However, if POET is an attribute rather than a child element, you're stuck with unwieldy constructs like this:

```
<POEM POET="Homer" POET_LANGUAGE="English"
 POEM_LANGUAGE="English">
  Tell me, O Muse, of the cunning man...
</POEM>
```

And it's even more bulky if you want to provide both the poet's English and Greek names.

```
<POEM POET_NAME_1="Homer" POET_LANGUAGE_1="English"
 POET_NAME_2="Ωμηος" POET_LANGUAGE_2="Greek"
 POEM_LANGUAGE="English">
  Tell me, O Muse, of the cunning man...
</POEM>
```

What's Your Meta-data Is Someone Else's Data

"Metaness" is in the mind of the beholder. Who is reading your document and why they are reading it determines what they consider to be data and what they consider to be meta-data. For example, if you're simply reading an article in a scholarly journal, then the author of the article is tangential to the information it contains. However, if you're sitting on a tenure and promotions committee scanning a journal to see who is publishing and who is not, then the names of the authors and the number of articles they've published may be more important to you than what they wrote (sad but true).

In fact, you may change your mind about what's meta and what's data. What's only tangentially relevant today, may become crucial to you next week. You can use style sheets to hide unimportant elements today, and change the style sheets to reveal them later. However, it's more difficult to later reveal information that was first stored in an attribute. Usually, this requires rewriting the document itself rather than simply changing the style sheet.

Elements Are More Extensible

Attributes are certainly convenient when you only need to convey one or two words of unstructured information. In these cases, there may genuinely be no current need for a child element. However, this doesn't preclude such a need in the future.

For instance, you may now only need to store the name of the author of an article, and you may not need to distinguish between the first and last names. However, in the future you may uncover a need to store first and last names, e-mail addresses, institution, snail mail address, URL, and more. If you've stored the author of the article as an element, then it's easy to add child elements to include this additional information.

Although any such change will probably require some revision of your documents, style sheets, and associated programs, it's still much easier to change a simple element to a tree of elements than it is to make an attribute a tree of elements. However, if you used an attribute, then you're stuck. It's quite difficult to extend your attribute syntax beyond the region it was originally designed for.

Good Times to Use Attributes

Having exhausted all the reasons why you should use elements instead of attributes, I feel compelled to point out that there are nonetheless some times when attributes make sense. First of all, as previously mentioned, attributes are fully appropriate for very simple data without substructure that the reader is unlikely to want to see. One example is the `HEIGHT` and `WIDTH` attributes of an `IMG`. Although the values of these attributes may change if the image changes, it's hard to imagine how the data in the attribute could be anything more than a very short string of text. `HEIGHT` and `WIDTH` are one-dimensional quantities (in more ways than one) so they work well as attributes.

Furthermore, attributes are appropriate for simple information about the document that has nothing to do with the content of the document. For example, it is often useful to assign an `ID` attribute to each element. This is a unique string possessed only by one element in the document. You can then use this string for a variety of tasks including linking to particular elements of the document, even if the elements move around as the document changes over time. For example:

```
<SOURCE ID="S1">
  <AUTHOR ID="A1">Donald Dewey</AUTHOR>
  <AUTHOR ID="A2">Nicholas Acocella</AUTHOR>
  <BOOK ID="B1">
    <TITLE ID="B2">
      The Biographical History of Baseball
    </TITLE>
    <PAGES ID="B3">169</PAGES>
    <YEAR ID="B4">1995</YEAR>
  </BOOK>
</SOURCE>
```

`ID` attributes make links to particular elements in the document possible. In this way, they can serve the same purpose as the `NAME` attribute of HTML's `A` elements. Other data associated with linking — `HREF`s to link to, `SRC`s to pull images and binary data from, and so forth — also work well as attributes.

Cross-Reference

You'll see more examples of this when XLL, the Extensible Linking Language, is discussed in Chapter 16, *XLinks*, and Chapter 17, *XPointers*.

Attributes are also often used to store document-specific style information. For example, if TITLE elements are generally rendered as bold text but if you want to make just one TITLE element both bold and italic, you might write something like this:

```
<TITLE style="font-style: italic">Significant Others</TITLE>
```

This enables the style information to be embedded without changing the tree structure of the document. While ideally you'd like to use a separate element, this scheme gives document authors somewhat more control when they cannot add elements to the tag set they're working with. For example, the Webmaster of a site might require the use of a particular DTD and not want to allow everyone to modify the DTD. Nonetheless, they want to allow them to make minor adjustments to individual pages. Use this scheme with restraint, however, or you'll soon find yourself back in the HTML hell XML was supposed to save us from, where formatting is freely intermixed with meaning and documents are no longer maintainable.

The final reason to use attributes is to maintain compatibility with HTML. To the extent that you're using tags that at least look similar to HTML such as <IMG>, <P>, and <TD>, you might as well employ the standard HTML attributes for these tags. This has the double advantage of enabling legacy browsers to at least partially parse and display your document, and of being more familiar to the people writing the documents.

Empty Tags

Last chapter's no-attribute approach was an extreme position. It's also possible to swing to the other extreme— storing all the information in the attributes and none in the content. In general, I don't recommend this approach. Storing all the information in element content—while equally extreme—is much easier to work with in practice. However, this section entertains the possibility of using only attributes for the sake of elucidation.

As long as you know the element will have no content, you can use empty tags as a short cut. Rather than including both a start and an end tag you can include one empty tag. Empty tags are distinguished from start tags by a closing /> instead of simply a closing >. For instance, instead of <PLAYER></PLAYER> you would write <PLAYER/>.

Empty tags may contain attributes. For example, here's an empty tag for Joe Girardi with several attributes:

```
<PLAYER GIVEN_NAME="Joe" SURNAME="Girardi"
  GAMES="78" AT_BATS="254" RUNS="31" HITS="70"
  DOUBLES="11" TRIPLES="4" HOME_RUNS="3"
  RUNS_BATTED_IN="31" WALKS="14" STRUCK_OUT="38"
  STOLEN_BASES="2" CAUGHT_STEALING="4"
```

```
SACRIFICE_FLY="1" SACRIFICE_HIT="8"
HIT_BY_PITCH="2"/>
```

XML parsers treat this identically to the non-empty equivalent. This `PLAYER` element is precisely equal (though not identical) to the previous `PLAYER` element formed with an empty tag.

```
<PLAYER GIVEN_NAME="Joe" SURNAME="Girardi"
  GAMES="78" AT_BATS="254" RUNS="31" HITS="70"
  DOUBLES="11" TRIPLES="4" HOME_RUNS="3"
  RUNS_BATTED_IN="31" WALKS="14" STRUCK_OUT="38"
  STOLEN_BASES="2" CAUGHT_STEALING="4"
  SACRIFICE_FLY="1" SACRIFICE_HIT="8"
  HIT_BY_PITCH="2"></PLAYER>
```

The difference between `<PLAYER/>` and `<PLAYER></PLAYER>` is syntactic sugar, and nothing more. If you don't like the empty tag syntax, or find it hard to read, you don't have to use it.

XSL

Attributes are visible in an XML source view of the document as shown in Figure 5-1. However, once a CSS style sheet is applied the attributes disappear. Figure 5-3 shows Listing 5-1 once the baseball stats style sheet from the previous chapter is applied. It looks like a blank document because CSS styles only apply to element content, not to attributes. If you use CSS, any data you want to display to the reader should be part of an element's content rather than one of its attributes.

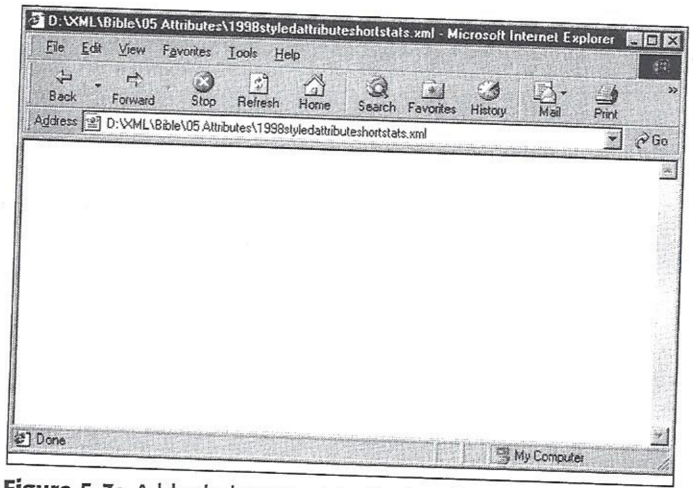

D:\XML\Bible\05 Attributes\1998styledattributeshortstats.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address D:\XML\Bible\05 Attributes\1998styledattributeshortstats.xml
Go
Done
My Computer

Figure 5-3: A blank document is displayed when CSS is applied to an XML document whose elements do not contain any character data.

However, there is an alternative style sheet language that does allow you to access and display attribute data. This language is the Extensible Style Language (XSL); and it is also supported by Internet Explorer 5.0, at least in part. XSL is divided into two sections, transformations and formatting.

The transformation part of XSL enables you to replace one tag with another. You can define rules that replace your XML tags with standard HTML tags, or with HTML tags plus CSS attributes. You can also do a lot more including reordering the elements in the document and adding additional content that was never present in the XML document.

The formatting part of XSL defines an extremely powerful view of documents as pages. XSL formatting enables you to specify the appearance and layout of a page including multiple columns, text flow around objects, line spacing, assorted font properties, and more. It's designed to be powerful enough to handle automated layout tasks for both the Web and print from the same source document. For instance, XSL formatting would allow one XML document containing show times and advertisements to generate both the print and online editions of a local newspaper's television listings. However, IE 5.0 and most other tools do not yet support XSL formatting. Therefore, in this section I'll focus on XSL transformations.

Cross-Reference

XSL formatting is discussed in Chapter 15, *XSL Formatting Objects*.

XSL Style Sheet Templates

An XSL style sheet contains templates into which data from the XML document is poured. For example, one template might look something like this:

```
<HTML>
  <HEAD>
    <TITLE>
      XSL Instructions to get the title
    </TITLE>
  </HEAD>
  <H1>XSL Instructions to get the title</H1>
  <BODY>
    XSL Instructions to get the statistics
  </BODY>
</HTML>
```

The italicized sections will be replaced by particular XSL elements that copy data from the underlying XML document into this template. You can apply this template to many different data sets. For instance, if the template is designed to work with the baseball example, then the same style sheet can display statistics from different seasons.

This may remind you of some server-side include schemes for HTML. In fact, this is very much like server-side includes. However, the actual transformation of the source XML document and XSL style sheet takes place on the client rather than on the server. Furthermore, the output document does not have to be HTML. It can be any well-formed XML.

XSL instructions can retrieve any data stored in the elements of the XML document. This includes element content, element names, and, most importantly for our example, element attributes. Particular elements are chosen by a pattern that considers the element's name, its value, its attributes' names and values, its absolute and relative position in the tree structure of the XML document, and more. Once the data is extracted from an element, it can be moved, copied, and manipulated in a variety of ways. We won't cover everything you can do with XML transformations in this brief introduction. However, you will learn to use XSL to write some pretty amazing documents that can be viewed on the Web right away.

Cross-Reference

Chapter 14, *XSL Transformations*, covers XSL transformations in depth.

The Body of the Document

Let's begin by looking at a simple example and applying it to the XML document with baseball statistics shown in Listing 5-1. Listing 5-2 is an XSL style sheet. This style sheet provides the HTML mold into which XML data will be poured.

Listing 5-2: An XSL style sheet

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          Major League Baseball Statistics
        </TITLE>
      </HEAD>
      <BODY>
        <H1>Major League Baseball Statistics</H1>

        <HR></HR>
        Copyright 1999
        <A HREF="http://www.macfaq.com/personal.html">
         Elliotte Rusty Harold
        </A>
```

Continued

Listing 5-2 *(continued)*

```
          <BR />
          <A HREF="mailto:elharo@metalab.unc.edu">
           elharo@metalab.unc.edu
          </A>

        </BODY>
      </HTML>
    </xsl:template>

  </xsl:stylesheet>
```

It resembles an HTML file included inside an `xsl:template` element. In other words its structure looks like this:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    HTML file goes here
  </xsl:template>

</xsl:stylesheet>
```

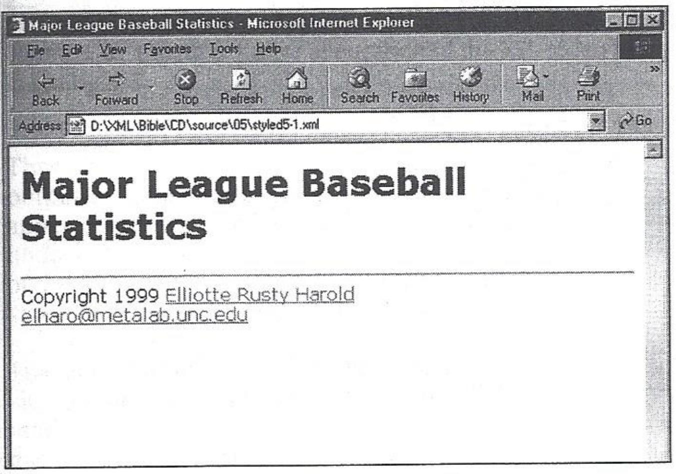
Listing 5-2 is not only an XSL style sheet; it's also a well-formed XML document. It begins with an XML declaration. The root element of this document is `xsl:stylesheet`. This style sheet contains a single template for the XML data encoded as an `xsl:template` element. The `xsl:template` element has a `match` attribute with the value / and its content is a well-formed HTML document. It's not a coincidence that the output HTML is well-formed. Because the HTML must first be part of an XSL style sheet, and because XSL style sheets are well-formed XML documents, all the HTML in a XSL style sheet must be well-formed.

The Web browser tries to match parts of the XML document against each `xsl:template` element. The / template matches the root of the document; that is the entire document itself. The browser reads the template and inserts data from the XML document where indicated by XSL instructions. However, this particular template contains no XSL instructions, so its contents are merely copied verbatim into the Web browser, producing the output you see in Figure 5-4. Notice that Figure 5-4 does not display any data from the XML document, only from the XSL template.

Attaching the XSL style sheet of Listing 5-2 to the XML document in Listing 5-1 is straightforward. Simply add a `<?xml-stylesheet?>` processing instruction with a `type` attribute with value `text/xsl` and an `href` attribute that points to the style sheet between the XML declaration and the root element. For example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="5-2.xsl"?>
<SEASON YEAR="1998">
...
```

This is the same way a CSS style sheet is attached to a document. The only difference is that the `type` attribute is `text/xsl` instead of `text/css`.


Major League Baseball Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address D:\XML\Bible\CD\source\05\styled5-1.xml
Go

Major League Baseball Statistics

Copyright 1999 Elliotte Rusty Harold
elharo@metalab.unc.edu

Figure 5-4: The data from the XML document, not the XSL template, is missing after application of the XSL style sheet in Listing 5-2.

The Title

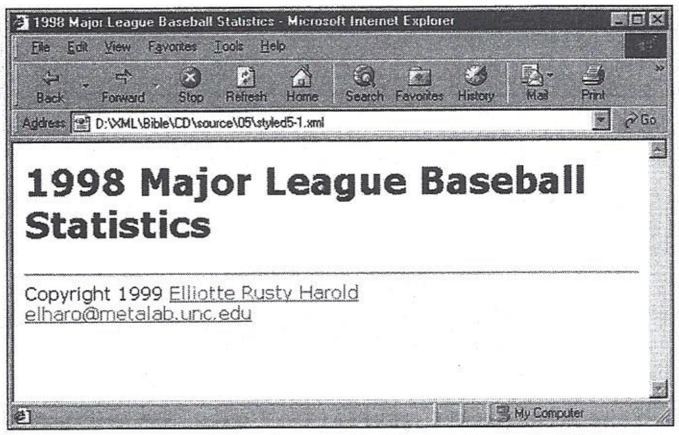
Of course there was something rather obvious missing from Figure 5-4 — the data! Although the style sheet in Listing 5-2 displays something (unlike the CSS style sheet of Figure 5-3) it doesn't show any data from the XML document. To add this, you need to use XSL instruction elements to copy data from the source XML document into the XSL template. Listing 5-3 adds the necessary XSL instructions to extract the `YEAR` attribute from the `SEASON` element and insert it in the `TITLE` and `H1` header of the resulting document. Figure 5-5 shows the rendered document.

Listing 5-3: An XSL style sheet with instructions to extract the SEASON element and YEAR attribute

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:for-each select="SEASON">
            <xsl:value-of select="@YEAR"/>
          </xsl:for-each>
          Major League Baseball Statistics
        </TITLE>
      </HEAD>
      <BODY>

      <xsl:for-each select="SEASON">
        <H1>
          <xsl:value-of select="@YEAR"/>
          Major League Baseball Statistics
        </H1>
      </xsl:for-each>

      <HR></HR>
      Copyright 1999
      <A HREF="http://www.macfaq.com/personal.html">
       Elliotte Rusty Harold
      </A>
      <BR />
      <A HREF="mailto:elharo@metalab.unc.edu">
       elharo@metalab.unc.edu
      </A>

      </BODY>
    </HTML>
  </xsl:template>

</xsl:stylesheet>
```

The new XSL instructions that extract the YEAR attribute from the SEASON element are:

```
<xsl:for-each select="SEASON">
  <xsl:value-of select="@YEAR"/>
</xsl:for-each>
```


1998 Major League Baseball Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address D:\XML\Bible\CD\source\05\styled5-1.xml
Go
My Computer

1998 Major League Baseball Statistics

Copyright 1999 Elliotte Rusty Harold
elharo@metalab.unc.edu

Figure 5-5: Listing 5-1 after application of the XSL style sheet in Listing 5-3

These instructions appear twice because we want the year to appear twice in the output document-once in the `H1` header and once in the `TITLE`. Each time they appear, these instructions do the same thing. `<xsl:for-each select="SEASON">` finds all `SEASON` elements. `<xsl:value-of select="@YEAR"/>` inserts the value of the `YEAR` attribute of the `SEASON` element — that is, the string "1998" — found by `<xsl:for-each select="SEASON">`.

This is important, so let me say it again: `xsl:for-each` selects a particular XML element in the source document (Listing 5-1 in this case) from which data will be read. `xsl:value-of` copies a particular part of the element into the output document. You need both XSL instructions. Neither alone is sufficient.

XSL instructions are distinguished from output elements like `HTML` and `H1` because the instructions are in the `xsl` namespace. That is, the names of all XSL elements begin with `xsl:`. The namespace is identified by the `xmlns:xsl` attribute of the root element of the style sheet. In Listings 5-2, 5-3, and all other examples in this book, the value of that attribute is `http://www.w3.org/TR/WD-xsl`.

Cross-Reference

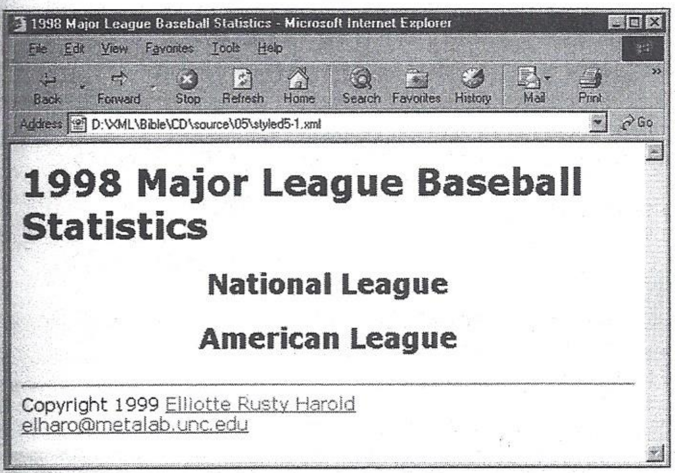
Namespaces are covered in depth in Chapter 18, *Namespaces*.

Leagues, Divisions, and Teams

Next, let's add some XSL instructions to pull out the two `LEAGUE` elements. We'll map these to `H2` headers. Listing 5-4 demonstrates. Figure 5-6 shows the document rendered with this style sheet.

Listing 5-4: An XSL style sheet with instructions to extract LEAGUE elements

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:for-each select="SEASON">
            <xsl:value-of select="@YEAR"/>
          </xsl:for-each>

          Major League Baseball Statistics
        </TITLE>
      </HEAD>
      <BODY>

      <xsl:for-each select="SEASON">
        <H1>
          <xsl:value-of select="@YEAR"/>
          Major League Baseball Statistics
        </H1>

        <xsl:for-each select="LEAGUE">
          <H2 ALIGN="CENTER">
            <xsl:value-of select="@NAME"/>
          </H2>
        </xsl:for-each>

      </xsl:for-each>

      <HR></HR>
      Copyright 1999
      <A HREF="http://www.macfaq.com/personal.html">
       Elliotte Rusty Harold
      </A>
      <BR />
      <A HREF="mailto:elharo@metalab.unc.edu">
       elharo@metalab.unc.edu
      </A>

      </BODY>
    </HTML>
  </xsl:template>

</xsl:stylesheet>
```


1998 Major League Baseball Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address D:\XML\Bible\CD\source\05\styled5-1.xml
Go

1998 Major League Baseball Statistics

National League

American League

Copyright 1999 Elliotte Rusty Harold
elharo@metalab.unc.edu

Figure 5-6: The league names are displayed as H2 headers when the XSL style sheet in Listing 5-4 is applied.

The key new materials are the nested `xsl:for-each` instructions

```
<xsl:for-each select="SEASON">
  <H1>
    <xsl:value-of select="@YEAR"/>
    Major League Baseball Statistics
  </H1>

  <xsl:for-each select="LEAGUE">
    <H2 ALIGN="CENTER">
      <xsl:value-of select="@NAME"/>
    </H2>
  </xsl:for-each>

</xsl:for-each>
```

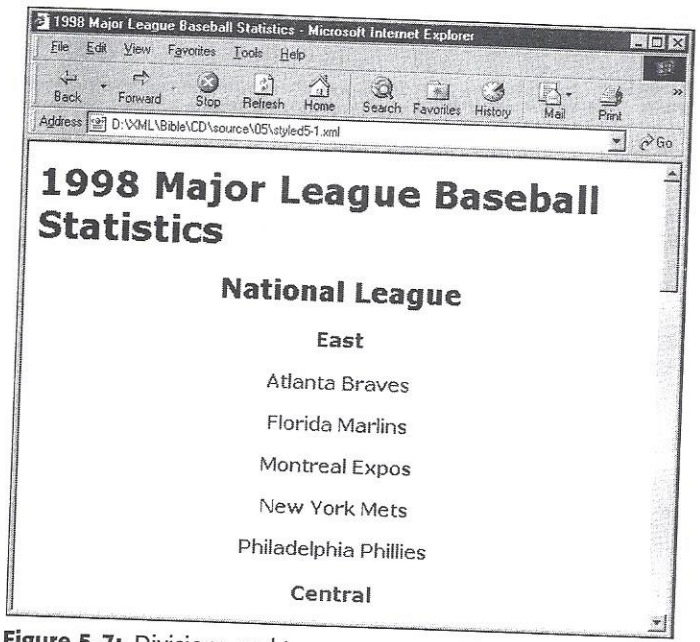
The outermost instruction says to select the `SEASON` element. With that element selected, we then find the `YEAR` attribute of that element and place it between `<H1>` and `</H1>` along with the extra text `Major League Baseball Statistics`. Next, the browser loops through each `LEAGUE` child of the selected `SEASON` and places the value of its `NAME` attribute between `<H2 ALIGN="CENTER">` and `</H2>`. Although there's only one `xsl:for-each` matching a `LEAGUE` element, it loops over all the `LEAGUE` elements that are immediate children of the `SEASON` element. Thus, this template works for anywhere from zero to an indefinite number of leagues.

The same technique can be used to assign `H3` headers to divisions and `H4` headers to teams. Listing 5-5 demonstrates the procedure and Figure 5-7 shows the document rendered with this style sheet. The names of the divisions and teams are read from the XML data.

Listing 5-5: An XSL style sheet with instructions to extract DIVISION and TEAM elements

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:for-each select="SEASON">
            <xsl:value-of select="@YEAR"/>
          </xsl:for-each>

          Major League Baseball Statistics
        </TITLE>
      </HEAD>
      <BODY>

      <xsl:for-each select="SEASON">
        <H1>
          <xsl:value-of select="@YEAR"/>
          Major League Baseball Statistics
        </H1>

        <xsl:for-each select="LEAGUE">
          <H2 ALIGN="CENTER">
            <xsl:value-of select="@NAME"/>
          </H2>

          <xsl:for-each select="DIVISION">
            <H3 ALIGN="CENTER">
            <xsl:value-of select="@NAME"/>
            </H3>

            <xsl:for-each select="TEAM">
              <H4 ALIGN="CENTER">
              <xsl:value-of select="@CITY"/>
              <xsl:value-of select="@NAME"/>
              </H4>
            </xsl:for-each>
          </xsl:for-each>

        </xsl:for-each>
      </xsl:for-each>

      <HR></HR>
      Copyright 1999
      <A HREF="http://www.macfaq.com/personal.html">
```

```
         Elliotte Rusty Harold
        </A>
        <BR />
        <A HREF="mailto:elharo@metalab.unc.edu">
         elharo@metalab.unc.edu
        </A>

        </BODY>
      </HTML>
    </xsl:template>

  </xsl:stylesheet>
```


1998 Major League Baseball Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address D:\XML\Bible\CD\source\05\styled5-1.xml
Go

1998 Major League Baseball Statistics

National League

East

Atlanta Braves

Florida Marlins

Montreal Expos

New York Mets

Philadelphia Phillies

Central

Figure 5-7: Divisions and team names are displayed after application of the XSL style sheet in Listing 5-5.

In the case of the `TEAM` elements, the values of both its `CITY` and `NAME` attributes are used as contents for the `H4` header. Also notice that the nesting of the `xsl:for-each` elements that selects seasons, leagues, divisions, and teams mirrors the hierarchy of the document itself. That's not a coincidence. While other schemes are possible that don't require matching hierarchies, this is the simplest, especially for highly structured data like the baseball statistics of Listing 5-1.

Players

The next step is to add statistics for individual players on a team. The most natural way to do this is in a table. Listing 5-6 shows an XSL style sheet that arranges the players and their stats in a table. No new XSL elements are introduced. The same `xsl:for-each` and `xsl:value-of` elements are used on the `PLAYER` element and its attributes. The output is standard HTML table tags. Figure 5-8 displays the results.

Listing 5-6: An XSL style sheet that places players and their statistics in a table

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:for-each select="SEASON">
            <xsl:value-of select="@YEAR"/>
          </xsl:for-each>

          Major League Baseball Statistics
        </TITLE>
      </HEAD>
      <BODY>

      <xsl:for-each select="SEASON">
        <H1>
          <xsl:value-of select="@YEAR"/>
          Major League Baseball Statistics
        </H1>

        <xsl:for-each select="LEAGUE">
          <H2 ALIGN="CENTER">
            <xsl:value-of select="@NAME"/>
          </H2>

          <xsl:for-each select="DIVISION">
            <H3 ALIGN="CENTER">
            <xsl:value-of select="@NAME"/>
            </H3>

            <xsl:for-each select="TEAM">
              <H4 ALIGN="CENTER">
              <xsl:value-of select="@CITY"/>
              <xsl:value-of select="@NAME"/>
              </H4>

              <TABLE>
    <THEAD>
     <TR>
      <TH>Player</TH><TH>P</TH><TH>G</TH>
      <TH>GS</TH><TH>AB</TH><TH>R</TH><TH>H</TH>
      <TH>D</TH><TH>T</TH><TH>HR</TH><TH>RBI</TH>
      <TH>S</TH><TH>CS</TH><TH>SH</TH><TH>SF</TH>
      <TH>E</TH><TH>BB</TH><TH>SO</TH><TH>HBP</TH>
     </TR>
    </THEAD>
   <TBODY>
    <xsl:for-each select="PLAYER">
     <TR>
      <TD>
       <xsl:value-of select="@GIVEN_NAME"/>
       <xsl:value-of select="@SURNAME"/>
      </TD>
      <TD><xsl:value-of select="@POSITION"/></TD>
      <TD><xsl:value-of select="@GAMES"/></TD>
      <TD>
        <xsl:value-of select="@GAMES_STARTED"/>
      </TD>
      <TD><xsl:value-of select="@AT_BATS"/></TD>
      <TD><xsl:value-of select="@RUNS"/></TD>
      <TD><xsl:value-of select="@HITS"/></TD>
      <TD><xsl:value-of select="@DOUBLES"/></TD>
      <TD><xsl:value-of select="@TRIPLES"/></TD>
      <TD><xsl:value-of select="@HOME_RUNS"/></TD>
      <TD><xsl:value-of select="@RBI"/></TD>
      <TD><xsl:value-of select="@STEALS"/></TD>
      <TD>
       <xsl:value-of select="@CAUGHT_STEALING"/>
      </TD>
      <TD>
       <xsl:value-of select="@SACRIFICE_HITS"/>
      </TD>
      <TD>
       <xsl:value-of select="@SACRIFICE_FLIES"/>
      </TD>
      <TD><xsl:value-of select="@ERRORS"/></TD>
      <TD><xsl:value-of select="@WALKS"/></TD>
      <TD>
       <xsl:value-of select="@STRUCK_OUT"/>
      </TD>
      <TD>
       <xsl:value-of select="@HIT_BY_PITCH"/>
      </TD>
     </TR>
    </xsl:for-each>
   </TBODY>
  </TABLE>

</xsl:for-each>
```

Continued

Listing 5-6 *(continued)*

```
          </xsl:for-each>

        </xsl:for-each>
      </xsl:for-each>

      <HR></HR>
      Copyright 1999
      <A HREF="http://www.macfaq.com/personal.html">
       Elliotte Rusty Harold
      </A>
      <BR />
      <A HREF="mailto:elharo@metalab.unc.edu">
       elharo@metalab.unc.edu
      </A>

      </BODY>
    </HTML>
  </xsl:template>

</xsl:stylesheet>
```

Separation of Pitchers and Batters

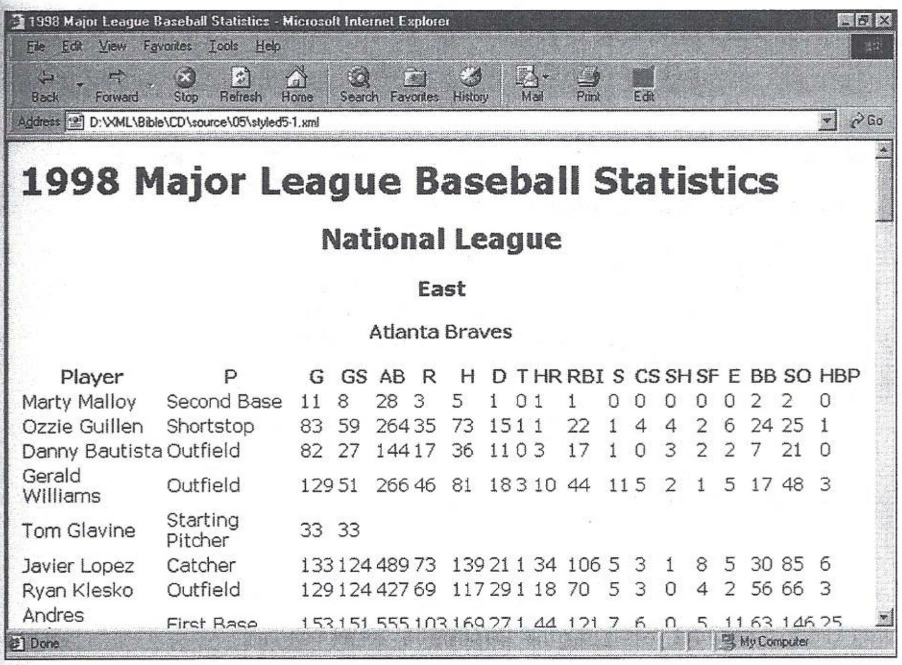
One discrepancy you might notice in Figure 5-8 is that the pitchers aren't handled properly. Throughout this chapter and Chapter 4, we've always given the pitchers a completely different set of statistics, whether those stats were stored in element content or attributes. Therefore, the pitchers really need a table that is separate from the other players. Before putting a player into the table, you must test whether he is or is not a pitcher. If his `POSITION` attribute contains the string "pitcher" then omit him. Then reverse the procedure in a second table that only includes pitchers-`PLAYER` elements whose `POSITION` attribute contains the string "pitcher".

To do this, you have to add additional code to the `xsl:for-each` element that selects the players. You don't select all players. Instead, you select those players whose `POSITION` attribute is not pitcher. The syntax looks like this:

```
<xsl:for-each select="PLAYER[(@POSITION != 'Pitcher')">
```

But because the XML document distinguishes between starting and relief pitchers, the true answer must test both cases:

```
<xsl:for-each select="PLAYER[(@POSITION != 'Starting Pitcher')
  $and$ (@POSITION != 'Relief Pitcher')]">
```


1998 Major League Baseball Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address D:\XML\Bible\CD\source\05\styled5-1.xml
Go
Done
My Computer

1998 Major League Baseball Statistics

National League

East

Atlanta Braves

Player	P	G	GS	AB	R	H	D	T	HR	RBI	S	CS	SH	SF	E	BB	SO	HBP
Marty Malloy	Second Base	11	8	28	3	5	1	0	1	1	0	0	0	0	0	2	2	0
Ozzie Guillen	Shortstop	83	59	264	35	73	15	1	1	22	1	4	4	2	6	24	25	1
Danny Bautista	Outfield	82	27	144	17	36	11	0	3	17	1	0	3	2	2	7	21	0
Gerald Williams	Outfield	129	51	266	46	81	18	3	10	44	11	5	2	1	5	17	48	3
Tom Glavine	Starting Pitcher	33	33															
Javier Lopez	Catcher	133	124	489	73	139	21	1	34	106	5	3	1	8	5	30	85	6
Ryan Klesko	Outfield	129	124	427	69	117	29	1	18	70	5	3	0	4	2	56	66	3
Andres	First Base	153	151	555	103	169	27	1	44	121	7	6	0	5	11	63	146	25

Figure 5-8: Player statistics are displayed after applying the XSL style sheet in Listing 5-6.

For the table of pitchers, you logically reverse this to the position being equal to either "Starting Pitcher" or "Relief Pitcher". (It is not sufficient to just change *not equal* to *equal*. You also have to change *and* to *or*.) The syntax looks like this:

```
<xsl:for-each select="PLAYER[(@POSITION = 'Starting Pitcher')
 $or$ (@POSITION = 'Relief Pitcher')]">
```

Note

Only a single equals sign is used to test for equality rather than the double equals sign used in C and Java. That's because there's no equivalent of an assignment operator in XSL.

Listing 5-7 shows an XSL style sheet separating the batters and pitchers into two different tables. The pitchers' table adds columns for all the usual pitcher statistics. Listing 5-1 encodes in attributes: wins, losses, saves, shutouts, etc. Abbreviations are used for the column labels to keep the table to a manageable width. Figure 5-9 shows the results.

Listing 5-7: An XSL style sheet that separates batters and pitchers

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:for-each select="SEASON">
            <xsl:value-of select="@YEAR"/>
          </xsl:for-each>

          Major League Baseball Statistics
        </TITLE>
      </HEAD>
      <BODY>

        <xsl:for-each select="SEASON">
          <H1>
            <xsl:value-of select="@YEAR"/>
            Major League Baseball Statistics
          </H1>

          <xsl:for-each select="LEAGUE">
            <H2 ALIGN="CENTER">
              <xsl:value-of select="@NAME"/>
            </H2>

            <xsl:for-each select="DIVISION">
              <H3 ALIGN="CENTER">
              <xsl:value-of select="@NAME"/>
              </H3>

              <xsl:for-each select="TEAM">
                <H4 ALIGN="CENTER">
                <xsl:value-of select="@CITY"/>
                <xsl:value-of select="@NAME"/>
                </H4>

                <TABLE>
                 <CAPTION><B>Batters</B></CAPTION>
                 <THEAD>
                  <TR>
                   <TH>Player</TH><TH>P</TH><TH>G</TH>
                   <TH>GS</TH><TH>AB</TH><TH>R</TH><TH>H</TH>
                   <TH>D</TH><TH>T</TH><TH>HR</TH><TH>RBI</TH>
                   <TH>S</TH><TH>CS</TH><TH>SH</TH><TH>SF</TH>
                   <TH>E</TH><TH>BB</TH><TH>SO</TH>
                   <TH>HBP</TH>
                  </TR>
                 </THEAD>
```

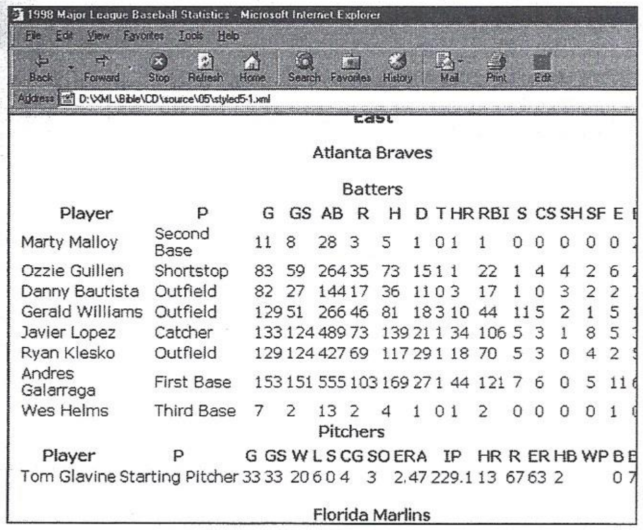
```
 <TBODY>
  <xsl:for-each select="PLAYER[(@POSITION
    != 'Starting Pitcher')
   $and$ (@POSITION != 'Relief Pitcher')]">
   <TR>
    <TD>
     <xsl:value-of select="@GIVEN_NAME"/>
     <xsl:value-of select="@SURNAME"/>
    </TD>
    <TD><xsl:value-of select="@POSITION"/></TD>
    <TD><xsl:value-of select="@GAMES"/></TD>
    <TD>
      <xsl:value-of select="@GAMES_STARTED"/>
    </TD>
    <TD><xsl:value-of select="@AT_BATS"/></TD>
    <TD><xsl:value-of select="@RUNS"/></TD>
    <TD><xsl:value-of select="@HITS"/></TD>
    <TD><xsl:value-of select="@DOUBLES"/></TD>
    <TD><xsl:value-of select="@TRIPLES"/></TD>
    <TD>
      <xsl:value-of select="@HOME_RUNS"/>
    </TD>
    <TD><xsl:value-of select="@RBI"/></TD>
    <TD><xsl:value-of select="@STEALS"/></TD>
    <TD>
     <xsl:value-of select="@CAUGHT_STEALING"/>
    </TD>
    <TD>
     <xsl:value-of select="@SACRIFICE_HITS"/>
    </TD>
    <TD>
     <xsl:value-of select="@SACRIFICE_FLIES"/>
    </TD>
    <TD><xsl:value-of select="@ERRORS"/></TD>
    <TD><xsl:value-of select="@WALKS"/></TD>
    <TD>
     <xsl:value-of select="@STRUCK_OUT"/>
    </TD>
    <TD>
     <xsl:value-of select="@HIT_BY_PITCH"/>
    </TD>
   </TR>
  </xsl:for-each>  <!— PLAYER —>
 </TBODY>
</TABLE>

<TABLE>
  <CAPTION><B>Pitchers</B></CAPTION>
  <THEAD>
   <TR>
    <TH>Player</TH><TH>P</TH><TH>G</TH>
    <TH>GS</TH><TH>W</TH><TH>L</TH><TH>S</TH>
```

Continued

Listing 5-7 *(continued)*

```
    <TH>CG</TH><TH>SO</TH><TH>ERA</TH>
    <TH>IP</TH><TH>HR</TH><TH>R</TH><TH>ER</TH>
    <TH>HB</TH><TH>WP</TH><TH>B</TH><TH>BB</TH>
    <TH>K</TH>
   </TR>
  </THEAD>
 <TBODY>
  <xsl:for-each select="PLAYER[(@POSITION
   = 'Starting Pitcher')
   $or$ (@POSITION = 'Relief Pitcher')]">
   <TR>
    <TD>
     <xsl:value-of select="@GIVEN_NAME"/>
     <xsl:value-of select="@SURNAME"/>
    </TD>
    <TD><xsl:value-of select="@POSITION"/></TD>
    <TD><xsl:value-of select="@GAMES"/></TD>
    <TD>
      <xsl:value-of select="@GAMES_STARTED"/>
    </TD>
    <TD><xsl:value-of select="@WINS"/></TD>
    <TD><xsl:value-of select="@LOSSES"/></TD>
    <TD><xsl:value-of select="@SAVES"/></TD>
    <TD>
     <xsl:value-of select="@COMPLETE_GAMES"/>
    </TD>
    <TD>
     <xsl:value-of select="@SHUT_OUTS"/>
    </TD>
    <TD><xsl:value-of select="@ERA"/></TD>
    <TD><xsl:value-of select="@INNINGS"/></TD>
    <TD>
    <xsl:value-of select="@HOME_RUNS_AGAINST"/>
    </TD>
    <TD>
     <xsl:value-of select="@RUNS_AGAINST"/>
    </TD>
    <TD>
     <xsl:value-of select="@EARNED_RUNS"/>
    </TD>
    <TD>
     <xsl:value-of select="@HIT_BATTER"/>
    </TD>
    <TD>
      <xsl:value-of select="@WILD_PITCH"/>
    </TD>
    <TD><xsl:value-of select="@BALK"/></TD>
    <TD>
     <xsl:value-of select="@WALKED_BATTER"/>
    </TD>
    <TD>
```

```
                    <xsl:value-of select="@STRUCK_OUT_BATTER"/>
                    </TD>
                  </TR>
                </xsl:for-each>  <!— PLAYER —>
              </TBODY>
            </TABLE>

          </xsl:for-each> <!— TEAM —>
        </xsl:for-each> <!— DIVISION —>
      </xsl:for-each> <!— LEAGUE —>
    </xsl:for-each> <!— SEASON —>


      <HR></HR>
      Copyright 1999
      <A HREF="http://www.macfaq.com/personal.html">
       Elliotte Rusty Harold
      </A>
      <BR />
      <A HREF="mailto:elharo@metalab.unc.edu">
       elharo@metalab.unc.edu
      </A>

      </BODY>
    </HTML>
  </xsl:template>

</xsl:stylesheet>
```


1998 Major League Baseball Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address D:\XML\Bible\CD\source\05\styled5-1.xml

East

Atlanta Braves

Batters

Player	P	G	GS	AB	R	H	D	T	HR	RBI	S	CS	SH	SF	E
Marty Malloy	Second Base	11	8	28	3	5	1	0	1	1	0	0	0	0	0
Ozzie Guillen	Shortstop	83	59	264	35	73	15	1	1	22	1	4	4	2	6
Danny Bautista	Outfield	82	27	144	17	36	11	0	3	17	1	0	3	2	2
Gerald Williams	Outfield	129	51	266	46	81	18	3	10	44	11	5	2	1	5
Javier Lopez	Catcher	133	124	489	73	139	21	1	34	106	5	3	1	8	5
Ryan Klesko	Outfield	129	124	427	69	117	29	1	18	70	5	3	0	4	2
Andres Galarraga	First Base	153	151	555	103	169	27	1	44	121	7	6	0	5	11
Wes Helms	Third Base	7	2	13	2	4	1	0	1	2	0	0	0	0	1

Pitchers

Player	P	G	GS	W	L	S	CG	SO	ERA	IP	HR	R	ER	HB	WP	B
Tom Glavine	Starting Pitcher	33	33	20	6	0	4	3	2.47	229.1	13	67	63	2		0

Florida Marlins

Figure 5-9: Pitchers are distinguished from other players after applying the XSL style sheet in Listing 5-7.

Element Contents and the select Attribute

In this chapter, I focused on using XSL to format data stored in the attributes of an element because it isn't accessible when using CSS. However, XSL works equally well when you want to include an element's character data rather than (or in addition to) its attributes. To indicate that an element's text is to be copied into the output document, simply use the element's name as the value of the `select` attribute of the `xsl:value-of` element. For example, consider, once again, Listing 5-8:

```
Listing 5-8greeting.xml<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="greeting.xsl"?>
<GREETING>
Hello XML!
</GREETING>
```

Let's suppose you want to copy the greeting "Hello XML!" into an `H1` header. First, you use `xsl:for-each` to select the `GREETING` element:

```
<xsl:for-each select="GREETING">
   <H1>
   </H1>
</xsl:for-each>
```

This alone is enough to copy the two `H1` tags into the output. To place the text of the `GREETING` element between them, use `xsl:value-of` with no `select` attribute. Then, by default, the contents of the current element (`GREETING`) are selected. Listing 5-9 shows the complete style sheet.

Listing 5-9: greeting.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <xsl:for-each select="GREETING">
          <H1>
            <xsl:value-of/>
          </H1>
        </xsl:for-each>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

You can also use select to choose the contents of a child element. Simply make the name of the child element the value of the select attribute of xsl:value-of. For instance, consider the baseball example from the previous chapter in which each player's statistics were stored in child elements rather than in attributes. Given this structure of the document (which is actually far more likely than the attribute-based structure of this chapter) the XSL for the batters' table looks like this:

```
<TABLE>
  <CAPTION><B>Batters</B></CAPTION>
  <THEAD>
   <TR>
    <TH>Player</TH><TH>P</TH><TH>G</TH>
    <TH>GS</TH><TH>AB</TH><TH>R</TH><TH>H</TH>
    <TH>D</TH><TH>T</TH><TH>HR</TH><TH>RBI</TH>
    <TH>S</TH><TH>CS</TH><TH>SH</TH><TH>SF</TH>
    <TH>E</TH><TH>BB</TH><TH>SO</TH><TH>HBP</TH>
   </TR>
  </THEAD>
 <TBODY>
  <xsl:for-each select="PLAYER[(POSITION
   != 'Starting Pitcher')
   $and$ (POSITION != 'Relief Pitcher')]">
   <TR>
    <TD>
     <xsl:value-of select="GIVEN_NAME"/>
     <xsl:value-of select="SURNAME"/>
    </TD>
    <TD><xsl:value-of select="POSITION"/></TD>
    <TD><xsl:value-of select="GAMES"/></TD>
    <TD>
      <xsl:value-of select="GAMES_STARTED"/>
    </TD>
    <TD><xsl:value-of select="AT_BATS"/></TD>
    <TD><xsl:value-of select="RUNS"/></TD>
    <TD><xsl:value-of select="HITS"/></TD>
    <TD><xsl:value-of select="DOUBLES"/></TD>
    <TD><xsl:value-of select="TRIPLES"/></TD>
    <TD><xsl:value-of select="HOME_RUNS"/></TD>
    <TD><xsl:value-of select="RBI"/></TD>
    <TD><xsl:value-of select="STEALS"/></TD>
    <TD>
     <xsl:value-of select="CAUGHT_STEALING"/>
    </TD>
    <TD>
     <xsl:value-of select="SACRIFICE_HITS"/>
    </TD>
    <TD>
     <xsl:value-of select="SACRIFICE_FLIES"/>
    </TD>
    <TD><xsl:value-of select="ERRORS"/></TD>
```

```
          <TD><xsl:value-of select="WALKS"/></TD>
          <TD>
           <xsl:value-of select="STRUCK_OUT"/>
          </TD>
          <TD>
           <xsl:value-of select="HIT_BY_PITCH"/>
          </TD>
        </TR>
      </xsl:for-each>  <!— PLAYER —>
     </TBODY>
    </TABLE>
```

In this case, within each PLAYER element, the contents of that element's GIVEN_NAME, SURNAME, POSITION, GAMES, GAMES_STARTED, AT_BATS, RUNS, HITS, DOUBLES, TRIPLES, HOME_RUNS, RBI, STEALS, CAUGHT_STEALING, SACRIFICE_HITS, SACRIFICE_FLIES, ERRORS, WALKS, STRUCK_OUT and HIT_BY_PITCH children are extracted and copied to the output. Since we used the same names for the attributes in this chapter as we did for the PLAYER child elements in the last chapter, this example is almost identical to the equivalent section of Listing 5-7. The main difference is that the @ signs are missing. They indicate an attribute rather than a child.

You can do even more with the select attribute. You can select elements: by position (for example, the first, second, last, seventeenth element, and so forth); with particular contents; with specific attribute values; or whose parents or children have certain contents or attribute values. You can even apply a complete set of Boolean logical operators to combine different selection conditions. We will explore more of these possibilities when we return to XSL in Chapter 14.

CSS or XSL?

CSS and XSL overlap to some extent. XSL is certainly more powerful than CSS. However XSL's power is matched by its complexity. This chapter only touched on the basics of what you can do with XSL. XSL is more complicated, and harder to learn and use than CSS, which raises the question, "When should you use CSS and when should you use XSL?"

CSS is more broadly supported than XSL. Parts of CSS Level 1 are supported for HTML elements by Netscape 4 and Internet Explorer 4 (although annoying differences exist). Furthermore, most of CSS Level 1 and some of CSS Level 2 is likely to be well supported by Internet Explorer 5.0 and Mozilla 5.0 for both XML and HTML. Thus, choosing CSS gives you more compatibility with a broader range of browsers.

Additionally, CSS is more stable. CSS level 1 (which covers most of the CSS you've seen so far) and CSS Level 2 are W3C recommendations. XSL is still a very early working draft, and won't be finalized until after this book is printed. Early adopters of XSL have already been burned once, and will be burned again before standards gel. Choosing CSS means you're less likely to have to rewrite your style sheets from month to month just to track evolving software and standards. Eventually, however, XSL will settle down to a usable standard.

Furthermore, since XSL is so new, different software implements different variations and subsets of the draft standard. At the time of this writing (spring 1999) there are at least three major variants of XSL in widespread use. Before this book is published, there will be more. If the incomplete and buggy implementations of CSS in current browsers bother you, the varieties of XSL will drive you insane.

However, XSL is definitely more powerful than CSS. CSS only allows you to apply formatting to element contents. It does not allow you to change or reorder those contents; choose different formatting for elements based on their contents or attributes; or add simple, extra text like a signature block. XSL is far more appropriate when the XML documents contain only the minimum of data and none of the HTML frou-frou that surrounds the data.

With XSL, you can separate the crucial data from everything else on the page, like mastheads, navigation bars, and signatures. With CSS, you have to include all these pieces in your data documents. XML+XSL allows the data documents to live separately from the Web page documents. This makes XML+XSL documents more maintainable and easier to work with.

In the long run XSL should become the preferred choice for real-world, data-intensive applications. CSS is more suitable for simple pages like grandparents use to post pictures of their grandchildren. But for these uses, HTML alone is sufficient. If you've really hit the wall with HTML, XML+CSS doesn't take you much further before you run into another wall. XML+XSL, by contrast, takes you far past the walls of HTML. You still need CSS to work with legacy browsers, but long-term XSL is the way to go.

Summary

In this chapter, you saw examples of creating an XML document from scratch. Specifically, you learned:

✦ Information can also be stored in an attribute of an element.

✦ An attribute is a name-value pair included in an element's start tag.

✦ Attributes typically hold meta-information about the element rather than the element's data.

✦ Attributes are less convenient to work with than the contents of an element.

✦ Attributes work well for very simple information that's unlikely to change its form as the document evolves. In particular, style and linking information works well as an attribute.

✦ Empty tags offer syntactic sugar for elements with no content.

✦ XSL is a powerful style language that enables you to access and display attribute data and transform documents.

In the next chapter, we'll specify the exact rules that well-formed XML documents must adhere to. We'll also explore some additional means of embedding information in XML documents including comments and processing instructions.

✦ ✦ ✦

CHAPTER 12

Cascading Style Sheets Level 1

CSS is a very simple and straightforward language for applying styles such as bold and Helvetica to particular XML elements. Most of the styles CSS supports are familiar from any conventional word processor. For example, you can choose the font, the font weight, the font size, the background color, the spacing of various elements, the borders around elements, and more. However, rather than being stored as part of the document itself, all the style information is placed in a separate document called a style sheet. One XML document can be formatted in many different ways just by changing the style sheet. Different style sheets can be designed for different purposes — for print, the Web, presentations, and for other uses — all with the styles appropriate for the specific medium, and all without changing any of the content in the document itself.

What Is CSS?

Cascading Style Sheets (referred to as CSS from now on) were introduced in 1996 as a standard means of adding information about style properties such as fonts and borders to HTML documents. However, CSS actually works better with XML than with HTML because HTML is burdened with backwards-compatibility between the CSS tags and the HTML tags. For instance, properly supporting the CSS `nowrap` property requires eliminating the non-standard but frequently used `NOWRAP` element in HTML. Because XML elements don't have any predefined formatting, they don't restrict which CSS styles can be applied to which elements.

A CSS style sheet is a list of rules. Each rule gives the names of the elements it applies to and the styles it wants to apply to those elements. For example, consider Listing 12-1, a CSS style sheet for poems. This style sheet has five rules. Each rule has a selector—the name of the element to which it applies—and a list of properties to apply to instances of that element. The first rule says that the contents of the POEM element should be displayed in a block by itself (display: block). The second rule says that the contents of the TITLE element should be displayed in a block by itself (display: block) in 16-point (font-size: 16pt) bold type (font-weight: bold). The third rule says that the POET element should be displayed in a block by itself (display: block) and should be set off from what follows it by 10 pixels (margin-bottom: 10px). The fourth rule is the same as the third rule except that it applies to STANZA elements. Finally, the fifth rule simply states that VERSE elements are also displayed in their own block.

Listing 12-1: A CSS style sheet for poems

```
POEM   { display: block }
TITLE  { display: block; font-size: 16pt; font-weight: bold }
POET   { display: block; margin-bottom: 10px }
STANZA { display: block; margin-bottom: 10px }
VERSE  { display: block }
```

In 1998, the W3C published a revised and expanded specification for CSS called CSS Level 2 (CSS2). At the same time, they renamed the original CSS to CSS Level 1 (CSS1). CSS2 is mostly a superset of CSS1, with a few minor exceptions, which I'll note as we encounter them. In other words, CSS2 is CSS1 plus aural style sheets, media types, attribute selectors, and other new features. Consequently, almost everything said in this chapter applies to both CSS1 and CSS2. CSS2 will be covered in the next chapter as an extension to CSS1.

Parts of CSS Level 1 are supported by Netscape Navigator 4.0 and Internet Explorer 4.0 and 5.0. Unfortunately, they often aren't the same parts. Mozilla 5.0 is supposed to provide no-uncompromising support for CSS Level 1 and most of CSS Level 2. Internet Explorer 5.0 does a better job than Internet Explorer 4.0 but it's still missing some major pieces, especially in regards to the box model and pseudo-elements. I'll try to point out areas in which one or the other browser has a particularly nasty problem.

Attaching Style Sheets to Documents

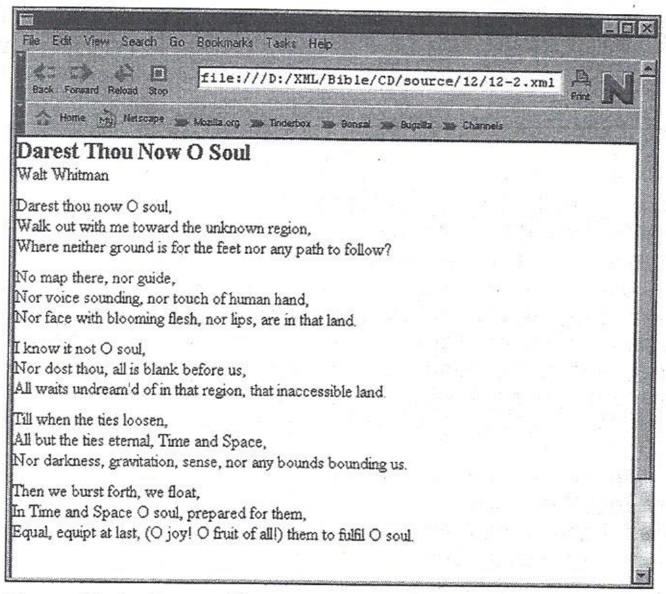
To really make sense out of the style sheet in Listing 12-1, you have to give it an XML document to play with. Listing 12-2 is a poem from Walt Whitman's classic book of poetry, *Leaves of Grass,* marked up in XML. The second line is the <?xml-stylesheet?> processing instruction that instructs the Web browser loading this document to apply the style sheet found in the file poem.css to this document. Figure 12-1 shows this document loaded into an early alpha of Mozilla.

Listing 12-2: *Darest Thou Now O Soul* marked up in XML

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="poem.css"?>
<POEM>

  <TITLE>Darest Thou Now O Soul</TITLE>
  <POET>Walt Whitman</POET>

  <STANZA>
    <VERSE>Darest thou now O soul,</VERSE>
    <VERSE>Walk out with me toward the unknown region,</VERSE>
    <VERSE>Where neither ground is for the feet nor
           any path to follow?</VERSE>
  </STANZA>
   <STANZA>
     <VERSE>No map there, nor guide,</VERSE>
     <VERSE>Nor voice sounding, nor touch of
            human hand,</VERSE>
     <VERSE>Nor face with blooming flesh, nor lips,
            are in that land.</VERSE>
  </STANZA>
   <STANZA>
     <VERSE>I know it not O soul,</VERSE>
     <VERSE>Nor dost thou, all is blank before us,</VERSE>
     <VERSE>All waits undream'd of in that region,
            that inaccessible land.</VERSE>
  </STANZA>
   <STANZA>
     <VERSE>Till when the ties loosen,</VERSE>
     <VERSE>All but the ties eternal, Time and Space,</VERSE>
     <VERSE>Nor darkness, gravitation, sense,
            nor any bounds bounding us.</VERSE>
  </STANZA>
   <STANZA>
     <VERSE>Then we burst forth, we float,</VERSE>
     <VERSE>In Time and Space O soul,
            prepared for them,</VERSE>
     <VERSE>Equal, equipt at last, (O joy! O fruit of all!)
           them to fulfil O soul.</VERSE>
  </STANZA>

</POEM>
```

The `type` attribute in the `<?xml-stylesheet?>` processing instruction is the MIME type of the style sheet you're using. Its value is `text/css` for CSS and `text/xsl` for XSL.

Cross-Reference: CSS Level 2 is discussed in Chapter 13. XSL is covered in Chapters 14 and 15.


File Edit View Search Go Bookmarks Tasks Help
Back Forward Reload Stop
file:///D:/XML/Bible/CD/source/12/12-2.xml
Print
Home Netscape Mozilla.org Tinderbox Bonsai Bugzilla Channels

Darest Thou Now O Soul
Walt Whitman

Darest thou now O soul,
Walk out with me toward the unknown region,
Where neither ground is for the feet nor any path to follow?

No map there, nor guide,
Nor voice sounding, nor touch of human hand,
Nor face with blooming flesh, nor lips, are in that land.

I know it not O soul,
Nor dost thou, all is blank before us,
All waits undream'd of in that region, that inaccessible land.

Till when the ties loosen,
All but the ties eternal, Time and Space,
Nor darkness, gravitation, sense, nor any bounds bounding us.

Then we burst forth, we float,
In Time and Space O soul, prepared for them,
Equal, equipt at last, (O joy! O fruit of all!) them to fulfil O soul.

Figure 12-1: *Darest Thou Now O Soul* as rendered by Mozilla

The value of the `href` attribute in the `<?xml-stylesheet?>` processing instruction is a URL, often relative, where the style sheet is found. If the style sheet can't be found, the Web browser will probably use its default style sheet though some browsers may report an error instead.

You can apply the same style sheet to many documents. Indeed, you generally will. Thus, it's common to put your style sheets in some central location on your Web server where all of your documents can refer to them; a convenient location is the styles directory at the root level of the Web server.

```
<?xml-stylesheet type="text/css" href="/styles/poem.css"?>
```

You might even use an absolute URL to a style sheet on another Web site, though of course this does leave your site dependent on the status of the external Web site.

```
<?xml-stylesheet type="text/css"
      href="http://metalab.unc.edu/xml/styles/poem.css"?>
```

You can even use multiple `<?xml-stylesheet?>` processing instructions to pull in rules from different style sheets. For example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="/styles/poem.css"?>
<?xml-stylesheet type="text/css"
      href="http://metalab.unc.edu/xml/styles/poem.css"?>
<POEM>
...
```

CSS with HTML versus CSS with XML

Although the focus of this book is on XML, CSS style sheets also work with HTML documents. The main differences between CSS with HTML and CSS with XML are:

1. The elements you can attach a rule to are limited to standard HTML elements like P, PRE, LI, DIV, and SPAN.
2. HTML browsers don't recognize processing instructions, so style sheets are attached to HTML documents using LINK tags in the HEAD element. Furthermore, per-document style rules can be included in the HEAD in a STYLE element. For example:

```
<LINK REL=STYLESHEET TYPE="text/css" HREF="/styles/poem.css" >
<STYLE TYPE="text/css">
  PRE { color: red }
</STYLE>
```

3. HTML browsers don't render CSS properties as faithfully as XML browsers because of the legacy formatting of elements. Tables are notoriously problematic in this respect.

Note

Style sheets are more or less orthogonal to DTDs. A document with a style sheet may or may not have a DTD and a document with a DTD may or may not have a style sheet. However, DTDs do often serve as convenient lists of the elements that you need to provide style rules for.

In this and the next several chapters, most of the examples will use documents that are well-formed, but not valid. The lack of DTDs will make the examples shorter and the relevant parts more obvious. However in practice, most of the documents you attach style sheets to will probably be valid documents with DTDs.

Selection of Elements

The part of a CSS rule that specifies which elements it applies to is called a *selector.* The most common kind of selector is simply the name of an element; for instance TITLE in this rule:

```
TITLE { display: block; font-size: 16pt; font-weight: bold }
```

However, selectors can also specify multiple elements, elements with a particular CLASS or ID attribute and elements that appear in particular contexts relative to other elements.

Note

One thing you cannot do in CSS Level 1 is select elements with particular attribute names or values other than the predefined CLASS and ID attributes. To do this, you have to use CSS Level 2 or XSL.

Grouping Selectors

If you want to apply one set of properties to multiple elements, you can include all the elements in the selector separated by commas. For instance, in Listing 12-1 `POET` and `STANZA` were both styled as block display with a 10-pixel margin. You can combine these two rules like this:

```
POET, STANZA { display: block; margin-bottom: 10px }
```

Furthermore, more than one rule can apply style to a particular element. So you can combine some standard properties into a rule with many selectors, then use more specific rules to apply custom formatting to selected elements. For instance, in Listing 12-1 all the elements were listed as block display. This can be combined into one rule while additional formatting for the `POET`, `STANZA`, and `TITLE` elements is contained in separate rules, like this:

```
POEM, VERSE, TITLE, POET, STANZA { display: block }
POET, STANZA { margin-bottom: 10px }
TITLE {font-size: 16pt; font-weight: bold }
```

Pseudo-Elements

CSS1 supports two pseudo-elements that can address parts of the document that aren't normally identified as separate elements, but nonetheless often need separate styles. These are the first line and the first letter of an element.

Caution

The early betas of Internet Explorer 5.0 and earlier versions of Internet Explorer do not support these pseudo-elements. The early beta of Mozilla 5.0 *does* support them, but only for HTML.

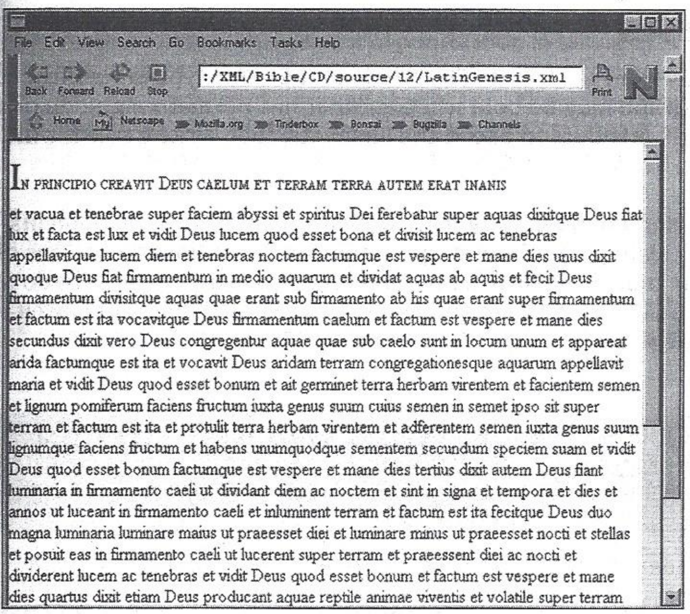
Addressing the First Letter

The most common reason to format the first letter of an element separately from the rest of the element is to insert a drop cap as shown in Figure 12-2. This is accomplished by writing a rule that is addressed with the element name, followed by `:first-letter`. For example:

```
CHAPTER:first-letter { font-size: 300%;
                       float: left; vertical-align: text-top }
```

Caution

As you may notice in Figure 12-2, the "drop" part of the drop cap (`float: left; vertical-align: text-top`) does not yet seem to work in either the early betas of Mozilla 5.0 or Internet Explorer 5.0, though the size of the initial letter can be adjusted.


File Edit View Search Go Bookmarks Tasks Help
Back Forward Reload Stop
:/XML/Bible/CD/source/12/LatinGenesis.xml
Print
Home My Netscape Mozilla.org Tinderbox Bonsai Bugzilla Channels

IN PRINCIPIO CREAVIT DEUS CAELUM ET TERRAM TERRA AUTEM ERAT INANIS
et vacua et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas dixitque Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona et divisit lucem ac tenebras appellavitque lucem diem et tenebras noctem factumque est vespere et mane dies unus dixit quoque Deus fiat firmamentum in medio aquarum et dividat aquas ab aquis et fecit Deus firmamentum divisitque aquas quae erant sub firmamento ab his quae erant super firmamentum et factum est ita vocavitque Deus firmamentum caelum et factum est vespere et mane dies secundus dixit vero Deus congregentur aquae quae sub caelo sunt in locum unum et appareat arida factumque est ita et vocavit Deus aridam terram congregationesque aquarum appellavit maria et vidit Deus quod esset bonum et ait germinet terra herbam virentem et facientem semen et lignum pomiferum faciens fructum iuxta genus suum cuius semen in semet ipso sit super terram et factum est ita et protulit terra herbam virentem et adferentem semen iuxta genus suum lignumque faciens fructum et habens unumquodque sementem secundum speciem suam et vidit Deus quod esset bonum factumque est vespere et mane dies tertius dixit autem Deus fiant luminaria in firmamento caeli ut dividant diem ac noctem et sint in signa et tempora et dies et annos ut luceant in firmamento caeli et inluminent terram et factum est ita fecitque Deus duo magna luminaria luminare maius ut praeesset diei et luminare minus ut praeesset nocti et stellas et posuit eas in firmamento caeli ut lucerent super terram et praeessent diei ac nocti et dividerent lucem ac tenebras et vidit Deus quod esset bonum et factum est vespere et mane dies quartus dixit etiam Deus producant aquae reptile animae viventis et volatile super terram

Figure 12-2: A drop cap on the first-letter pseudo element with small caps used on the first-line pseudo-element

Addressing the First Line

The first line of an element is also often formatted differently than the remainder of the text of the element. For instance, it may be printed in small caps instead of normal body text as shown in Figure 12-2. You can attach the `:first-line` selector to the name of an element to create a rule that only applies to the first line of the element. For example,

```
CHAPTER:first-line { font-variant: small-caps }
```

Exactly what this pseudo-element selects is relative to the current layout. If the window is larger and there are more words in the first line, then more words will be in small caps. If the window is made smaller or the font gets larger so the text wraps differently and fewer words are on the first line, then the words that are wrapped to the next line are no longer in small caps. The determination of which characters comprise the `first-line` pseudo-element is deferred until the document is actually displayed.

Pseudo-Classes

Sometimes you may want to style two elements of the same type differently. For example, one paragraph may be bold while another has normal weight. To do this, you can add a CLASS attribute to one of the elements, then write a rule for the elements in a given CLASS.

For example, consider a bibliography that contains many CITATION elements. A sample is shown in Listing 12-3. Now suppose you want to color all citations of the work of Alan Turing blue, while leaving the other citations untouched. To do this you have to add a CLASS attribute with a specific value — TURING works well — to the elements to be colored.

Listing 12-3: A bibliography in XML with three CITATION elements

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="biblio.css"?>
<BIBLIOGRAPHY>
  <CITATION CLASS="HOFSTADTER" ID="C1">
      <AUTHOR>Hofstadter, Douglas</AUTHOR>.
      "<TITLE>How Might Analogy, the Core of Human Thinking,
        Be Understood By Computers?</TITLE>"
      <JOURNAL>Scientific American</JOURNAL>,
      <MONTH>September</MONTH>
      <YEAR>1981</YEAR>
      <PAGES>18-30</PAGES>
  </CITATION>
  <CITATION CLASS="TURING" ID="C2">
    <AUTHOR>Turing, Alan M.</AUTHOR>
    "<TITLE>On Computable Numbers,
      With an Application to the Entscheidungs-problem</TITLE>"
    <JOURNAL>
      Proceedings of the London Mathematical Society</JOURNAL>,
    <SERIES>Series 2</SERIES>,
    <VOLUME>42</VOLUME>
    (<YEAR>1936</YEAR>):
    <PAGES>230-65</PAGES>.
  </CITATION>
  <CITATION CLASS="TURING" ID="C3">
    <AUTHOR>Turing, Alan M.</AUTHOR>
    "<TITLE>Computing Machinery &amp; Intelligence</TITLE>"
    <JOURNAL>Mind</JOURNAL>
    <VOLUME>59</VOLUME>
    (<MONTH>October</MONTH>
    <YEAR>1950</YEAR>):
    <PAGES>433-60</PAGES>
  </CITATION>
</BIBLIOGRAPHY>
```

Note

One of the more annoying aspects of CSS Level 1 is that it makes mixed content more necessary. There's a lot of punctuation in Listing 12-3 that is not really part of the content; for example the parentheses placed around the `YEAR` element and the quotation marks around the `TITLE`. These are presentation elements that should be part of the style sheet instead. CSS Level 2 allows extra text such as punctuation to be inserted before and after elements.

The style sheet in Listing 12-4 uses a `CLASS` selector to color elements in the `TURING` class blue.

Caution

`CLASS` attributes are supported by IE5 but not by Mozilla as of the milestone 3 release. Mozilla will probably support `CLASS` attributes by the time it's officially released.

Listing 12-4: A style sheet that colors elements in the TURING class blue

```
BIBLIOGRAPHY { display: block }
CITATION.TURING { color: blue }
CITATION { display: block }
JOURNAL  { font-style: italic }
```

Note

In a valid document, the `CLASS` attribute must be declared as a possible attribute of the styled elements. For example, here's a DTD for the bibliography of Listing 12-3:

```
<!ELEMENT BIBLIOGRAPHY (CITATION*)>
<!ATTLIST CITATION CLASS CDATA #IMPLIED>
<!ATTLIST CITATION ID    ID    #REQUIRED>

<!ELEMENT CITATION   ANY>
<!ELEMENT AUTHOR    (#PCDATA)>
<!ELEMENT TITLE     (#PCDATA)>
<!ELEMENT JOURNAL   (#PCDATA)>
<!ELEMENT MONTH     (#PCDATA)>
<!ELEMENT YEAR      (#PCDATA)>
<!ELEMENT SERIES    (#PCDATA)>
<!ELEMENT VOLUME    (#PCDATA)>
<!ELEMENT PAGES     (#PCDATA)>
```

In general, I do not recommend this approach. You should, if possible, attempt to add additional element markup to the document rather than relying on `CLASS` attributes. However, `CLASS` attributes may be necessary when the information you're selecting does not conveniently map to particular elements.

Selection by ID

Sometimes, a unique element needs a unique style. You need a rule that applies to exactly that one element. For instance, suppose you want to make one element in a list bold to really emphasize it in contrast to its siblings. In this case, you can write a rule that applies to the `ID` attribute of the element. The selector is the name of the element, followed by a # and the value of the `ID` attribute.

For example, Listing 12-5 is a style sheet that selects the `CITATION` element from the bibliography in Listing 12-3 with the ID `C3` and makes it, and only it, bold. Other `CITATION` elements appear with the default weight. All `CITATION` elements are displayed in block fashion and all `JOURNAL` elements are italicized.

Listing 12-5: A style sheet that makes the CITATION element with ID C3 bold

```
BIBLIOGRAPHY { display: block }
CITATION#C3  { font-weight: bold }
CITATION     { display: block }
JOURNAL      { font-style: italic }
```

Caution

`ID` selectors are supported by IE5, and by Mozilla for HTML elements, but not XML elements as of the milestone 3 release. Mozilla will probably fully support `ID` selectors by the time it's officially released.

Contextual Selectors

Often, the formatting of an element depends on its parent element. You can write rules that only apply to elements found inside a named parent. To do this, prefix the name of the parent element to the name of the styled element.

For example, a `CODE` element inside a `PRE` element may be rendered in 12-point Courier. However, if the body text of the document is written in 10-point Times, a `CODE` element that's inline with other body text may need to be rendered in 10-point Courier. The following rules accomplish exactly that:

```
BODY     { font-family: Times, serif; font-size: 10pt }
CODE     { font-family: Courier, monospaced; font-size: 10pt }
PRE      { font-size: 12pt }
PRE CODE { font-size: 12pt }
```

This says that inside the `BODY` element, the font is 10-point Times. However, inside a `CODE` element the font changes to Courier, still 10-point. However, if the `CODE` element is inside a `PRE` element then the font grows to 12 points.

You can expand this to look at the parent of the parent, the parent of the parent of the parent, and so forth. For example, the following rule says that a NUMBER element inside a YEAR element inside a DATE element should be rendered in a monospaced font:

```
DATE YEAR NUMBER { font-family: Courier, monospaced }
```

In practice, this level of specificity is rarely needed. In cases in which it does seem to be needed, you can often rewrite your style sheet to rely more on inheritance, cascades, and relative units, and less on the precise specification of formatting.

STYLE Attributes

When hand-authoring documents, it's not uncommon to want to apply a particular style one time to a particular element without editing the style sheet for the document. Indeed, you may want to override some standard default style sheet for the document that you can't change. You can do this by attaching a STYLE attribute to the element. The value of this attribute is a semicolon-separated list of style properties for the element. For example, this CITATION uses a STYLE attribute to make itself bold:

```
<CITATION CLASS="TURING" ID="C3" STYLE="font-weight: bold">
  <AUTHOR>Turing, Alan M.</AUTHOR>
  "<TITLE>Computing Machinery &amp; Intelligence</TITLE>"
  <JOURNAL>Mind</JOURNAL>
  <VOLUME>59</VOLUME>
  (<MONTH>October</MONTH>
  <YEAR>1950</YEAR>):
  <PAGES>433-60</PAGES>
</CITATION>
```

If the properties defined in a STYLE attribute conflict with the properties defined in the style sheet, then the properties defined in the attribute take precedence.

Avoid using STYLE attributes if at all possible. Your documents will be much cleaner and more maintainable if you keep all style information in separate style sheets. Nonetheless, there are times when STYLE attributes are too quick and convenient to ignore.

Again, if you use this approach in a valid document, you will need to declare the STYLE attribute in an ATTLIST declaration for the element you're styling. For example:

```
<!ELEMENT CITATION ANY>
<!ATTLIST CITATION CLASS CDATA #IMPLIED>
<!ATTLIST CITATION ID    ID    #REQUIRED>
<!ATTLIST CITATION STYLE CDATA #IMPLIED>
```

Caution

STYLE attributes are supported by IE5, and by Mozilla for HTML elements, but not XML elements as of the milestone 3 release. Mozilla will probably fully support STYLE attributes by the time it's officially released.

Inheritance

CSS does not require that rules be specifically defined for each possible property of each element in a document. For instance, if there is not a rule that specifies the font size of an element, then the element inherits the font size of its parent. If there is not a rule that specifies the color of an element, then the element inherits the color of its parent. The same is true of most CSS properties. In fact, the only properties that aren't inherited are the background and box properties.

For example, consider these rules:

```
P     { font-weight: bold;
        font-size: 24pt;
        font-family: sans-serif}
BOOK  { font-style: italic; font-family: serif}
```

Now consider this XML fragment:

```
<P>
  Michael Willhoite's <BOOK>Daddy's Roommate</BOOK> is
  the #10 most frequently banned book in the U.S. in the 1990s.
</P>
```

Although the BOOK element has not been specifically assigned a font-weight or a font-size, it will be rendered in 24-point bold because it is a child of the P element. It will also be italicized because that is specified in its own rule. BOOK *inherits* the font-weight and font-size of its parent P. If later in the document a BOOK element appears in the context of some other element, then it will inherit the font-weight and font-size of that element.

The font-family is a little trickier because both P and BOOK declare conflicting values for this property. Inside the BOOK element, the font-family declared by BOOK takes precedence. Outside the BOOK element, P's font-family is used. Therefore, "Daddy's Roommate" is drawn in a serif font, while "most frequently banned book" is drawn in a sans serif font.

Often you want the child elements to inherit formatting from their parents. Therefore, it's important not to over-specify the formatting of any element. For instance, suppose I had declared that BOOK was written in 12-point font like this:

```
BOOK { font-style: italic; font-family: serif; font-size: 12pt}
```

Then the example would be rendered as shown in Figure 12-3:

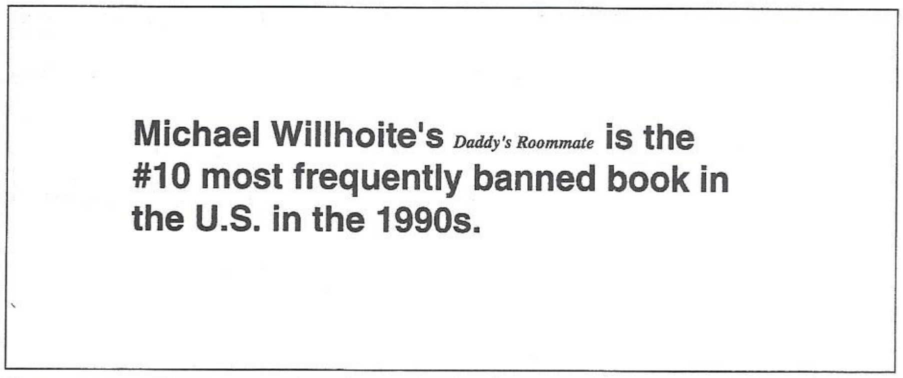

Michael Willhoite's *Daddy's Roommate* is the #10 most frequently banned book in the U.S. in the 1990s.

Figure 12-3: The BOOK written in a 12-point font size

You could fix this with a special rule that uses a contextual selector to pick out `BOOK` elements inside `P` elements, but it's easier to simply inherit the parent's `font-size`.

One way to avoid problems like this, while retaining some control over the size of individual elements is to use relative units like ems and ex's instead of absolute units like points, picas, inches, centimeters, and millimeters. An em is the width of the letter *m* in the current font. An ex is the height of the letter *x* in the current font. If the font gets bigger, so does everything measured in ems and ex's.

A similar option that's available for some properties is to use percentage units. For example, the following rule sets the font size of the `FOOTNOTE_NUMBER` element to 80 percent of the font size of the parent element. If the parent element's font size increases or decreases, `FOOTNOTE_NUMBER`'s font size scales similarly.

```
FOOTNOTE_NUMBER { font-size: 80% }
```

Exactly what the percentage is a percentage of varies from property to property. In the `vertical-align` property, the percentage is of the line height of the element itself. However in a margin property, a percentage is a percentage of the element's width.

Cascades

It is possible to attach more than one style sheet to a document. For instance, a browser may have a default style sheet which is added to the one the designer provides for the page. In such a case, it's possible that there will be multiple rules that apply to one element, and these rules may conflict. Thus, it's important to determine in which order the rules are applied. This process is called a *cascade*, and is where cascading style sheets get their name.

There are several ways a CSS style sheet can be attached to an XML document:

1. The `<?xml-stylesheet?>` processing instruction can be included in the XML document.
2. The style sheet itself may import other style sheets using `@import`.
3. The user may specify a style sheet for the document using mechanisms inside their browser.
4. The browser provides default styles for most properties.

The @import Directive

Style sheets may contain `@import` directives that load style sheets stored in other files. An absolute or relative URL is used to identify the style sheets. For example,

```
@import url(http://www.w3.org/basicstyles.css);
@import url(/styles/baseball.css);
```

These `@import` directives must appear at the beginning of the style sheet, before any rules. Rules in the importing style sheet always override those in the imported style sheets. The imported style sheets cascade in the order they're imported. Cycles (for example, poem.css imports stanza.css which imports poem.css) are prohibited.

The !important Declaration

In CSS1, author rules override reader rules unless the reader attaches an `!important` declaration to the property. For example, the following rule says that the `TITLE` element should be colored blue even if the author of the document requested a different color. On the other hand, the `font-family` should be `serif` only if the author rules don't disagree.

```
TITLE { color: blue !important font-family: serif}
```

However, author rules may also be declared important. In such a case, the author rule overrides the reader rule.

Note

This is a very bad idea. Readers should always have the option to choose the way they view something. It simply isn't possible to write one style sheet that's appropriate for people using color and black-and-white monitors, the seeing and the sight-impaired, people browsing on 21-inch monitors, television sets, and PDAs. Too many Web designers vastly over-specify their styles, only to produce pages that are completely unreadable on systems that aren't exactly like their own. Fortunately, CSS2 reverses this precedence so that reader rules have the ultimate say.

Cascade Order

Styles are chosen from the available style rules for an element. In general, more specific rules win. For instance, consider this fragment:

```
<OUEVRE>
  <PLAY ID="x02" CLASS="WILDE">
    The Importance of Being Earnest
  </PLAY>
</OUEVRE>
```

The most specific rules are preferred. Thus, one that selected the PLAY element by its ID would be preferred to one that selected the PLAY by its CLASS. A rule that selected the PLAY by its CLASS would be preferred to one that selected PLAY elements contained in OUEVRE elements. Finally, if none of those applied, a generic PLAY rule would be selected. If no selector matches, the value inherited from the parent element is used. If there is no value inherited from the parent element, the default value is used.

If there is more than one rule at a given level of specificity, the cascading order is resolved in the following order of preference:

1. Author declarations marked important.
2. Reader declarations marked important.
3. Author declarations not marked important.
4. Reader declarations not marked important.
5. The latest rule in the style sheet.

Tip

Try to avoid depending on cascading order. It's rarely a mistake to specify as little style as possible and let the reader/browser preferences take control.

Comments in CSS Style Sheets

CSS style sheets can include comments. CSS comments are like C's /* */ comments, not like <!– –> XML and HTML comments. Listing 12-6 demonstrates. This style sheet doesn't merely apply style rules to elements. It also describes, in English, the results those style rules are supposed to achieve.

Listing 12-6: A style sheet for poems with comments

```
/* Work around a Mozilla bug */
POEM { display:block }

/* Make the title look like an H1 header */
TITLE  { display: block; font-size: 16pt; font-weight: bold }
POET   { display: block; margin-bottom: 10 }

/* Put a blank line in-between stanzas,
   only a line break between verses */
STANZA { display: block; margin-bottom: 10 }
VERSE  { display: block }
```

CSS isn't nearly as convoluted as XML DTDs, Java, C, or Perl, so comments aren't quite as necessary as they are in other languages. However, it's rarely a bad idea to include comments. They can only help someone who's trying to make sense out of a style sheet you wrote and who is unable to ask you directly.

CSS Units

CSS properties have names and values. Table 12-1 lists some of these property names and some of their values.

The names are all CSS keywords. However, the values are much more diverse. Some of them are keywords like the `none` in `display: none` or the `solid` in `border-style: solid`. Other values are numbers with units like the `0.5in` in `margin-top: 0.5in` or the `12pt` in `font-size: 12pt`. Still other values are URLs like the `http://www.idgbooks.com/images/paper.gif` in `background-image: url(http://www.idgbooks.com/images/paper.gif)` or RGB colors like the `#CC0033` in `color: #CC0033`. Different properties permit different values. However, there are only four different kinds of values a property may take on. These are:

1. length
2. URL
3. color
4. keyword

Keywords vary from property to property, but the other kinds of values are the same from property to property. That is, a length is a length is a length regardless of which property it's the value of. If you know how to specify the length of a