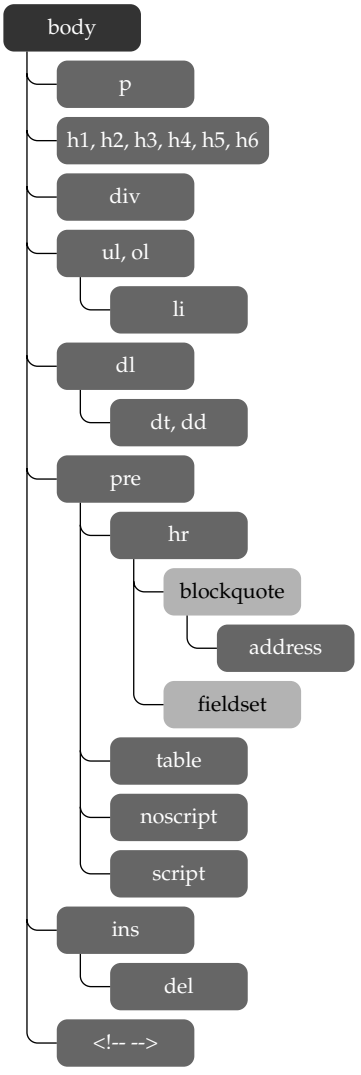
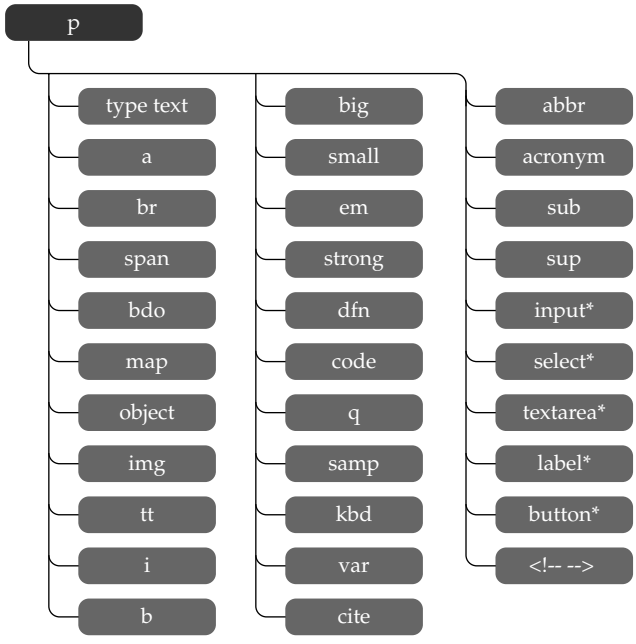
The full syntax of the elements allowed in the **`body`** element is a bit more involved than the full syntax of the **`head`**. This diagram shows what is directly included in the body:



Going deeper into the full syntax in a single diagram is unreasonable to present. Just as an example, take the **p** element and continue to expand, keeping in mind that these elements will also loop back on each other and expand out as well:



(*) when the element is ultimately a descendent of a form element

While it might be difficult to meaningfully present the entire syntax of HTML graphically in a diagram, the diagram presented here should drive home the point that HTML is quite structured and the details of how elements may be used are quite clear. Now that you have some insight into the syntax of markup, the next section discusses how browsers deal with it.

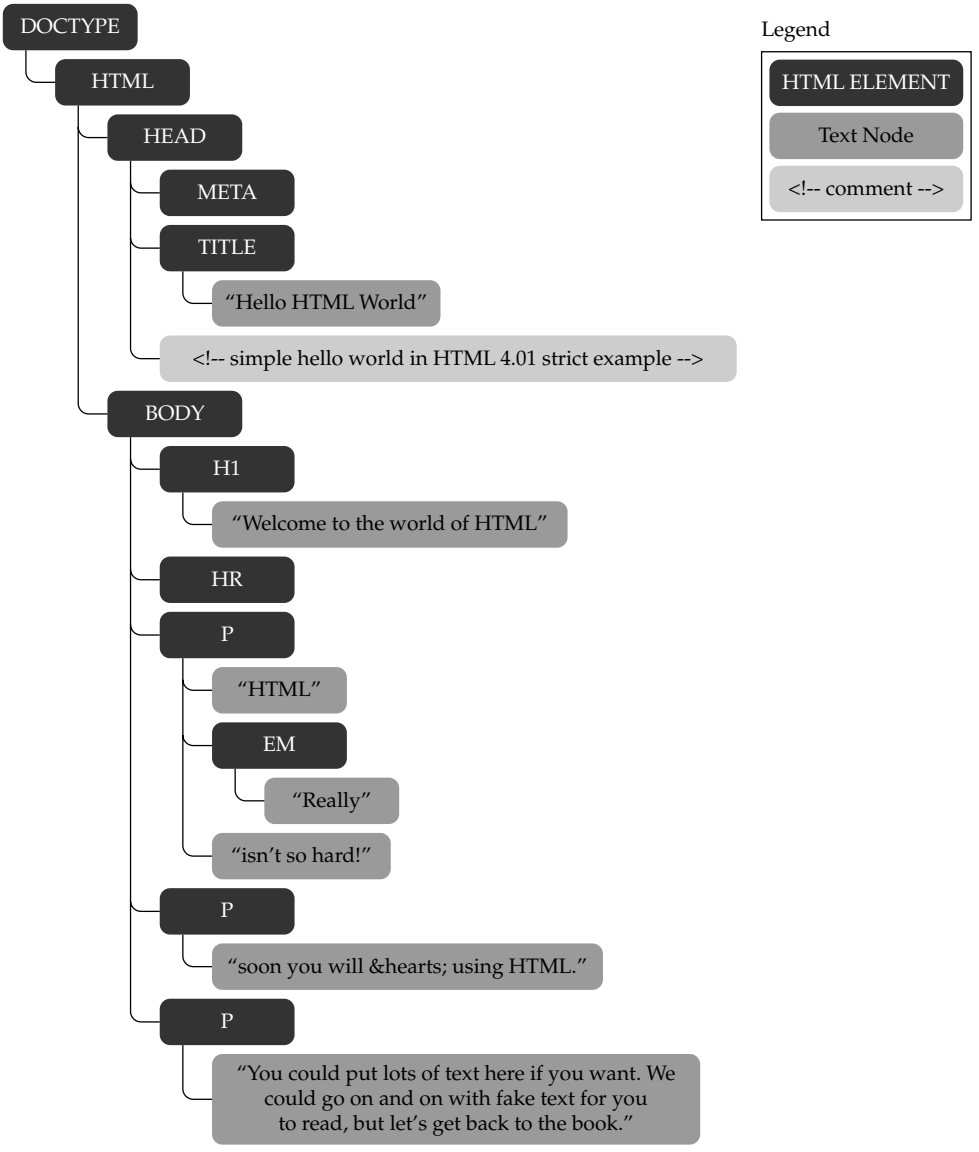
Browsers and (X)HTML

When a browser reads a marked-up document, such as the "hello world" example repeated here,

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Hello HTML World</title>
<!-- Simple hello world in HTML 4.01 strict example -->
</head>
<body>
<h1>Welcome to the World of HTML</h1>
```

```
<hr>
<p>HTML <em>really</em> isn't so hard!</p>
<p>Soon you will &hearts; using HTML.</p>
<p>You can put lots of text here if you want.
We could go on and on with fake text for you
to read, but let's get back to the book.</p>
</body>
</html>
```

it builds a parse tree to interpret the structure of the document, possibly like this:



These parse trees, often called DOM (Document Object Model) trees, are the browsers' interpretation of the markup provided and are integral to determining how to render the page visually using both default (X)HTML style and any CSS attached. JavaScript will also use this parse tree when scripts attempt to manipulate the document. The parse tree serves as the skeleton of the page, so making sure that it is correct is quite important, but sadly we'll see very often it isn't.

NOTE *The syntax trees presented earlier look very similar to the parse trees, and they should, because any particular parse tree should be derivable from the particular markup language's content model.*

Browsers are actually quite permissive in what they will render. For example, consider the following markup:

```
<TITLE>Hello HTML World</title>
<!-- Simple hello malformed world -- example -->
</head>
<body>
<h1>Welcome to the World of HTML</H1>
<hr />
<p>HTML <eM>really</Em> isn't so hard!
<P>Soon you will &hearts; using HTML.
<p>You can put lots of text here if you want.
We could go on and on with fake text for you
to read, <foo>but</foo> let's get back to the book.
</html>
```

This example misses important tags, doesn't specify encoding types, has a malformed comment, uses inconsistent casing, doesn't close tags, and even uses some unknown element **foo**. However, this will render exactly the same visually as the correct markup previously presented, as shown in Figure 1-3.

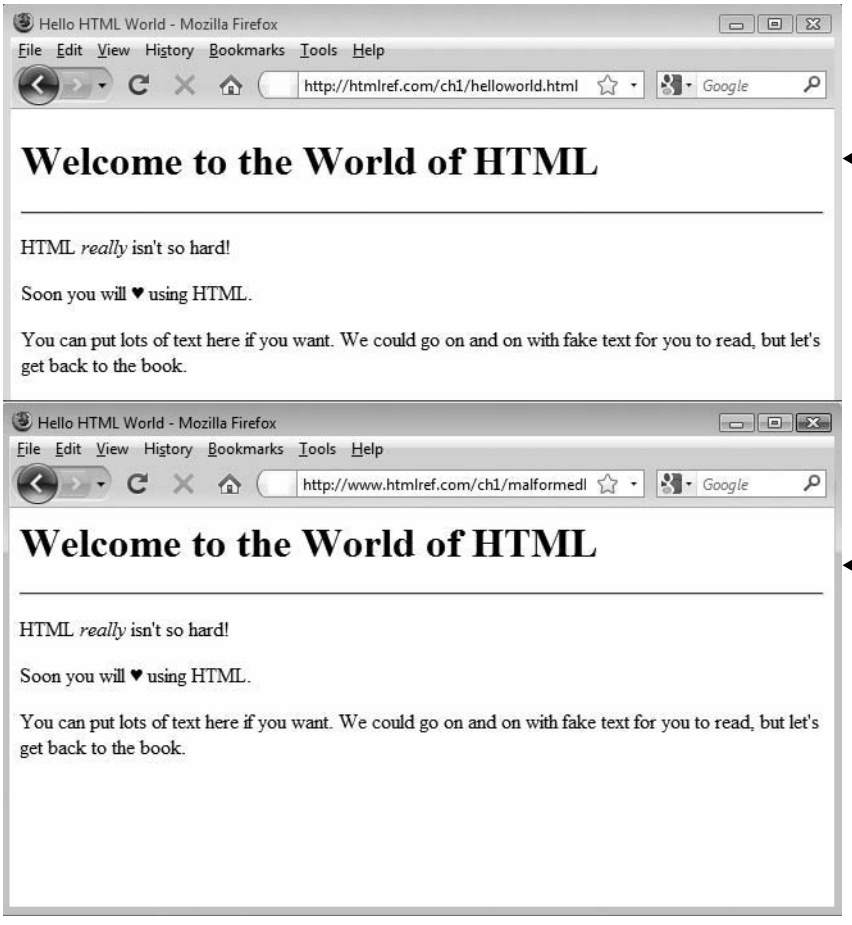


FIGURE 1-3 Malformed markup works!?

Now if you look at the parse tree formed by the browser, you will note that many of the mistakes appear to be magically fixed by the browser:

Of course, the number of assumptions that a browser may make to fix arbitrary syntactical mistakes is likely quite large and different browsers may assume different "fixes." For example, given this small fragment of markup

```
<p>Making malformed HTML <em><strong>really<em><strong> isn't so hard!</p>
```

leading browsers will form their parse trees a bit differently, as shown in Figure 1-4.

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

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<title>Malformed HTML</title>
</head>
<body>
<p>Making malformed HTML <em><strong>really<em><strong>
isn't so hard!</p>
</body>
</html>
```

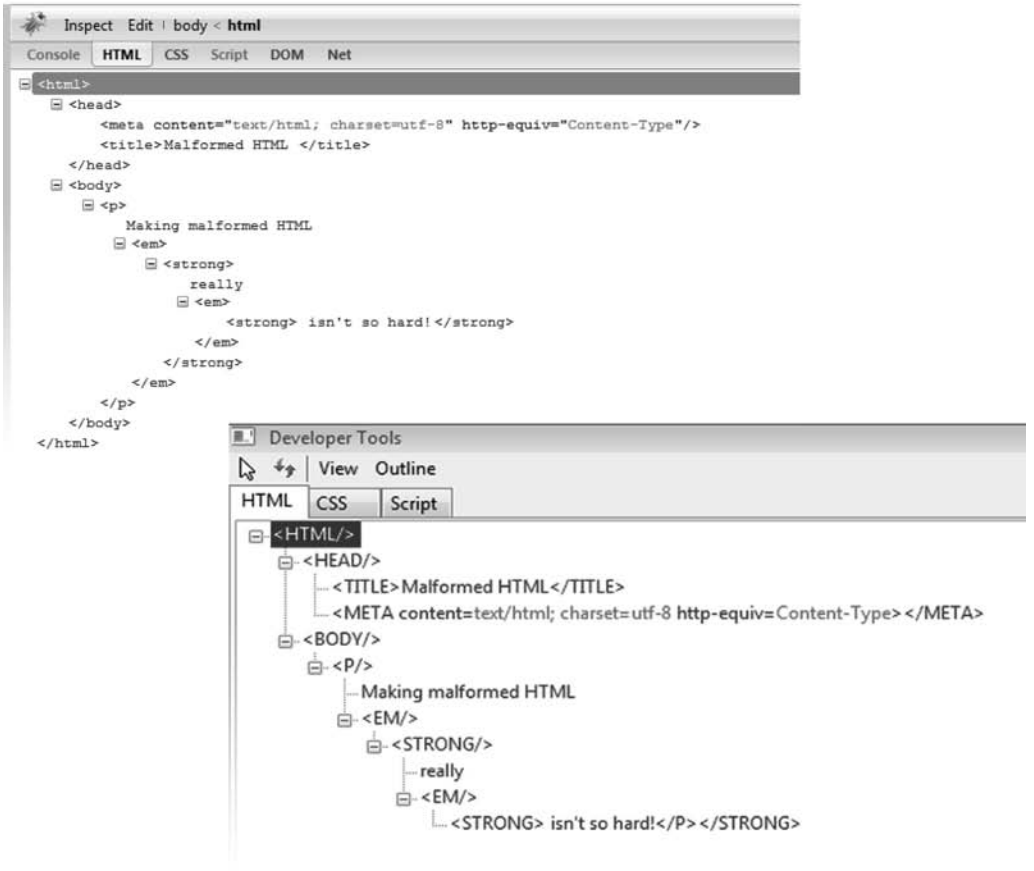


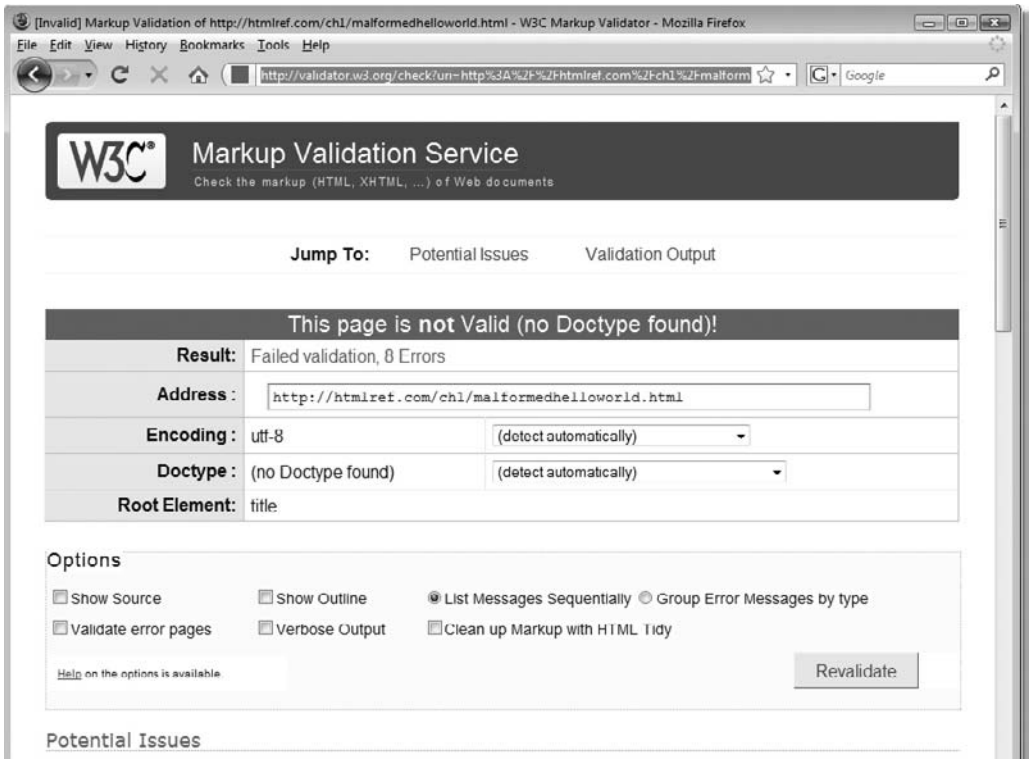
FIGURE 1-4 Same markup, different parse, as shown in Firefox 3 (above) and Internet Explorer 8 (below)

Simply put, it is quite important to aim for correct markup as a solid foundation for a Web page and to not assume the markup is correct just because it appears to render correctly in your favorite browser.

Validation

As shown earlier, a DTD defines the actual elements, attributes, and element relationships that are valid in documents. Now you can take a document written in (X)HTML and then check whether it conforms to the rules specified by the DTD used. This process of checking whether a document conforms to the rules of the DTD is called *validation*.

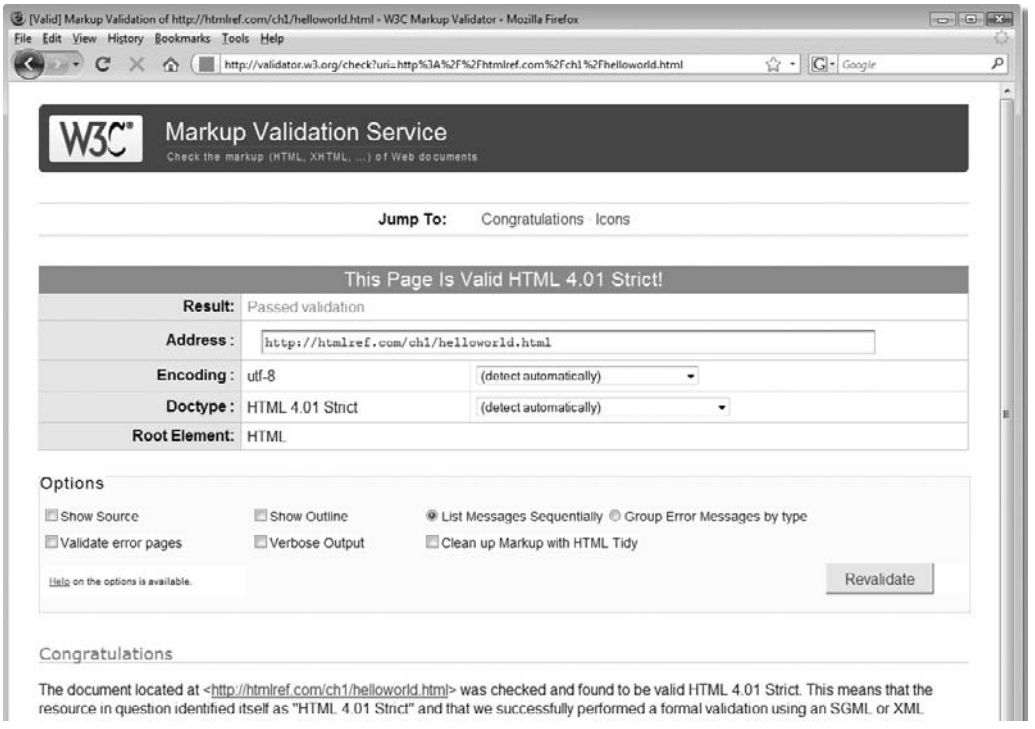
The **`<!DOCTYPE>`** declaration allows validation software to identify the HTML DTD being followed in a document, and verify that the document is syntactically correct—in other words, that all tags used are part of a particular specification and are being used correctly. An easy way to validate a document is simply to use an online service such as the W3C Markup Validation Service, at http://validator.w3.org. If the malformed example from the previous section is passed to this service, it clearly shows that the page has errors:



Pass the URL to the service yourself by using this link in the address bar:

```
http://validator.w3.org/check?uri=http%3A%2F%2Fhtmlref.com%2Fch1%2Fmalformedhelloworld.html
```

By reading the validator's messages about the errors it detected, you can find and correct the various mistakes. After all mistakes are corrected, the document should validate cleanly:



Web developers should aim to start with a baseline of valid markup before trying to address various browser quirks and bugs. Given that so many Web pages on the Web are poorly coded, some developers opt to add a "quality" badge to a page to show or even prove standards conformance: