



# XHTML: Give structure to content

By Charles Wyke-Smith

## TABLE OF CONTENTS

- 2 Sorry, IDWIMIE
- 3 The times they are a-changing
- 5 XHTML and how to write it
- 13 Document hierarchy:  
Meet the XHTML family

Stylin' with CSS is all about designing and building webpages that look stylish and professional, that are intuitive for visitors to use, that work across a wide variety of user agents (browsers, handhelds, cell phones, and so on) and whose content can be easily updated and distributed for other purposes.

Like any artist, your ability to achieve your creative vision is governed by your technical abilities. In this tutorial, I'm not going to wade into all the excruciating details that underpin the workings of the web, but I will say that without a certain degree of technical knowledge, you will never achieve your web design goals. So this tutorial is intended to give you the core technical information you need to realize your creative vision, and hopefully to give you the confidence to "go in" further as you build your skills. But most of all, we will focus on design; by this I mean design in its broadest sense, not just aesthetics (visual appearance), but also ergonomics (logical organization) and functionality (useful capabilities).

Everything in this tutorial is based on web standards, the rules of browser behavior defined in the recommendations of W3C (the World Wide Web Consortium), which all the major browser developers have agreed to follow. As you will see, browsers based on the Gecko engine—the current versions Mozilla/Firefox and Netscape—and those based on the Konquerer engine—including the excellent browser Safari for Mac—do a much better job delivering standards-based performance than the once-ubiquitous Microsoft® Internet Explorer, which fails to implement many CSS specifications.

A Sample XHTML Document

http://www.bbd.com/stylin/chi

Amazon eBay Yahoo! News PBS Kids

**Stylin'**  
the designer's guide to  
Cascading Style Sheets

a New Riders book by Charles Wyke-Smith

## MOVING TO XHTML

Creating XHTML compliant pages simply requires following a few simple rules. These rules may seem counter-intuitive or just a lot of extra work at first, but the benefits are significant and actually make coding sites much easier. Also, XHTML code can be easily validated online so you can be sure your code is correctly written.

Here are the key requirements for successful validation of your XHTML code.

1. Declare a DOCTYPE
2. Declare an XML namespace
3. Declare your content type
4. Close every tag, enclosing or non-enclosing
5. All tags must be nested correctly
6. Inline tags can't contain block level tags
7. Write tags in lowercase
8. Attributes must have values and must be quoted
9. Use encoded equivalents for left brace and ampersand

[more about these requirements](#)

Here are some useful links from the web site of the W<sup>3</sup>C (World Wide Web Consortium), the guiding body of the web's development.

- [W3C's XHTML validator](#)
- [W3C's CSS validator](#)

## Sorry, IDWIMIE

You might expect Internet Explorer to be the best browser, but, despite its current dominance, that is far from true. Frequently, I mention a CCS feature and tell you IDWIMIE—It Doesn't Work In Microsoft Internet Explorer. Internet Explorer is frozen in a non-standards past and, as a consequence, is losing market share rapidly. For some of Internet Explorer's shortcomings, there are workarounds known as hacks—the non-standard use of CSS to fool particular browsers into seeing or ignoring certain styles. It's tedious and time-consuming to create hacks; Internet Explorer must get with the program or continue to lose ground to its more compliant fellow browsers.

For us website designers and the visitors to the sites we create, web standards offer the prospect of sites displaying and behaving consistently in every browser, on every platform. We're not there yet, but the days of every browser supporting a different feature set, with all the resultant technical inconsistencies that can make cross-browser/cross-platform web development slow and frustrating, are it seems, almost over.

So, following the rules of web standards, Stylin' shows you how to publish content by defining its structure with XHTML and then defining its presentation with CSS.

1. **Content** is the collective term for all the text, images, videos, sounds, animations, and files (such as PDF documents) that you want to deliver to your audience.
2. **XHTML** (eXtensible HyperText Markup Language) enables you to define what each element of your content is. Is it a heading or a paragraph? Is it a list of items, a hyperlink, or an image? You determine this by adding XHTML markup to your content. Markup comprises tags (the tag name is enclosed in angle brackets < >) that identify each element of your content. You create an XHTML element (hereafter just called an element) by either surrounding a piece of content with an opening and a closing tag like this:

```
<p>This text content is defined by the tag as a paragraph</p>
```

Or, for content that is not text (an image, in this example), by using a single tag:

```

```

This tutorial focuses on XHTML and how to use it, but the most important thing to know right now is this:

XHTML defines a document's *structure*.

3. **CSS** (Cascading Style Sheets) enable you to define how each marked-up element of your content is presented on the page. Is that paragraph's typeface Helvetica or Times, is it bold or italicized? Is it indented or flush with the edge of the page? CSS controls the formatting and positioning of each of the content elements. To format the size of the text in a paragraph, I might write:

```
p {font-size: 12px;}
```

Which would make it 12 pixels high. Almost this entire tutorial is dedicated to teaching you CSS, but the most important thing to know right now is this:

CSS defines a document's *presentation*.

Providing a means of separating a document's structure from its presentation was the core objective in the development of web standards, and it is key to development of content that is both portable (can be displayed on multiple devices) and durable (ready for the future).

## The times they are a-changing

To get a sense of how far from standards compliance most of today's web sites are, we need to take a quick look back at HTML's development and examine the difficulties most sites face today.

As of early 2005, the typical website's markup is loaded with masses of presentational markup aimed at the capabilities of the browser for which it was written (such as Internet Explorer 5 for Windows, or the utterly obsolete Netscape 4.0). In today's world of information delivery, where you want to get your content out on not just today's standards-compliant browsers, but on cell phones, PDAs, and even the door of your viewers' refrigerators, you may be unpleasantly surprised to find that your website's content is tightly locked in a million yards of old presentational code.

### A legacy of kluges

HTML was originally intended to be used to lay out pages of text containing hyperlinks to other pages of text. It was not intended to enable complex brochure-like layouts. But as soon as the web broke out of academia and into the mainstream, that's exactly what designers wanted to do with it. And the kluges abounded.

For example, if a photographic image was considered too close to the edge of the page, a designer might use a 1-pixel-square transparent GIF image, "force-sized" with a width attribute to be much larger so that it could invisibly shove the image out into the page.

Tables were also used in creative ways. Tables are an HTML element designed for laying out grids of data (like an Excel spreadsheet), but if you need to divide a page into say, a header, navigation, and content areas, then you might use a table to divide up the page, and you would drop each piece of content into a different cell of the table. Basically, it got to the point where table-based design was almost a standard in itself and was taught as good practice in untold numbers of web books.

When it comes to (ab)using HTML, I know of what I speak; I did these things for years myself, along with the rest of the world's web developers—there was no other way.

Although the end justified the means, there was an unforeseen, or simply ignored, consequence that now affects almost every website: the content of the world's webpages is crammed with markup that is only there to achieve the desired presentation on a small group of soon-to-be obsolete web browsers; it provides no actual information at all. It's safe to say that most webpages are over 60 percent presentational markup, and the result is that the actual content is almost impossible to extract for other uses.

And while we are on this sad subject...

### Let's play spot the content!

Take this snippet of markup from the Microsoft home page, July 1, 2004:

```
<table cellpadding="0" cellspacing="0" width="100%"
height="19" border="0" ID="Table5">

<tr>

<td nowrap="true" id="homePageLink"><</td>

<td><span class="ltsep">|</span></td>

<td class="lt0" nowrap="true" onmouseenter="mhHover
('localToolbar', 0*2+2, 'lt1')" onmouseleave="mhHover
('localToolbar', 0*2+2, 'lt0')"><a href="http://go.microsoft.com/
?LinkId=508110">MSN Home</a></td>

<td><span class="ltsep">|</span></td>

<td class="lt0" nowrap="true" onmouseenter="mhHover
('localToolbar', 1*2+2, 'lt1')" onmouseleave="mhHover
('localToolbar', 1*2+2, 'lt0')"><a href="http://go.microsoft.com/
?linkid=317769">Subscribe</a></td>
```

```

<td><span class="ltsep">|</span></td>

<td class="lt0" nowrap="true" onmouseenter="mhHover
('localToolbar', 2*2+2, 'lt1')" onmouseleave="mhHover
('localToolbar', 2*2+2, 'lt0')"><a href="http://go.microsoft.com/
?linkid=317027">Manage Your Profile</a></td>

<td width="100%"></td>

</tr>

</table>

```

All of this code produces just one row of buttons on this page (Figure 1):

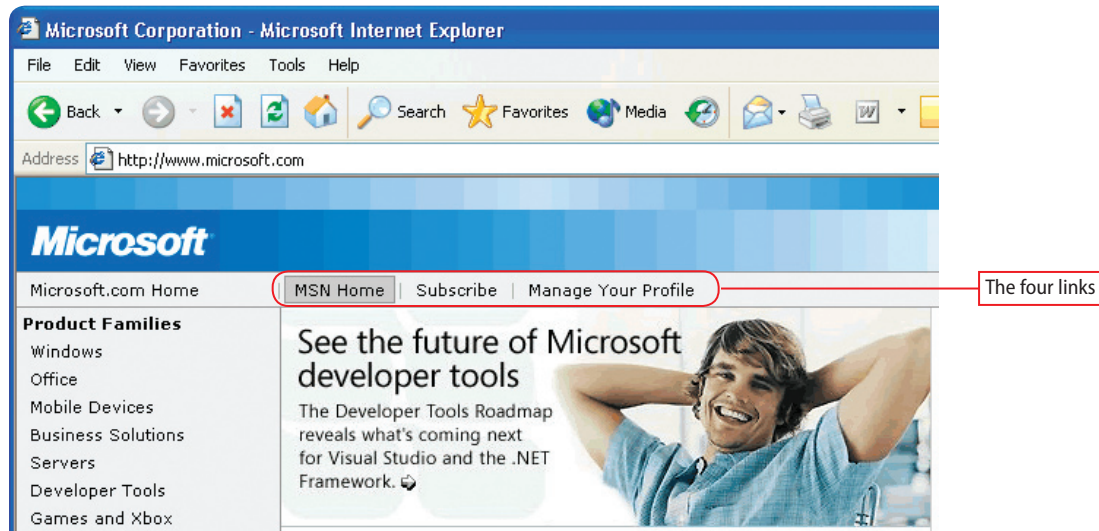


Figure 1: All of the code above generates just the three links seen in the row below the Microsoft logo.

The total *content* of the code is 247 characters out of 956, or less than 26 percent. The remaining 74 percent is just gooey chocolate sauce. Except for the `href` attributes, everything inside the tags is presentation and could all be ripped out and converted into a few brief definitions in a style sheet. The table is not used to display data; its purpose is solely to line everything up. The rest of the code is mostly concerned with making rollovers work. Each link requires a class to identify it to JavaScript, a forced `nowrap` attribute to keep the words on the link together, and two JavaScript function calls—yeah, on *every* link. (As an aside, rollovers are easy to create with CSS and require two simple CSS styles.) Note also that a table cell that contains a nested span with a class is required to display each tiny vertical line between the links.

Please don't think I'm picking on Microsoft—almost any site you might choose is equally challenged.

Today, with browsers and many other devices standardizing around XHTML and CSS, non-compliant websites are finding that it is difficult to deliver their existing content on these newer devices and browsers. Have you seen your home page on a handheld computer lately?

### The future will be here soon

So let's leave our past, redolent with bloated markup and nested tables, far behind, and look forward. Although bringing your current websites into the modern age may take a substantial amount of work, you can console yourself that by following the new web standards, you can do it once and do it right. If you are starting a new site, you can do it right first time.

In *Stylin'*, you learn to future-proof your site by separating the content from the presentation; you do this by creating pages of XHTML markup with only content in them, and then, using a single line of code, you link these pages to a separate file called a style sheet, which contains the presentation rules that define how the markup should be displayed.

The power of this church-and-state separation is this—you can have different style sheets for browsers, for PDAs, for cell phones, for printing, for screen readers for the visually impaired, and so on; each style sheet causes the content to be presented in the best possible way for that use, but you only ever need *one* version of the XHTML content markup. As you will see, an XHTML page can automatically select the correct style sheet for each environment in which it finds itself. In this way, your write-once, use-many content becomes truly portable, flexible, and ready for whatever presentational requirements the future may bring its way. Note, however, that like any great vision of the future, there are still some current realities that we need to deal with.

## XHTML and how to write it

Because CSS is a mechanism for styling XHTML, you can't start using CSS until you have a solid grounding in XHTML. And what, exactly, is XHTML? XHTML is a reformulation of HTML as XML—did you get that? Put (very) simply, XHTML is based on the free-form structure of XML, where tags can be named to actually describe the content they contain; for example, `<starname>Cher</starname>`. This very powerful capability of XML means that when you develop your set of custom tags for your XML content, you also must create a second document, known as a *DTD* (document type definition) or a similarly formatted XML schema, to explain to the device that is interpreting the XML for how to handle those tags.

XML has been almost universally adopted in business, and the fact that the same X (for eXtensible) is now in XHTML emphasizes the unstoppable movement toward the separation of presentation and content.

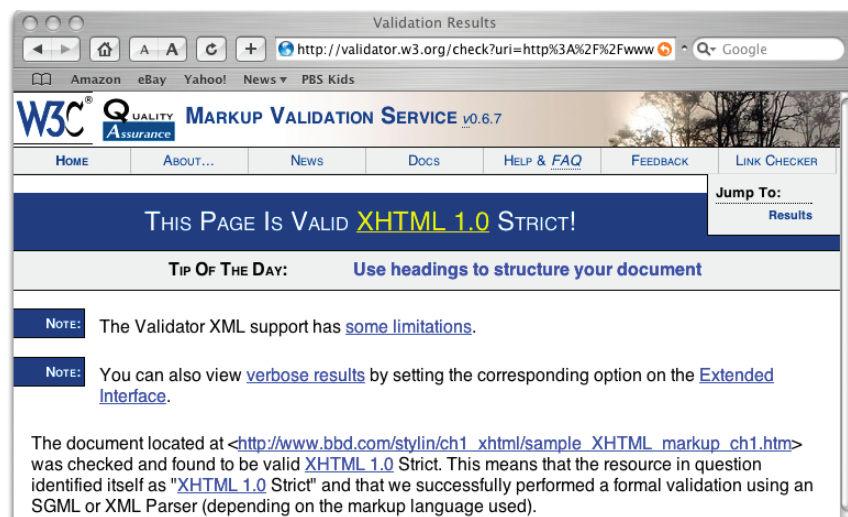
The rest of this tutorial is dedicated to the latest, completely reformulated, totally-modern, and altogether more flexible version of HTML. Ladies and gentlemen, please welcome XHTML!

### XHTML markup rules

Correctly written XHTML markup gives you the best chance that your pages will display correctly in a broad variety of devices for years to come. The clean, easy-to-write, and flexible nature of XHTML produces code that loads fast, is easy to understand when editing, and prepares your content for use in a variety of applications.

You can easily determine if your site complies with web standards—if your markup is *well-formed and valid XHTML*, and your style sheet is *valid CSS*, then it will comply. (Whether it's well designed or not is a rather more subjective matter, but we will consider that as we go along.)

Well formed means that the XHTML is structured correctly, according to the markup rules described in this tutorial. Valid means the markup contains only XHTML, with no meaningless tags, tags that are not closed properly, or deprecated (phased out, but still operational) HTML tags. You can check to see if your page meets these criteria by uploading the page onto a server and then going to <http://validator.w3.org> and entering the page's URL. Press Submit, and in a few seconds you are presented with either a detailed list of the page's errors or the very satisfying "This Page Is Valid XHTML!" message (*Figure 2*). CSS can be validated in the same way at <http://jigsaw.w3.org/css-validator>.



**Figure 2:** If your site complies with web standards, you'll get the ever-gratifying This Page Is Valid XHTML message from the W3C validator.

Here's the complete (and mercifully, short) list of the coding requirements for XHTML compliance:

1. **Declare a DOCTYPE.** The DOCTYPE goes before the opening html tag at the top of the page and tells the browser whether the page contains HTML, XHTML, or a mix of both, so that it can correctly interpret the markup. There are three main DOCTYPEs that let the browser know what kind of markup it is dealing with:

**Strict:** All markup is XHTML compliant.

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

**Transitional:** This states that markup is a mix of XHTML and deprecated HTML. Many sites are currently using this one, so their old HTML code works as well (in this context, "as well" means "also" rather than "equally well") as the XHTML they are now adding.

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

**Frameset:** This is the same as transitional but in this case frames, which are deprecated under XHTML, are OK too.

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

It is important to specify a DOCTYPE. Browsers that don't see a DOCTYPE in the markup assume that the site page was coded for browsers developed long before web standards.

Without a DOCTYPE, many browsers go into what is known as *Quirks mode*, a backwards-compatibility feature supported by Mozilla, Internet Explorer 6 for Windows, and Internet Explorer 5 for Macintosh. In Quirks mode, the browser functions as if it has no knowledge of the modern DOM (document object model), and pretends it has never heard of web standards. This ability to switch modes depending on the DOCTYPE, or lack thereof, enables browsers to do the best possible job of interpreting the code of both standards-compliant and non-compliant sites.

Note that for some weird reason, the DOCTYPE tag does not need to be closed with a backslash and DOCTYPE is always in caps. This entirely contradicts XHTML rules 4 and 7 below. Go figure.

2. **Declare an XML namespace.** Note this line is your new html tag. Here's an example:

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

When a browser is handling an XHTML page and wants to know what's in the DTD, which lists and defines all the valid XHTML tags, here's where it can find it—buried away on the servers of the WC3.

In short, the DOCTYPE and namespace declarations ensure that the browser interprets your XHTML code as you intended.

3. **Declare your content type.** The content type declaration goes in the head of your document, along with any other meta tags you may add. The most common is

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

This simply states what character coding was used for the document. ISO-8859-1 is the Latin character set, used by all standard flavors of English, so if you are coding for an audience who uses the alphabet instead of, for example, Chinese or Farsi characters, this is the one you need. If your next site is going to be in Cyrillic or Hebrew, you can find the appropriate content types on Microsoft's site (<http://msdn.microsoft.com/workshop/author/dhtml/reference/charsets/charset4.asp>).

#### 4. Close every tag, whether enclosing or nonenclosing.

Enclosing tags have content within them, like this:

```
<p>This is a paragraph of text inside paragraph tags.  
To be XHTML compliant, it must, and in this case does,  
have a closing tag.</p>
```

Non-enclosing tags do not go around content but still must be closed, using space-slash at the end, like this:

```

```

#### 5. All tags must be nested correctly.

If a tag opens before a preceding one closes, it must be closed before that preceding one closes. For example:

```
<p>It's <strong>very important</strong> to nest tags  
correctly.</p>
```

Here, the strong tag is correctly placed inside the <p>; it closes before the containing p tag is closed. A tag enclosed inside another in this way is said to be nested.

This is wrongly nested:

```
<p>The nesting of these tags is <strong>wrong<p></strong>
```

Multiple elements can be nested inside a containing element; a list nests multiple li elements inside a single ul or ol element, like this:

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
</ul>
```

Because CSS relies on proper nesting in order to target styles to elements, you have to get this right or your code won't validate.

#### 6. Inline tags can't contain block level tags.

Block-level tags are tags that provide visual structure to your document, such as `p` (paragraph) and `div` (division). Block-level elements stack on top of one another on the page—if you have two paragraphs, the second paragraph appears by default under the previous one; no line breaks are required. By contrast, inline tags, such as `a` (anchor, a hyperlink) and `em` (emphasis, usually displayed as italics) occur in the normal flow of text, and don't force a new line.

Remember that if you nest a block element, such as a paragraph `p`, inside an inline element, such as a link `a`, your code won't validate.

Also, some block-level elements can't contain other block-level elements either; for instance, a `h1-6` (heading) tag can't contain a paragraph. Besides using validation, you can let common sense be your guide to avoid these problems; you wouldn't put an entire paragraph inside a paragraph heading when you were writing on paper or in Word, so don't do illogical things like that in your XHTML either, and you won't go far wrong.

#### 7. Write tags entirely in lowercase.

Self-explanatory—no capital letters at all. I've always done this myself, but if you haven't, the days of `P` are over; now it has to be `p`. Sorry.

8. Attributes must have values and must be quoted. Some tags' attributes don't need values in HTML, but in XHTML, all attributes must have values. For example, if you previously used the `select` tag to create a drop-down menu in HTML and wanted to choose which item showed by default when the page loaded, you might have written something like this:

```
<SELECT NAME=ANIMALS>
<OPTION VALUE=Cats>Cats</OPTION>
<OPTION VALUE=Dogs SELECTED>Dogs</OPTION>
</SELECT>
```

Which would have given you a drop-down menu with Dogs displayed by default.

The equivalent valid XHTML is this:

```
<select name="animals">
<option value="cats"></option>
<option value="dogs" selected="selected">Dogs</option>
</select>
```

Note that in this revised version, all the attribute names are in lowercase and all the values are in quotes.

#### 9. Use the encoded equivalents for a left angle bracket and ampersand within content.

When XHTML encounters a left angle bracket, `<` (also known as the less-than symbol), it quite reasonably assumes you are starting a tag. But what if you actually want that symbol to appear in your content? The answer is to encode it using an entity. An entity is a short string of characters that represents a single character; when used, this string causes XHTML to interpret and display the character correctly and not to confuse it with markup. The entity for the left angle-bracket/less-than symbol is `&lt;`;—remember LT stands for less than.

Entities not only help avoid parsing errors like the one just mentioned, but they also enable certain symbols to be displayed at all, such as `&copy;` for the copyright symbol (©). Every symbolic entity begins with an ampersand (&) and ends with a semicolon (;). Because of this, you probably aren't surprised to find out that XHTML regards ampersands in your code as the start of entities, and so you must also encode ampersands as entities when you want them to appear in your content; the ampersand entity is `&amp;`.

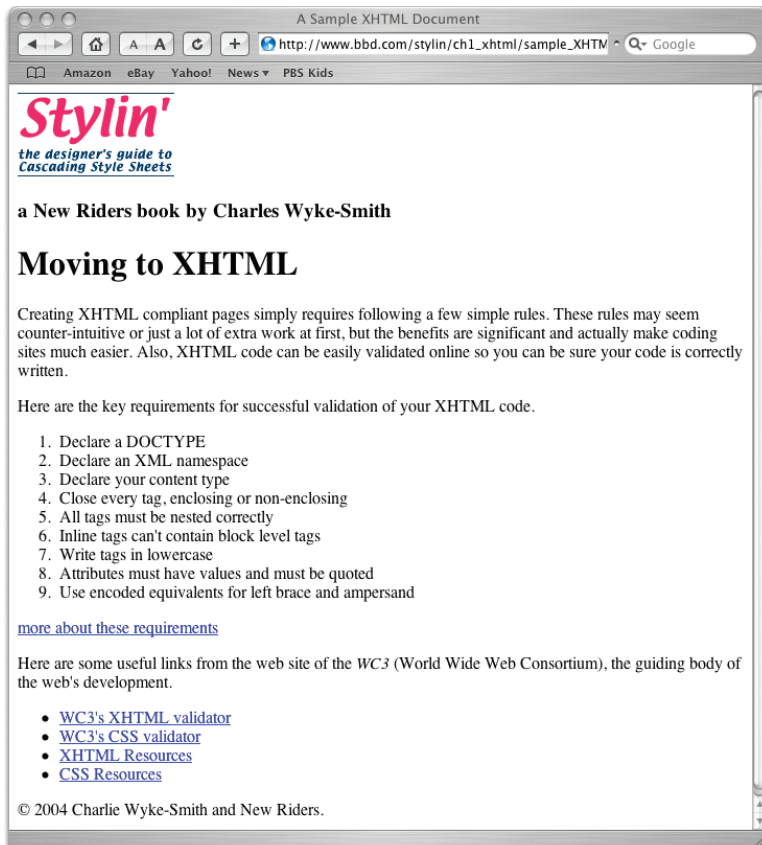
A good rule of thumb is that if a character you want to use is not printed on the keys of your keyboard (such as é, \*, ©, or £), you need to use an entity in your markup.

There are some 50,000 entities total, which encompass the character sets of most of the world's major languages, but you can find a shorter list of the commonly used entities at the Web Design Group site ([www.htmlhelp.com/reference/html40/entities](http://www.htmlhelp.com/reference/html40/entities)).

And those are the rules of XHTML markup; they are relatively simple, but you must follow them exactly if you want your pages to validate (and you do).

## Understanding markup

Here is a sample unstyled but valid XHTML page that illustrates the rules of XHTML (*Figure 3*):



**Figure 3:** This unstyled but valid XHTML isn't visually interesting, but it is definitely usable.

The page isn't pretty, but it is certainly usable. And, this page's markup is lean and simple. There is no presentational code, and this XHTML passes muster with the WC3 HTML validator.

Now let's get into more detail on the XHTML rules by taking a look at the markup that created the page shown in *Figure 3* line by line.

### Lines 1 – 2

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

Here the DOCTYPE is set to XHTML 1.0 Strict. In this case, you're indicating that code will be interpreted as pure, non-backward-compatible XHTML.

I focus on the strict DOCTYPEs throughout this tutorial, which means I do not use any deprecated HTML. If you need to support deprecated HTML tags such as frames, you need a different DOCTYPE (see "XHTML Markup Rules," #3 earlier in this tutorial).

### Line 3

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

Next is the opening `html` tag, which did not have attributes in the past. Now it has a URL that points to the namespace (the collection of XML declarations and attributes) of this document.

As mentioned earlier in "XHTML Markup Rules", the DOCTYPE and namespace declarations ensure that the browser understands what flavor of (X)HTML you are using, so it interprets your code as you intended.

#### Line 4

```
<head>
```

This tag opens the document head. The head of your document, which is sandwiched between the `head` and `/head` tags, contains information that, with the exception of the title, is not displayed to the viewer. Besides the essential `head` tags I list next (Lines 5–9), optionally there can be others: `meta` tags can contain all kinds of information (page descriptions, keywords, author names, etc.) used by search engines and other indexing software that might visit your site.

There can also be style tags that contain JavaScript and CSS that relate to, and can only be used by, the page they are on.

#### Line 5

```
<title>A Sample XHTML Document</title>
```

Technically, you don't *have* to use a `title` tag for your page to validate, but if you don't add it, the validator will encourage you to add it.

#### Lines 6 – 7

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

These two required meta head tags provide information that helps the browser and server properly manage and display the page.

XHTML insists that you provide character encoding information, which ensures that the browser is displaying the pages with an appropriate character set. Here, in the first `meta` tag, 8859-1 is the code for Latin-1, the alphabet and associated symbols used in writing English and some other languages (see “XHTML Markup Rules,” #3 earlier in this tutorial). Note that as nonclosing tags, they are both closed with the space-slash-angle bracket construction.

Language information is also required. In the second `meta` tag, I state that the language is U.S. English; a language type such as Chinese causes the browser to display text from right to left.

#### Lines 8 – 9

```
<link href="demo_styles.css" rel="stylesheet" type="text/css" />
</head>
```

The `link` tag links the XHTML markup to a CSS style sheet, which is a separate file located using the `href`. The `link` tag isn't required, but linking is how you relate a style sheet to your markup, and by adding the same style sheet link to each page of your site, you can enable the pages to all share the same set of styles. You can also use the `@import` tag to link to a style sheet.

Make sure you close the document head using the `/head` tag.

#### Line 10

```
<body>
```

Start the document body. The body contains the content that displays on your page.

#### Line 11

```
<!--header-->
```

This is a comment. It is not displayed; it is just here to make the code more understandable.

Note that in XHTML you can only use two dashes, instead of the unlimited number allowed by HTML, at the start and end of each comment.

## Line 12

```
<div id="logo"> 
```

Divs divide the page into rectangular, box-like areas. These areas are invisible unless you turn their borders on or color their backgrounds. This `div` tag has an `id` attribute with the value of `"logo"`; you can use this ID name to target CSS styles at this div to set its position, size, background color, and much more; furthermore, the div allows you to position all the content within it as a group and target styles at each of the tags it contains.

The logo image tag (`img`) is a nonclosing element and is therefore closed with a slash before the closing brace. Note the `alt` tag, which displays if the graphic doesn't load or is spoken by a screen reader. You must use `alt` tags on every image, even if the value is `""` (that is, two quotes with nothing, not even a space, in between). Only do this if the image serves no informational purpose. You can leave the `alt` tags blank on everything, but such tags will be flagged by an XHTML validator. Also, this isn't very user friendly and does not aid accessibility. Note that all attribute values (such as the `150` and `80` in this example) must now be in quotes. Yes, really.

## Lines 13 – 15

```
<h3>a New Riders book by Charles Wyke-Smith</h3>
</div>
<!--end header-->
```

A size 3 text heading is a block-level element and therefore it occurs on a new line, or more precisely, under the previous element. No `br /` tags are required.

```
</div>
```

Remember to close the header division using the `/div` tag and make a comment that the header ends here.

## Lines 16 – 20

```
<!--main content-->
<div class="contentarea">
  <h1>Moving to XHTML</h1>
  <p>Creating XHTML compliant pages simply requires following a few simple rules. These rules may seem counter-intuitive or just a lot of extra work at first, but the benefits are significant and actually make coding sites much easier. Also, XHTML code can be easily validated online, so you can be sure your code is correctly written.</p>
  <p>Here are the key requirements for successful validation of your XHTML code.</p>
```

Now, the content area starts with a `div`, which is a block-level element. The main header is size 1 text. Next, are two paragraphs. Paragraph tags, like all enclosing tags, must be closed with a backslash tag; in this case, `/p`. Note that paragraphs are block-level elements and have a default amount of space around them, top and bottom.

## Lines 21 – 31

```
<ol>
  <li>Declare a DOCTYPE.</li>
  <li>Declare an XML namespace.</li>
  <li>Declare your content type.</li>
  <li>Close every tag, enclosing or non-enclosing.</li>
  <li>All tags must be nested correctly.</li>
```

```

<li>Inline tags can't contain block-level tags.</li>
<li>Write tags in lowercase.</li>
<li>Attributes must have values and must be quoted.</li>
<li>Use encoded equivalents for left brace and ampersand.</li>
</ol>

```

This is an ordered list; each list item has a number by default. (Unordered lists (`ul`) have bullets by default rather than numbers).

#### Line 32

```
<a href="more.htm">more about these requirements</a>
```

This is a hyperlink to a page named `more.htm` in the same folder as the current page.

#### Lines 33 - 34

```

</div>
<!--end main content-->

```

This closes the content area `div`. The comment is, of course, optional.

#### Lines 35 - 37

```

<!--navigation-->
<div id="navigation">
  <p>Here are some useful links from the web site of the <acronym
title="World Wide Web Consortium">WC3</acronym> (World Wide Web
Consortium), the guiding body of the web's development.</p>

```

It's good practice to style acronyms in a way that differentiates them from the text around them. Internet Explorer does not provide any default styling for acronyms; Safari will put them in italics (such as in Figure 1.3). If you add a `title` tag to an acronym, a tool tip containing the text from the title attribute pops up when a user mouses over it. It's also good practice to indicate the tool-tip's availability by underlining the acronym with a dotted line; this is achieved by styling the acronym element with a dotted `border-bottom`. Don't make the underline solid, which by convention would indicate the text is a link. These same markup techniques can also be applied to the `abbr` (abbreviation) tag.

#### Lines 38 - 45

```

<ul>
  <li><a href="http://validator.w3.org">WC3's XHTML
validator</a></li>
  <li><a href="http://jigsaw.w3.org/css-validator/">WC3's CSS
validator</a></li>
  <li><a href="http://www.w3.org/MarkUp/">XHTML Resources</a></li>
  <li><a href="http://www.w3.org/Style/CSS/">CSS Resources</a></li>
</ul>
</div>
<!--end navigation-->

```

This navigation aid is constructed as a list in which each list item is a link. All of this is inside a `div` block with an ID that enables you to reference it accurately from the style sheet. Note that there is no line break (which, for you purists, is purely presentational markup) at the end of each link; none is needed. By default, links appear in a row because they are inline elements, but here, because they are contained within list items, which are block-level elements, they display stacked.

#### Lines 46 - 50

```
<!--footer-->
<div id="homepagefooter">
  <p>&copy; 2004 Charlie Wyke-Smith and New Riders.</p>
</div>
<!--end footer-->
```

The last element of the page is a `div` that contains the footer text inside a paragraph tag.

#### Lines 51 – 53

```
</body>
</html>
<!--end of sample doc-->
```

Now you just close out the body and the page, and you're done. Any questions? No? Good! Moving right along...

## Document hierarchy: Meet the XHTML family

OK, the *document hierarchy* is one more important concept you need to understand before you can get to CSS. The document hierarchy is like a family tree or an organizational chart based on the nesting of a page's XHTML tags. A good way to learn to understand this concept is to take a snip of the body section of the markup we just discussed and strip out the content so that you can better see the organization of the tags. Here's the stripped-down header:

```
<body>
  <!--header - this is just a comment, not code-->
  <div id="logo">
    <img />
    <h3> </h3>
  </div>
  <!--end header - remaining tags removed here for
  clarity-->
</body>
```

Now you can clearly see the relationships of the tags; for example, in the markup, you can see that the `body` tag contains (or nests) all the other tags. You can also see that the `div` tag (with the ID of "logo") contains two tags; an image tag and head 3 tag.

Figure 4 shows another way to represent this structure—with a hierarchy diagram.

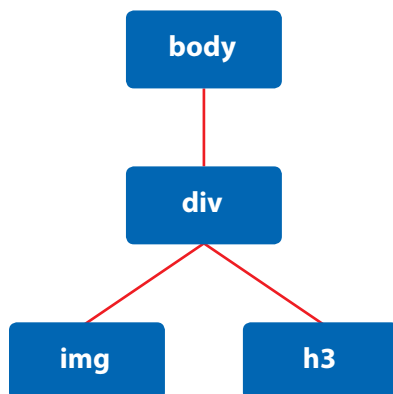


Figure 4: You can clearly see the hierarchical structure in this diagram.

When examining this hierarchical view, we can say that the both the `img` tag and the `h3` tag are the children of the `div` tag, because it is the containing element of both. In turn, the `div` tag is the parent tag of both of them, and the `img` tag and the `h3` tag are siblings of one another because they both have the same parent tag. Finally, the `body` tag is an ancestor tag of the `img` and `h3` tags, because they are indirectly descended from it. In the same way, the `img` and `h3` tags (and the `div`, for that matter) are descendants of the `body` tag.

To quote Sly Stone: “It’s a family affair...”

In CSS, you write a kind of shorthand based on these relationships; for example

```
div#logo img {some CSS styling in here}
```

Such a CSS rule only targets `img` tags inside of (descended from) the `div` with the ID of “`logo`” (the `#` is the CSS symbol for an ID). This rule means “any image that is descended from the `div` with an ID of “`logo`”; other `img` tags in the page are unaffected by this rule because they aren’t contained within the “`logo`” `div`. In this way, you can add a border around just this image or set its margin to move it away from surrounding elements.

The important concept to understand is that every element within the body of your document is a descendant of the `body` tag, and, depending on its location in the markup, the element could be an ancestor, a parent, a child, or a sibling to other tags in the document hierarchy.

By creating rules that use (and often combine) references to IDs, classes, and the hierarchy structure, you have means by which you can accurately dictate which CSS rules affect which XHTML elements.

#### ABOUT THE AUTHOR

Charles Wyke-Smith has been creating websites since 1994. He currently runs a web consulting business in Napa Valley, California, focusing on user experience, information architecture, and interface design. In addition to being the former vice president of web development for eStar.com, a celebrity information site, Wyke-Smith has worked as a web design consultant for Wells Fargo, ESPN Videogames, and The University of San Francisco. An accomplished speaker and instructor, Wyke-Smith has also taught multimedia and interface design and has spoken at many industry conferences.

#### FOR MORE INFORMATION

Visit: [www.adobe.com/designcenter/main.html](http://www.adobe.com/designcenter/main.html)

Better by Adobe.™

**Adobe Systems Incorporated**  
345 Park Avenue, San Jose, CA 95110-2704 USA  
[www.adobe.com](http://www.adobe.com)

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Apple is a trademark of Apple Computer, Inc., registered in the United States and other countries. Intel is a registered trademark of Intel Corporation in the U.S. and other countries.

© 2006 Adobe Systems Incorporated. All rights reserved.  
Printed in the USA 3/06

