1>
<p>Copy.</p>
<p>Copy.</p>
<p>Copy.</p>
<p>Copy.</p>
<div id="footer">
<p>Copyright &copy; 2003 <a href="/" title="i3forum
home page.">i3Forum</a>, Inc.</p>
</div>
</td>
</tr>
</table>
</body>
```

In Chapter 9, we'll explore CSS basics. Then, in Chapter 10, we'll use CSS to add visual control and panache to our hybrid site.