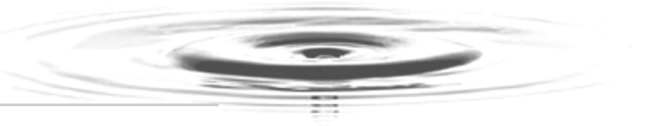
Excerpt from "Spring into HTML and CSS" by Molly E. Holzschlag (Addison-Wesley Professional)



CHAPTER 1

Building an HTML Page

Web documents are meant to be constructed logically. You must have certain elements in place for your document to conform and validate. Conformance means that the document actually follows the language and language version in which it is being written. Validation is the technical process by which we test conformance, allowing us to find errors and fix mistakes.

The first thing you can do to make absolutely certain that your pages are given the best fighting chance at conformance is to begin with all the required and structural elements that are needed *before* you begin to add text and other content.

Ironically, it's only been in hindsight that the majority of people working on websites have improved upon the way they use markup. The Web was in such a state of evolutionary, rapid growth that new elements and features were being added to browsers—and HTML—all the time. Many of these features made it into the actual specifications, but many did not. What's more, elements of HTML pages that should have been included from the beginning have often been left out, even by professionals.

How is this possible, you might wonder? Well, the primary piece of software used to interpret HTML is the desktop web browser. These browsers have a long history of forgiving errors. Of course, they also have a long history of introducing errors! Browsers have been both the blessing and the curse of the Web because they have allowed for innovation but often spent more time adding fun features rather than basic support for the languages they are meant to support. As a result, the Web is a mishmash of HTML use—most of it not conforming or valid—and, in light of this chapter's discussion, many times authored without the basic structural components required by the language.

A movement is afoot to bring better standards to browsers, to the tools that people use to develop websites, and to those of us interested in creating pages that not only work, but work well, regardless of whether our goals are personal or professional.

In this chapter, you learn to create a template that will serve as the foundation for everything you do in this book. This template will contain all the necessary and helpful technical and structural bits that form the basis of a document that will conform and validate, too.

Declaring and Identifying the Document

The first thing you'll want to do in your page is add a bit of code that declares which type of document you're using and identifies the language version. This is done with Standardized General Markup Language (SGML), which is the parent language to HTML and appears in this important declaration, known as the *DOCTYPE declaration*. This declaration is a unique piece of code, and a suitable declaration must be used in every document you create.

Example 1-1 shows the DOCTYPE declaration we'll be using in all examples in this book:

EXAMPLE 1-1 The DOCTYPE declaration for XHTML 1.0 Transitional

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

Look a little weird? Not to worry. I'll go through it with you so you have a firm understanding of what each bit of this declaration means. First, there's the opening `<!`, which many readers who have looked at HTML before might wonder about. The angle bracket is a familiar component in HTML, but the exclamation mark appears in only one other situation with HTML: in comments, which you'll also learn about in this chapter. This symbol isn't used too often because it's SGML syntax being used in the context of HTML. Here, it simply means that a declaration is about to begin. This is then followed by the term `DOCTYPE`, which states that this code is declaring the *document type*.

The next bit is `html`, which defines this document type as being written in HTML. Note that it's in lower case here. This is significant because we're using XHTML—and because XHTML is case sensitive, this particular part of the declaration must be in lower case. If it's not, your document will not validate. The word `PUBLIC` is an important piece of information. This means that the particular document type being referenced is a public document. Many companies create unique versions of XHTML, with customized elements and attributes. For our purposes, the public version of HTML that we're going to use is absolutely sufficient.

The ensuing syntax `"-//W3C//DTD XHTML 1.0 Transitional//EN"` defines the host of the document's language type and version (The World Wide Web Consortium, W3C), and states that the document is being written according to the XHTML 1.0 Transitional Document Type Definition (DTD). A DTD is simply a long laundry list of allowed elements and attributes for that language and language version. Finally, there's a complete URL that goes to the DTD, `"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"`. If you were to load this into your browser, you'd see the actual DTD, for XHTML 1.0 Transitional (see Figure 1-1).

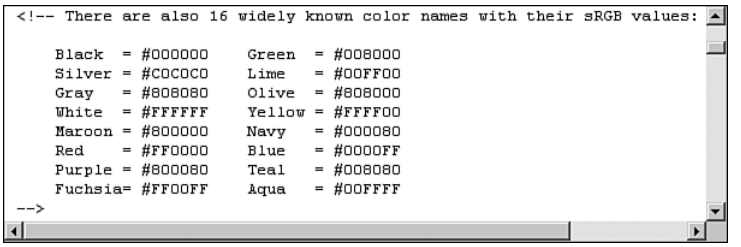


FIGURE 1-1 A portion of the XHTML Transitional DTD.

With this declaration at the top of your document, you will be able to author the document and then run it through a validator to check for conformance. The validator uses the information in the declaration and compares your document to the DTD you've declared. If you've followed the rules allowed in the DTD you've declared here, you should have no errors whatsoever, which is, of course, our goal.

QUANTUM LEAP

Due to discrepancies in the way that many browsers were handling various aspects of HTML and CSS, a means of gaining better performance for those documents written to specification became evident. Tantek Celik, then a developer for Microsoft, created a switching mechanism in IE that corrected numerous problems. This switch uses properly formed DOCTYPE declarations to switch the browser from "quirks" mode (the forgiving mode I described in this chapter's introduction) to "compliance" mode, which allows sites written in compliant markup and CSS to perform much more efficiently. One important point: Never place anything above a DOCTYPE declaration, or you might end up with browser display issues.

To learn more about DOCTYPE switching, see http://gutfeldt.ch/matthias/articles/doctypeswitch.html. There's also a great chart there showing numerous declarations and which ones actually flip the switch. XHTML 1.0 Transitional with a proper DOCTYPE as shown in this section was chosen also because it performs this function.

Although DOCTYPE declarations are never displayed, their necessity is inarguable. Using these properly, you can't go wrong: You'll have valid pages that are also interpreted by the browser in as optimal of a situation as possible.

Adding the *html* Element

After the DOCTYPE declaration, you'll want to begin building your document from its root element. I use the term *root* purposely because all documents create a document tree, something that we'll be exploring at length. Understanding the tree created by HTML documents plays an important role in being able to effectively style those documents using CSS.

The `html` element is considered the root element of any HTML document. Remember, the declaration isn't an HTML element—it's SGML. So the first element to appear takes on the important root status.

Example 1-2 shows the `html` element, with its opening tag and closing tag.

EXAMPLE 1-2 The root HTML element

```
<html>

</html>
```

Pretty basic, right? Well, in XHTML, we have to add one other important piece to the opening tag, and that's the XML namespace for XHTML. This is just another way of identifying the language being used within the document. I won't go into the ideological reasons we do this, but suffice it to say that it must be there to validate (see Example 1-3).

EXAMPLE 1-3 The root HTML element with the XML namespace attribute and value

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

</html>
```

You can see the `xmlns` attribute, which stands for "XML namespace" and the value is a URL, which, if you follow it, leads nowhere exciting, I promise! You'll just get a page saying you've reached the XML namespace for XHTML. Again, this is just another identifier.

NOTE
You'll notice a few other bits of syntax, including the `xml:lang` attribute, which defines the language of the document using XML syntax (remember, XHTML is a combo of HTML and XML), in this case `en` for English, and the `lang` attribute from HTML, declaring the same information. These are optional attributes, but we'll use them both, for full compatibility.

The *head* and *title* Elements

Now you've got the very basic beginnings of a document, with the DOCTYPE declaration in place and the root element at the ready. You'll now begin adding other important pieces of the document, beginning with the `head` element. This element is where all things necessary for the document's display and performance are placed—but are not literally seen within the browser window. To create the `head` section, you simply add the `head` tags within the upper portion of your template, right below the opening `<html>` tag (see Example 1-4).

EXAMPLE 1-4 Building the template: Adding a `head` section

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

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

</head>

</html>
```

Notice that the `head` element requires no attributes but simply has the opening and closing tags. This identifies the `head` region. Table 1-1 shows some of the various elements that you can place within the `head` of your document.

TABLE 1-1 Elements in the `Head` Portion of the Document

Element	What It Does
`title`	This element enables you to title your document. This title will then appear in the title bar of your browser. The `title` element is required.
`meta`	The `meta` element is used for numerous concerns, including keywords and descriptions, character encoding, and document authorship. The `meta` element is not required, and your use of it will vary according to your specific needs.
`script`	This element enables you to insert scripts directly into your document or, as is the preference, link from the page to the script you'd like to use. It is used as needed.
`style`	The `style` element enables you to place style information into the individual page. This is known as *embedded* style, which you'll read more about in Chapter 7, "Using CSS." It is used as needed.
`link`	The `link` element is most commonly used to link to an external style sheet, although it can be used for other purposes, such as linking to an alternative page for accessibility, or to link to a favicon, those popular icons you see in the address bar on certain websites.

The *title* Element in Detail

The `title` element is the only *required* element within the `head` element. This element displays any text within it in the browser bar (see Figure 1-2) along with the browser's name at the end of the text.

FIGURE 1-2 The title element text will appear in the browser's title bar.

Aside from the fact that you have to have the `title` element in place, writing good titles is a first-line technique that accomplishes three things:

- Provides a title for the page,
- Offers users *orientation*—that is, it helps them know where they are on the Web and within the site itself
- Provides additional information about the site page

Writing effective titles means addressing these three concerns. A good title example appears in Example 1-5.

EXAMPLE 1-5 Title example with site name and location for user orientation

```
<title>molly.com – books – HTML &amp; CSS</title>
```

Note that the page is titled using the site name, the site section, and the subsection, providing useful information for the visitor.

An ineffective example can be seen in Example 1-6.

EXAMPLE 1-6 Title example with site name and location for user orientation

```
<title>Read my books!</title>
```

Here, there's no information that helps us. So while the technical requirement of having a title is fulfilled, the practical needs are not.

NOTE

Although you cannot use HTML inside a title, you can use character entities, as you can see in Example 1-5, where I used the entity `&` to create the `&` symbol. For more information on available character entities, see Appendix A, "XHTML Reference."

The *meta* Element

Although it is not required in a document, the `meta` element performs so many different functions that it's a good idea to become familiar with it right away.

Document Encoding

Document encoding means setting the character set for your page, which is particularly important when writing documents in other languages. For many years, those of us writing in Latin characters (including English) used the ISO 8859-1 character set. The ISO sets and subsets cover a wide range of languages. But nowadays, we have UTF-8, a more universal format following a different standard than ISO values. UTF-8 can be helpful in a variety of browsers, but there are some limitations. If you are publishing in another language, such as Russian or Japanese, you'll want to have your document encoding set up under ISO rather than Unicode character sets.

NOTE
Ideally, character encoding is set on the server and not in a meta element. However, you can set it using a `meta` element. See http://www.webstandards.org/learn/askw3c/dec2002.html.

Example 1-7 shows a `meta` element that defines the UTF-8 format, suitable for documents in English as well as other languages, depending upon your browser support.

EXAMPLE 1-7 Using `meta` to declare document encoding with Unicode

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

Example 1-8 shows a `meta` element for a document written in Russian, using the ISO method.

EXAMPLE 1-8 Using `meta` to declare document encoding for Cyrillic, using ISO

```
<meta http-equiv="Content-Type" content="text/html; charset= iso-8859-5" />
```

Keywords, Description, and Authorship

The `meta` element can be used to describe keywords, describe the site, and define the author, too. This is extremely helpful for public search engines as well as for any search engine you might be running on your own site.

Keywords are single words and word combinations that would be used during a search. This assists people looking for specific topics to find the information you're providing (see Example 1-9).

EXAMPLE 1-9 Using `meta` for keywords and keyword combinations

```
<meta name="keywords" content="molly, molly.com, html, xhtml, css, design, web
design, development, web development, perl, color, web color, blog, web log,
weblog, books, computer books, articles, tutorials, learn, author, instructor,
instruction, instructing, training, education, consult, consultation, consultant,
famous people page, famous people list, standards, web standards, web standards
project, wsp, wasp, digital web, digital web magazine, web techniques, web
techniques magazine, web review, webreview, webreview.com, wow, world organization
of webmasters, conference, conferences, user interface, usability, accessibility,
internationalization, web culture" />
```

You'll notice that although I use the word *web* a great deal, it's in combination with other keywords. Most search engines will lock you out if you use multiple single keywords. This used to be a way of getting higher ranking, but no longer. Use keywords that make sense, or if you want to have multiple instances of a word, use it in a realistic combination.

Descriptions are typically 25 words or less and describe the purpose of your document (see Example 1-10).

EXAMPLE 1-10 The `meta` element used for site or page description

```
<meta name="description" content="I'm Molly E. Holzschlag, and this Web site
shares my Web development work and personal thoughts." />
```

Short and to the point! Another use is to define the author of the document, as shown in Example 1-11.

EXAMPLE 1-11 Using `meta` to describe page authorship

```
<meta name="Author" content="Molly E. Holzschlag" />
```

Of course, this information is never displayed on your web page itself. Instead, as with all elements and attributes within the `head` portion of a document, this information is used by the browser and other resources such as search engines.

NOTE
Other uses for the `meta` element are to refresh documents automatically and to restrict search engines from logging specific pages. Learn more at http://www.learnthat.com/courses/computer/metatags/meta.html.

The *body* Element

The `body` element is where all the action takes place. It's the element where you'll be placing the content of your page and marking it up using XHTML to structure it accordingly. The element goes within the `html` element, directly below the `head`——makes sense, doesn't it? (See Example 1-12.)

EXAMPLE 1-12 Placing the `body` element

```
<html>
<head>
<title>Appropriate Title Text Here</title>
</head>
<body>
</body>
</html>
```

When viewed in a browser, the information within the `body` element is what is displayed in the browser window, also referred to as the *viewport*. This is the content area only—no browser chrome (which refers to the browser's interface components, such as scrollbars and status bars). Figure 1-3 shows Google in a web browser. Only the displayed content is within the viewport.

FIGURE 1-3 Viewing body text within the browser viewport.

HTML Comments

Another important piece of markup that you'll want to get started using right away is HTML comments. Comments enable you to hide content or markup for temporary purposes or backward compatibility, to identify sections within your document, and to provide directives for other folks who might be working on the page.

The syntax for an HTML comment looks like this:

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

What you are hiding, identifying, or providing in terms of guidance goes between the opening and closing portions of a comment. Example 1-13 hides the text content within the body using comments.

EXAMPLE 1-13 Hiding text content and markup

```
<body>
<!--
<p>The content of this paragraph will not appear within the body so long as it's
within a comment.</p>
-->
<p>The content of this paragraph will be displayed, because it's outside of the
comment field.</p>
</body>
```

You can denote sections within your document, as shown in Example 1-14.

EXAMPLE 1-14 Hiding text content and markup

```
<body>

<!-- begin primary content -->


<!--begin footer information-->

</body>
```

Finally, comments are a useful way to provide directives (see Example 1-15).

EXAMPLE 1-15 Providing guidance inside a comment

```
<body>
<!-- Angie: please be sure to use lists instead of tables in this section  -->
</body>
```

Reviewing the Template

Time to wrap up our exploration of a template by actually putting all the components together (see Example 1-16).

EXAMPLE 1-16 Viewing the structure of an XHTML document

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

<head>

<title>your site : location within site : topic title</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="keywords" content="your, keywords, here" />
<meta name="description" content="your description here" />
<meta name="author" content="your name here" />

</head>

<body>

<!-- main content section  -->

</body>
</html>
```

Copy this markup, save it to a work folder on your computer, and name it `index.html`. This will be the file you'll open and add content and additional markup to as we progress.

Text Is Next!

Of course, if you were to open the document you just created in a web browser, the viewport section—where content normally is displayed—would be completely blank! This is because the only thing in the body is a comment, which hides the text within it. The one thing you will notice is the title of your site, which appears in the title bar at the top of your browser.

Beginning with structure is a great way to learn how to create great pages right away. Although it might seem frustrating to do all this complicated stuff and end up with no visible results, I promise you that the long-term rewards are worth it. You'll end up with far more control and understanding of everything to do with markup and CSS, I assure you.

So after you've saved your document, go ahead and validate it. Browse to http://validate.w3.org/, find the file upload section, and upload your file. Run it through the validator. Find any errors? Fix them and try again. No errors? Great job.

And I promise you that in the next chapter, "Adding Text and Links," you'll end up with something to actually view within your browser!