CHAPTER 7

Tables and Layout

A table is an orderly arrangement of data distributed across a grid of rows and columns similar to a spreadsheet. In printed documents, tables commonly serve a subordinate function, illustrating some point described by accompanying text. Tables still perform this illustrative function in HTML documents. However, because HTML alone does not offer the same layout capacities available to print designers, Web page tables also are commonly used to structure a page for layout. But unlike printed tables, HTML tables can contain information that is *dynamic,* or even interactive, such as the results of a database query. We'll see that even when their impressive layout duties are retired in favor of CSS, tables will still have an important role in the Web developer's toolbox.

Introduction to Tables

In its simplest form, a table places information inside the cells formed by dividing a rectangle into rows and columns. Most cells contain data; some cells, usually on the table's top or side, contain headings. HTML and XHTML represent a basic table using four elements. In markup, a table tag pair, **<table>** … **</table>**, contains an optional **caption** element, followed by one or more rows, **<tr>** … **</tr>**. Each row contains cells holding a heading, **<th>** … **</th>**, or data, **<td>** … **</td>**. However, for most purposes, the following example illustrates a basic table. Note that the only attribute used in this example is **border**, which is used to specify a 1-pixel border so it is clear what the table looks like. The rendering for the simple table under various browsers is shown in Figure 7-1.

```
<!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" lang="en">
<head>
<title>Simple Table Example</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>

<table border="1">
<caption>Basic Fruit Comparison Chart</caption>
```

```
<tr>
  <th>Fruit</th>
  <th>Color</th>
</tr>

<tr>
  <td>Apple</td>
  <td>Red</td>
</tr>

<tr>
  <td>Kiwi</td>
  <td>Green</td>
</tr>

<tr>
  <td>Watermelon</td>
  <td>Pink</td>
</tr>
</table>
</body>
</html>
```

A table is made up of rows enclosed within **<tr>** … **</tr>**. The number of rows in the table is determined by the number of occurrences of the **tr** element. What about columns? Generally, the number of columns in a table is determined by the maximum number of data cells in one row indicated by **<td>** … **</td>**, or headings indicated by **<th>** … **</th>** within the table. The headings for the table are set using the **th** element. Generally, the browser renders the style of headings differently, usually centering the contents of the heading and placing the text in bold style. The actual cells of the table are indicated by the **td** element. Both the **td** and **th** elements can enclose an arbitrary amount of data of just about any type. In the previous example, a full paragraph of text could be enclosed in a table cell along with an image, lists, and links. The table might also have a caption enclosed within **<caption>** …

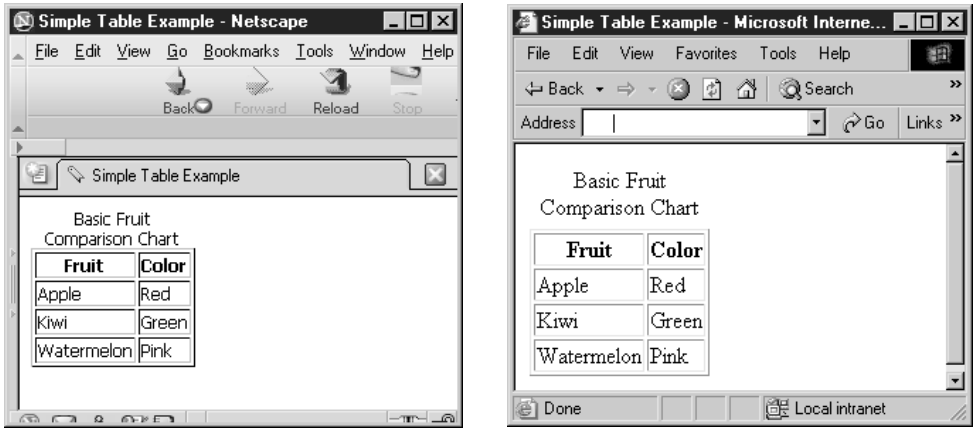


FIGURE 7-1 Browser renderings of a simple example

</caption>, whose contents generally are rendered above or below the table, indicating what the table contains.

NOTE *Under Internet Explorer 4 or better, it is possible to hint to the browser the number of columns used in the table by setting the* ***cols*** *attribute. This is used to improve rendering speed, but is nonstandard.*

Technically speaking, under HTML 4 transitional, the closing tags for the **<tr>**, **<th>**, and **<td>** tags are optional. Although this might make for cleaner-looking code in your HTML documents, HTML writers are still encouraged to use the closing tags, as well as indentation. This ensures that table cells and rows are clearly defined, particularly for nested tables. It also helps to avoid problems with versions of Netscape that often "break" tables that don't use closing tags for these elements. And because XHTML, which requires closing tags for all nonempty elements, is the new standard, you should always close table tags.

The rowspan and colspan Attributes

Whereas the preceding example shows that it is possible to create a table with a simple structure, what about when the table cells need to be larger or smaller? The following markup creates tables that are somewhat more complicated. By adding the **rowspan** and **colspan** attributes to the table elements, it is possible to create data cells that span a given number of rows or columns. The rendering of this code appears in Figure 7-2.


```
<!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" lang="en">
<head>
<title>ROWSPAN and COLSPAN</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>

<table border="1">
  <caption>ROWSPAN Example</caption>
   <tr>
       <td rowspan="2">Element 1</td>
       <td>Element 2</td>
   </tr>
   <tr>
       <td>Element 3</td>
   </tr>
  </table>

<br /><br />

<table border="1">
  <caption>COLSPAN Example</caption>
    <tr>
       <td colspan="3">Element 1</td>
    </tr>
    <tr>
```

```
        <td>Element 2</td>
        <td>Element 3</td>
        <td>Element 4</td>
    </tr>
</table>

</body>
</html>
```

The basic idea of the **rowspan** and **colspan** attributes for **<td>** and **<th>** is to extend the size of the cells across two or more rows or columns, respectively. To set a cell to span three rows, use **<td rowspan="3">**; to set a heading to span two columns, use **<th colspan="2">**. Setting the value of **colspan** or **rowspan** to more than the number of columns or rows in the table should not extend the size of the table. Be aware, however, that some browsers require precise use of these span attributes. Consider the following markup:

```
<table border="1">
<tr>
    <td>Element 1</td>
    <td>Element 2</td>
    <td rowspan="2">Element 3</td>
</tr>

<tr>
    <td>Element 4</td>
    <td>Element 5</td>
    <td>Element 6</td>
</tr>
</table>
```

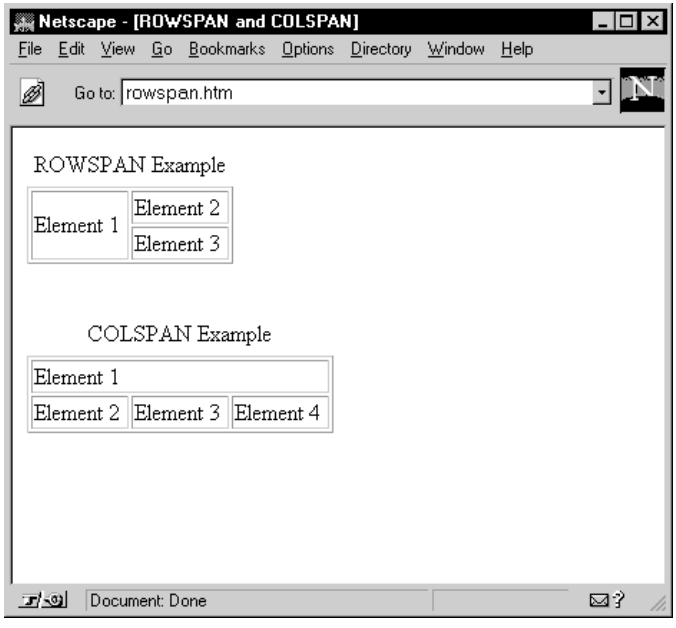
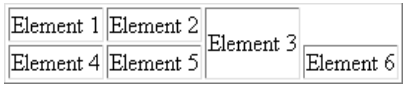


FIGURE 7-2 Rendering of rowspan and colspan

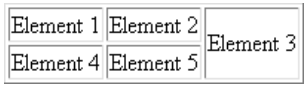
Most browsers will render this code something like this:



The reason is quite simple: The last data cell in the second row should have been removed to account for the rowspan in cell 3 of the first row, like this:

```
<table border="1">
<tr>
    <td>Element 1</td>
    <td>Element 2</td>
    <td rowspan="2">Element 3</td>
</tr>

<tr>
    <td>Element 4</td>
    <td>Element 5</td>
</tr>
</table>
```

The rendering of the improved markup now works properly:



Full Table Example

Aside from being able to span rows and columns, the **table** element, and its enclosed elements **td**, **th**, and **caption**, support a variety of attributes for alignment (**align** and **valign**), sizing (**width**), presentation (**bgcolor** and **background**), and layout (**cellpadding** and **cellspacing**). These attributes will be explained in the next section, but you can probably infer their use by looking at the following example, which shows a more complex kind of table:

```
<!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" lang="en">
<head>
<title>Complex Table Example</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>

<table align="left" border="1" width="300" cellspacing="0">
<caption align="bottom">The Super Widget</caption>

<tr>
  <td rowspan="2">
  <img src="widget.gif" alt="super widget" width="100" height="120" />
```

```
   </td>
  <th bgcolor="lightgreen">Specifications</th>
</tr>

<tr>
<td valign="middle" align="left">
    <ul>
     <li>Diameter: 10 cm</li>
     <li>Composition: Kryptonite</li>
     <li>Color: Green</li>
    </ul>
  </td>
</tr>
</table>

<p>Notice how the text of a paragraph can flow around a table
just as it would any other embedded object form. Notice how
the text of a paragraph can flow around a table just as it
would any other embedded object form. Notice how the text
of a paragraph can flow around a table just as it would
any other embedded object form. Notice how the text
of a paragraph can flow around a table just as it would
any other embedded object form. Notice how the text
of a paragraph can flow around a table just as it would
any other embedded object form. Notice how the text
of a paragraph can flow around a table just as it would
any other embedded object form. Notice how the text
of a paragraph can flow around a table just as it would
any other embedded object form.</p>

</body>
</html>
```

The rendering of the previous example, as shown in Figure 7-3, suggests that it is possible to place any form of content in a cell including even other tables, although this is not recommended if it can be avoided. Furthermore, it appears we have full control over the individual size of the cells and the table itself. With these features, we now have the facilities required to control layout using a **<table>** tag to create a grid on the page.

Tables for Layout

Tables can be a very important tool for HTML-based page layout. The foundation of graphic design is the ability to spatially arrange visual elements in relation to each other. Tables can be used to define a layout grid for just this purpose. Prior to the advent of style sheets supporting positioning (see Chapter 10), tables were the only reliable way to accomplish this. Even with CSS well supported at the time of this edition's writing, tables still remain the most commonly used technique in Web design.

The key to using a table to create a precise page grid is the use of the **width** attribute. The **width** attribute for the **table** element specifies the width of a table in pixels, or as a

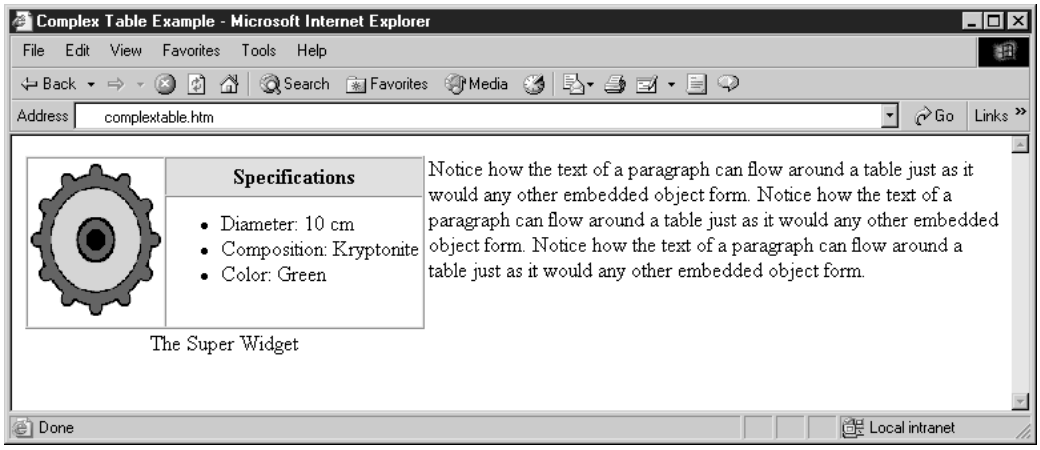
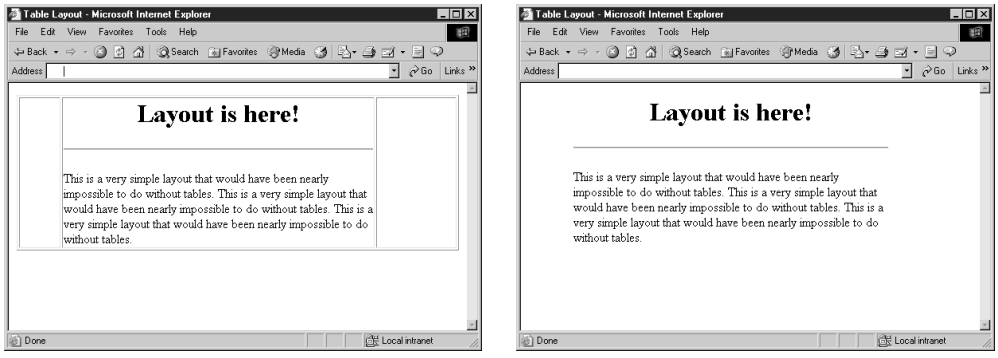


FIGURE 7-3 Advanced table hints at layout possibilities

percentage value such as 80%. It also is possible to set the individual pixel widths of each cell within the table, using a **width** attribute for the **td** or **th** element. Imagine trying to create a 400-pixel column of text down the page with a buffer of 50 pixels on the left and 100 pixels on the right. In HTML before tables, this would be literally impossible without making the text a giant image. With a table, it is easy, as shown by the markup code here:

```
<!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" lang="en">
<head>
<title>Table Layout</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
<table border="0">
<tr>
   <td width="50"> </td>
   <td width="400">
     <h1 align="center">Layout is here!</h1>
     <hr />
     <p>This is a very simple layout that would
      have been nearly impossible to do without tables.
      This is a very simple layout that would
      have been nearly impossible to do without tables.
      This is a very simple layout that would
      have been nearly impossible to do without tables.</p>
   </td>
   <td width="100"> </td>
  </tr>
</table>
</body>
</html>
```

In the preceding code, the **border** value is set to zero. The rendering of the example is shown here, with both the border on and off.



While the border attribute for a **<table>** tag isn't necessary because the browser does not draw a border by default, it is better practice to keep the attribute in, set to zero. The presence of the attribute allows borders to be quickly turned on and off by setting the value to 1 or 0 to check to see what is going on with a particular layout.

TIP *When creating empty table cells, it is a good idea to put a nonbreaking space (** **) or even a clear pixel gif (for example, space.gif) into the cell so it doesn't collapse.*

Besides basic text layout, tables also can be used to provide more precise layout in relation to a background. One popular design concept employs a vertical strip of colored background on the left of the page, which contains navigation controls; the rest of the document contains the main text. Without tables, it is difficult to keep body content off darker regions of a background tile. The following is an example of the markup code to create a two-column design that works on top of a 100-pixel-wide color background:

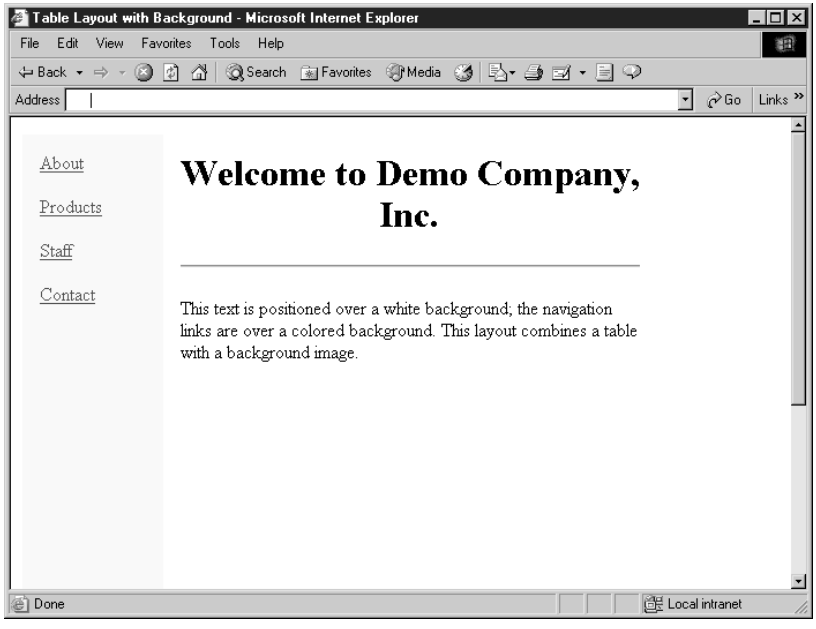
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>Table Layout with Background</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body background="yellowtile.gif">

<table width="550" cellspacing="0" cellpadding="15">
<tr>
    <td width="100" valign="top">
     <a href="about.html">About</a><br /><br />
     <a href="products.html">Products</a><br /><br />
     <a href="staff.html">Staff</a><br /><br />
     <a href="contact.html">Contact</a><br /><br />
    </td>

    <td width="450">
```

```
      <h1 align="center">Welcome to Demo Company, Inc.</h1>
      <hr />
      <p>This text is positioned over a white background;
        the navigation links are over a colored background.
        This layout combines a table with a background image.
      </p>
     </td>
</tr>
</table>
</body>
</html>
```

The rendering of this layout appears in Figure 7-4. Note how the foreground content (the **<body>** content) is aligned over the **background** image. Another way to achieve such effects is to set the **bgcolor** attribute for the table cells and forego the background image altogether.

```
<table width="550" cellspacing="0" cellpadding="10">
 <tr>
    <td width="100" valign="top" bgcolor="yellow">
      … links here …
    </td>
    <td width="450">
       … content here …
    </td>
 </tr>
</table>
```


FIGURE 7-4 Rendering of two-column layout



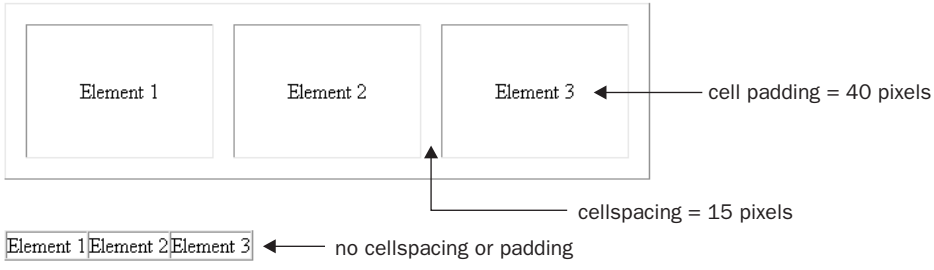
TIP *Often, when setting up table layouts, you will want to control page margins as well. Recall that setting nonstandard* ***<body>*** *attributes like so,* ***<body marginheight="0" marginwidth="0" leftmargin="0" topmargin="0">****, or using the CSS property margin for the* ***body*** *element will help rid your page of unsightly gaps.*

cellpadding and cellspacing

The space between cells in a table is controlled by the **cellspacing** attribute for **<table>**. The value is measured in pixels or percentage values. When using tables for layout, cells should jut up next to each other, so this attribute is often set to **0**, as in previous examples. However, it is possible to give space between cells by setting this attribute to a positive integer or percentage value. Similarly, the padding between cell walls and the content they surround is controlled by the **cellpadding** attribute, which is also often set to **0** in tables used for layout. The two approaches are illustrated by this markup:

```
<table border="1" cellspacing="15" cellpadding="40">
<tr>
   <td>Element 1</td>
   <td>Element 2</td>
   <td>Element 3</td>
</tr>
</table>

<br /><br />

<table border="1" cellspacing="0" cellpadding="0">
<tr>
   <td>Element 1</td>
   <td>Element 2</td>
   <td>Element 3</td>
</tr>
</table>
```

The code above renders the following:



Cell Alignment

Cells defined by **<td>** or **<th>** are generally aligned horizontally by setting the **align** attribute to **left**, **right**, or **center** with **left** being the default. It is also possible to justify their contents by setting the attribute to **justify,** or to align contents on a particular character such as a decimal point using the value **char**. However, aligning on characters is still not that well-supported in browsers. The contents of cells can also be aligned vertically by setting **valign** on **<th>** or **<td>** tags to **top**, **middle**, **bottom** or **baseline**. The following example illustrates the more common uses of these values.

```
<table border="1" cellspacing="0" cellpadding="0" width="100%">
<tr>
   <td align="left">Left</td>
   <td align="center">Center</td>
   <td align="right">Right</td>
</tr>
<tr>
   <td valign="top" height="100">Top</td>
   <td valign="middle">Middle</td>
   <td valign="bottom">Bottom</td>
</tr>
</table>
```

A typical rendering of this markup is shown here.

Left	Center	Right
Top	Middle	Bottom

Colored Tables and Cells

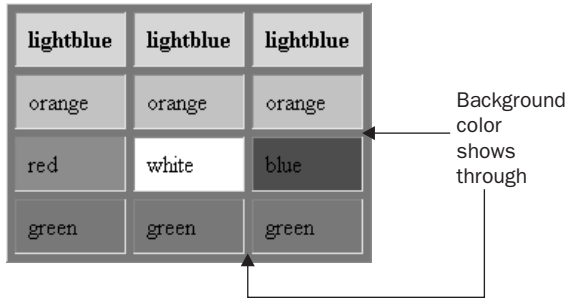
As already mentioned in this chapter, table elements also can be assigned background colors using the **bgcolor** attribute. The **bgcolor** attribute is valid for **<table>**, **<tr>**, **<th>**, and **<td>**.

```
<table border="1" cellspacing="0" cellpadding="8" bgcolor="green">
<tr>
  <th bgcolor="lightblue">lightblue</th>
  <th bgcolor="lightblue">lightblue</th>
  <th bgcolor="lightblue">lightblue</th>
</tr>

<tr bgcolor="orange">
  <td>orange</td>
  <td>orange</td>
  <td>orange</td>
```

```
</tr>

<tr>
  <td bgcolor="red">red</td>
  <td bgcolor="white">white</td>
  <td bgcolor="blue">blue</td>
</tr>

<tr>
  <td>green</td>
  <td>green</td>
  <td>green</td>
</tr>
</table>
```

In this code, the header cells (**th**) in the first row will have a light blue background; all three cells (**td**) in the second row will have an orange background as defined for the entire row (**tr**); the three cells in the third row will have different background colors as defined by the **bgcolor** attribute for each **<td>** tag; and the cells in the last row, which have no background color defined for themselves or their row, will default to the green background color defined in the **<table>** tag.

Recall that the **cellspacing** attribute for **<table>**, which sets how many pixels of space are included between table cells, is set to zero; if it is set to a higher value, the background color will display in the areas between cells in most browsers.



Additional proprietary attributes also have been defined for table border colors. Internet Explorer 4 and higher defines a **bordercolor** attribute for <**table>** as well as for cells. Netscape will recognize the **bordercolor** attribute for **<table>**, but not on the cells. Strict standards-compliant browsers such as Opera 7 will recognize none of these attributes. As an example, the markup

```
<table bordercolor="#ff0000" border="1">
<tr>
  <td bordercolor="#0000ff">Blue Border</td>
  <td>Red Border</td>
</tr>
</table>
```

will render a table with a red border around the entire table and its first cell will have a blue border in Internet Explorer, but not in Netscape.

Internet Explorer 4 and higher also provide two more border color attributes: **bordercolordark** and **bordercolorlight**.

```
<table bordercolorlight="#ff0000" bordercolordark="#0000ff"
       border="4">
<tr>
   <td>Cell</td>
</tr>
</table>
```

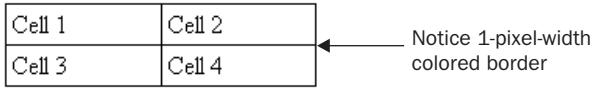
Under Internet Explorer, this example will render a two-tone outer border for the table in which the colors simulate a three-dimensional shading, as shown here:

While these proprietary attributes are useful for creating a bevel style effect on a table, the effect is easily accomplished in standard CSS.

Basic HTML initially didn't provide great support for table borders and colors; with all the proprietary extensions and subtle variations, page authors often ended up frustrated. However, sometimes a simple workaround such as using a nested table can solve the problem, albeit messily. Consider the markup here:

```
<table cellspacing="0" cellpadding="0" border="0" width="200">
<tr>
<td bgcolor="#990000">
<!-- begin nested table -->
<table cellspacing="1" cellpadding="3" border="0" width="200">
<tr>
    <td bgcolor="#FFFFFF" width="100">Cell 1</td>
    <td bgcolor="#FFFFFF" width="100">Cell 2</td>
</tr>

<tr>
    <td bgcolor="#FFFFFF" width="100">Cell 3</td>
    <td bgcolor="#FFFFFF" width="100">Cell 4</td>
</tr>
</table>
<!-- end nested table -->
</td>
</tr>
</table>
```

The outer table employs a single table cell with its **bgcolor** set to red. The cells in the nested table have their **bgcolor** set to white. The **cellspacing** for the nested table is set to 1, allowing the black background of the outer table to show through in the spaces between the cells:



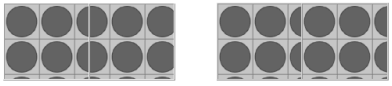
This "hack" will work for browsers even as far back as Netscape 3. Still, some care must be taken in using this approach. The two tables must be the same width, or the border effect could be uneven. Given the reliance on tables in layouts, such tricky markup is much more common than readers might imagine.

Background Images in Tables

Using the **background** attribute it is also possible to apply background images to tables and table cells. Defining a table with the code

```
<table width="100%" border="1" cellpadding="0" cellspacing="0"
       background="tabletile.gif">
```

would place a repeating background tile behind the table, as shown here:



Notice that the table on the left is the typical rendering in modern browsers, but beware that the table on the right shows Netscape 4 tiling backgrounds very differently. It is also possible to set table cells (**<td>** and **<th>**), but the tiling effect will be limited to the cell it is defined on, so be careful if you have adjacent cells with different backgrounds.

Applied Layout Using Tables

Now that we understand the basics of using tables for layout, we approach trying to lay pages out using only HTML/XHTML. Don't worry; creating relatively sophisticated layouts with tables doesn't have to be daunting. A little planning and the right tools can go a long way toward achieving a successful layout.

Centered Layout

Let's start first with a very simple design. In this case, we want to create a centered region with content that looks like a printed page on a background color. In this case, we could use a single cell table and set width, alignment, and padding to pull off the design quite easily, as shown here:

```
<!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" lang="en">
<head>
<title>Centered Table</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body bgcolor="navy">
<table width="80%" align="center" cellpadding="10">
  <tr>
    <td bgcolor="white">
       <h1 align="center">Heading</h1>
```

```
        <hr width="80%" />

        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
         Nulla nulla. Ut vel magna eu velit tristique tempus. Nunc
         a wisi at ligula euismod tempus. Curabitur vestibulum
         viverra tellus. Phasellus vestibulum. Duis justo...</p>

      … more content …

        </td>
   </tr>
</table>
</body>
</html>
```

A full rendering of the example is shown in Figure 7-5.

To expand upon the previous example, we might want to add in navigation and a site label across the top. That could be done easily with two extra rows of cells and appropriate use of the **colspan** attribute. However, it is not always wise to build complex tables when

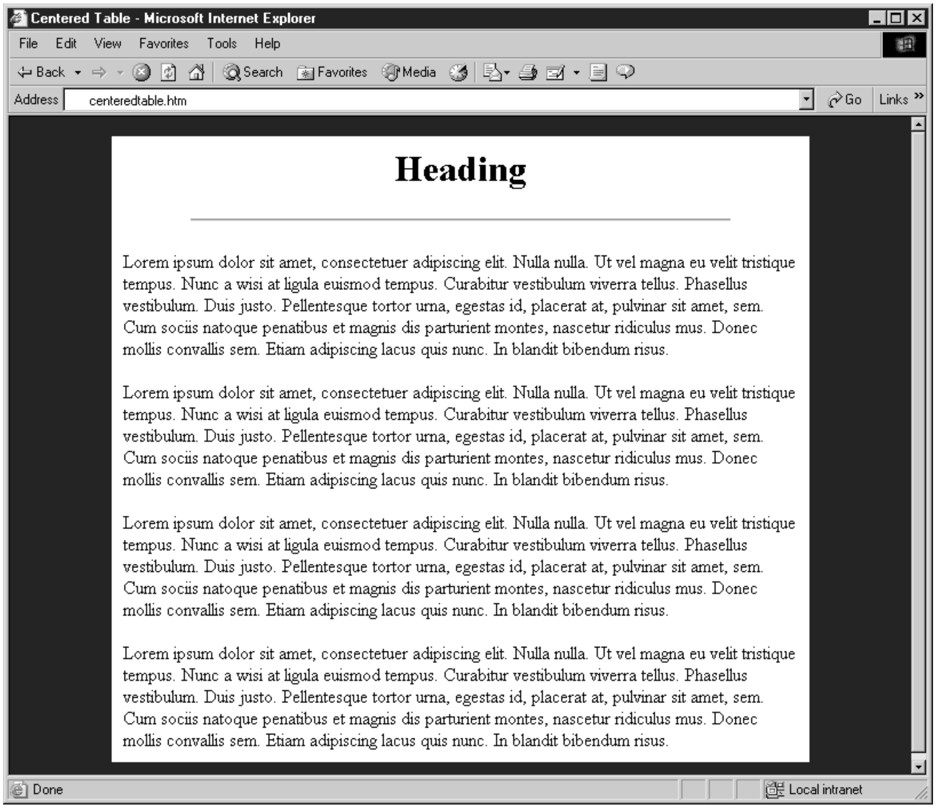


FIGURE 7-5 Centered page using a table

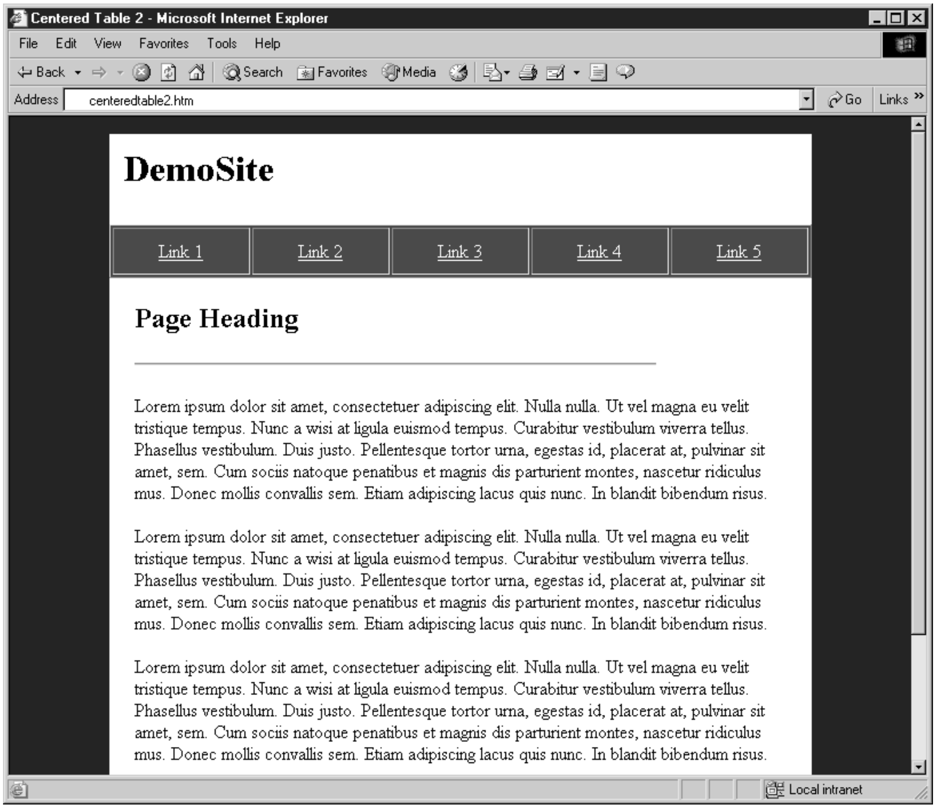


FIGURE 7-6 Centered layout variation

you can simply stack tables, as you can see in the following example, which is presented in Figure 7-6.

```
<!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" lang="en">
<head>
<title>Centered Table 2</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body bgcolor="navy" link="white" alink="white" vlink="white">

<table width="80%" align="center" cellpadding="10" bgcolor="white">
  <tr>
       <td colspan="5"><h1>DemoSite</h1></td>
  </tr>
</table>

<table width="80%" align="center" cellpadding="10" border="1"
```

```
        bgcolor="maroon">
  <tr>
        <td align="center"><a href="#">Link 1</a></td>
        <td align="center"><a href="#">Link 2</a></td>
        <td align="center"><a href="#">Link 3</a></td>
        <td align="center"><a href="#">Link 4</a></td>
        <td align="center"><a href="#">Link 5</a></td>
  </tr>
</table>

<table width="80%" align="center" cellpadding="20" bgcolor="white">
<tr>
   <td colspan="5" bgcolor="white">
        <h2>Page Heading</h2>
        <hr width="80%" align="left" />

        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
           Nulla nulla. Ut vel magna eu velit tristique tempus. Nunc
           a wisi at ligula euismod tempus. Curabitur vestibulum
           viverra tellus. Phasellus vestibulum. Duis justo.</p>

     …more content…

        </td>
  </tr>
</table>
</body>
</html>
```


Top-Left-Bottom "TLB" Layout

A modification of the previous layout would provide for secondary navigation on the left of the content as well as backup navigation or extra information at the bottom of the page. This type of template is very commonly used on the Web and is often called a TLB design for "top-left-bottom." A simple example of a TLB template is shown here with a rendering in Figure 7-7.

```
<!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" lang="en">
<head>
<title>TLB Template</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body bgcolor="#ffffff">

<!--BEGIN: Label or primary nav table -->
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr>
   <td width="100%" bgcolor="yellow">
    <h2 align="center">Site Heading and/or Navigation</h2>
   </td>
</tr>
</table>
```

```
<!--END: Label of primary nav table-->



<!--BEGIN: Secondary nav and content -->
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr>
   <td width="10" bgcolor="red">   </td>
   <td width="90" valign="top" bgcolor="red">
     <br />
     <a href="#">Link</a><br />
     <a href="#">Link</a><br />
     <a href="#">Link</a><br />
     <a href="#">Link</a><br />
     <a href="#">Link</a><br />
     <a href="#">Link</a><br />
   </td>
   <td width="10" bgcolor="white">   </td>
<td>
    <br />
    <h2>Page Heading</h2>
    <hr />

    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
       sed diam nonummy nibh euismod tincidunt ut laoreet dolore
       magna aliquam erat volutpat. Ut wisi enim ad minim veniam,
       quis nostrud exerci tation ullamcorper suscipit lobortis
       nisl ut aliquip ex ea commodo consequat.</p>

…more content…

   </td>

   <td width="10" bgcolor="white">   </td>
</tr>
</table>
<!-- END: secondary nav and content -->



<!--BEGIN: footer navigation and legal-->
<div align="center">
<br />
<font size="-2">
   <a href="#">Link</a> |
   <a href="#">Link</a> |
   <a href="#">Link</a> |
   <a href="#">Link</a> |
   <a href="#">Link</a> |
   <a href="#">Link</a>
</font>
<br /><em>&copy;2003 DemoCompany Inc.</em>
</div>
<!-- END: footer nav -->
</body>
</html>
```

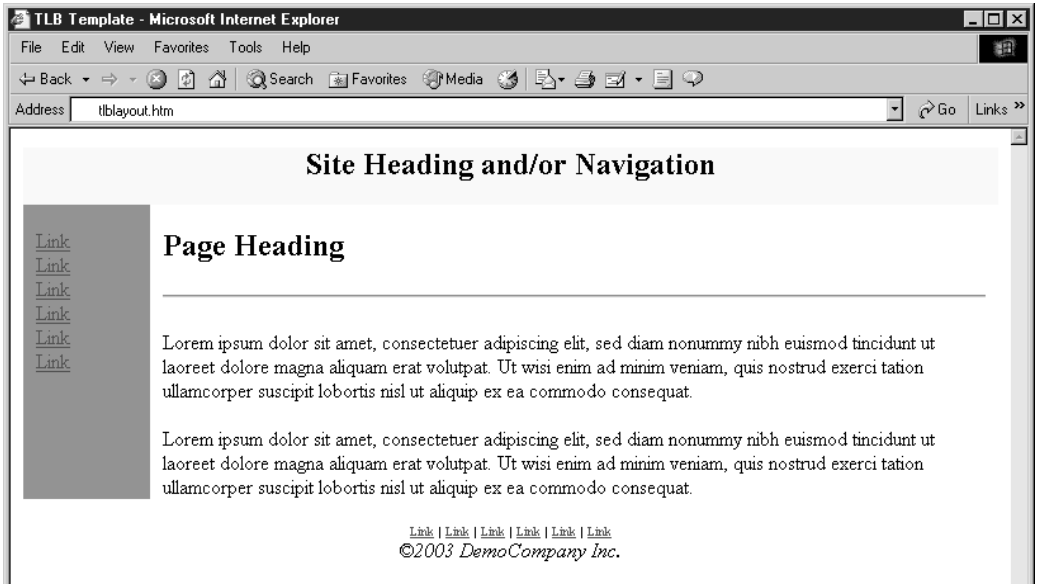


FIGURE 7-7 TLB template example

Stretchable Table Layouts

Another common design is one that stretches to fit the available screen region. The so-called fluid or stretchable design requires that some cells be set with values and others not, to be elastic in response to available screen width. Stretchable designs are particularly common in three-column layouts, as shown in this example:

```
<!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" lang="en">
<head>
<title>Stretch Template</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body bgcolor="#006699">

      <table border="0" width="100%" cellspacing="0" cellpadding="15">

<tr>

   <!-- just a gap -->
   <td width="20" bgcolor="#006699">   </td>

      <!-- navigation column fixed size -->
   <td width="150" bgcolor="#ffcc00" valign="top">
      <h3 align="center">Navigation</h3>
      <a href="#">Link</a><br />
```

```
      <a href="#">Link</a><br />
      <a href="#">Link</a><br />
      <a href="#">Link</a><br />
   </td>

      <!-- just a gap -->
   <td width="20" bgcolor="#ffffff"> </td>

      <!-- content region variable size -->
   <td bgcolor="#ffffff" valign="top">
   <h2 align="center">Stretch Demo</h2>
   <hr />
      <p>Content goes here. Content goes here. Content goes
    here. Content goes here. Content goes here. Content
    goes here. Content goes here. Content goes here.
    Content goes here.</p>
    </td>



   <!--right column fixed size-->
   <td width="100" bgcolor="#ffcc00">Right column text here</td>

   <!--right margin gap-->
   <td width="20" bgcolor="#006699">  </td>
</tr>
</table>
</body>
</html>
```

The rendering of the stretchable design shown in Figure 7-8 demonstrates one downside of the approach. Without enough content, a stretchable table layout might look rather empty on a very large monitor.

TIP *While a stretchable or fluid design does fit to whatever screen the user wants, it can be rather limiting. Creating stretch points limits the design to simple colors or patterns because the relative areas are elastic and would distort an image placed there.*

Complex Table Layouts

It is also possible to apply tables to layouts in a more complicated fashion. Layouts combining text and images can be created using large graphics that incorporate text, but this approach produces pages that are slow to download. The code example that follows shows a more complicated layout that breaks up an image and reassembles it like a jigsaw puzzle, using

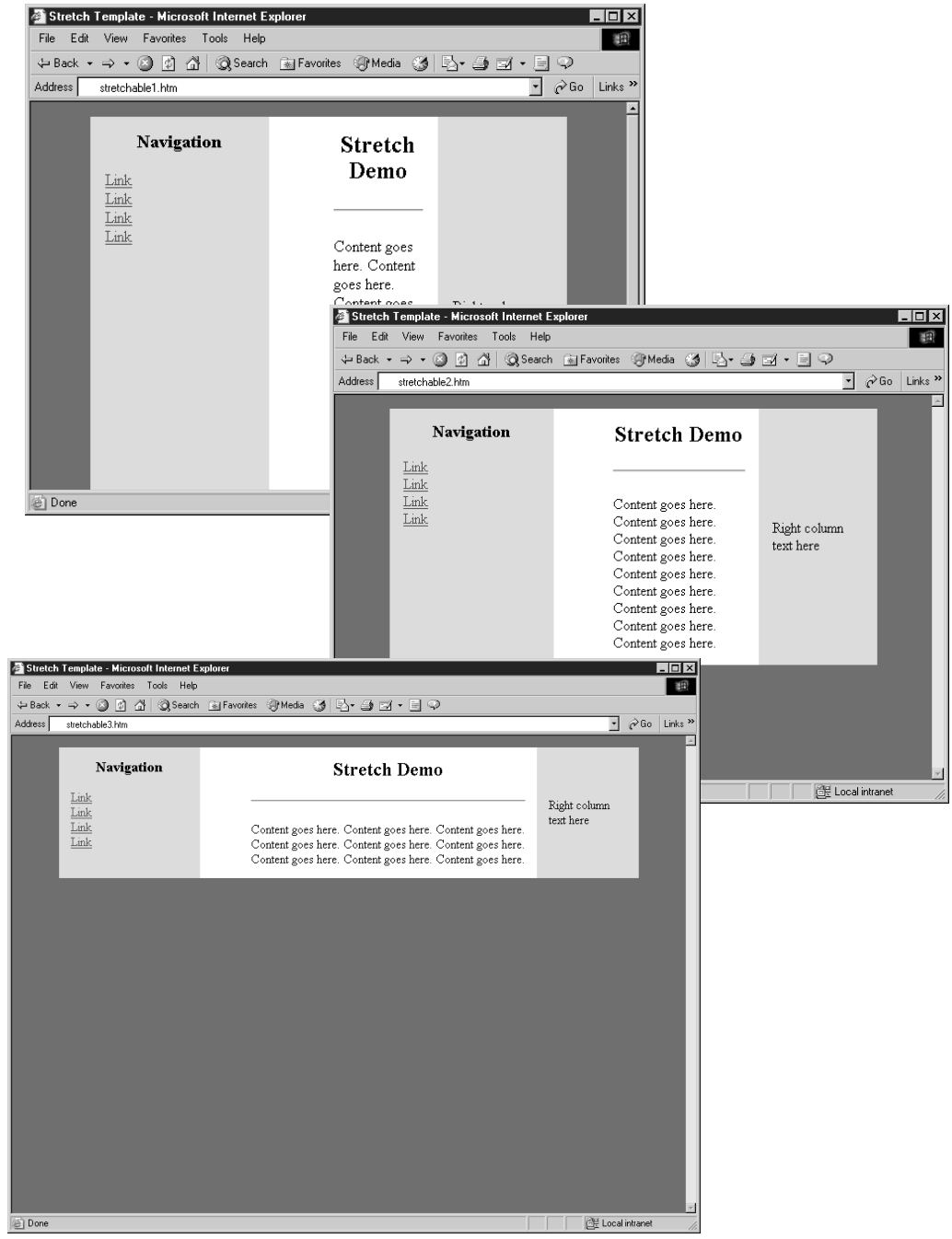


FIGURE 7-8 Stretchable design in action

a table as an invisible "frame" to hold it in place. Note that anchors have not been applied to the graphic links in this code (widgets.gif, and so on) in order to simplify the code example.

```
<!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" lang="en">
<head>
<title>Demo Company, Inc. Home Page</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>

<table border="0" cellpadding="0" cellspacing="0" width="570">
<tr>
<td>
<img src="roof.gif" alt="" border="0" height="45" width="124" />
</td>
<td colspan="4">
<img src="logo.gif" alt="DemoCompany Inc" border="0" height="45" width="446" />
</td>
</tr>

<tr>
<td valign="top" rowspan="7" width="124">
<img src="building.gif" alt="DemoCompany Building" border="0" height="248" width="124" />
</td>

<td rowspan="7" valign="top" width="185">
<img src="headline.gif" alt="We make widgets!" border="0" height="45"
width="185" />
And now, thanks to our merger with Massive Industries, we are now
the world's largest manufacturer of Gadgets&trade; and other
useless products.
<br /><br />
To learn more about our products or our growing monopoly,
click on any of the links to the right.</td>

<td rowspan="3" width="68" valign="top">
<img src="curve.gif" alt="" border="0" height="108" width="68" />
</td>

<td colspan="2" width="193" valign="top">
<img src="blank.gif" alt="" border="0" height="35" width="193" />
</td>
</tr>

<tr>
<td colspan="2" width="193" valign="top">
<img src="widgets.gif" alt="widgets" border="0" height="35" width="193" />
</td>
</tr>

<tr>
<td colspan="2" width="193" valign="top">
<img src="gadgets.gif" alt="gadgets" border="0" height="38" width="193" />
```

```
</td>
</tr>

<tr>
<td colspan="2" rowspan="4" width="136" valign="top">
<img src="gear.gif" alt="" border="0" height="140" width="136" />
</td>
<td valign="top" width="125">
<img src="sales.gif" alt="sales" border="0" height="29" width="125" />
</td>
</tr>

<tr>
<td valign="top" width="125">
<img src="about.gif" alt="about" border="0" height="36" width="125" />
</td>
</tr>

<tr>
<td valign="top" width="125">
<img src="history.gif" alt="history" border="0" height="35" width="125" />
</td>
</tr>

<tr>
<td valign="top" width="125">
<img src="map.gif" alt="map" border="0" height="40" width="125" />
</td>
</tr>

<tr>
<td colspan="2" width="309"> </td>
<td width="68"> </td>
<td width="68"> </td>
<td valign="top" width="125">
<img src="lowcurve.gif" alt="" border="0" height="31" width="125" />
</td>
</tr>
</table>

</body>
</html>
```

When creating a layout like this, it is very important to set the **cellpadding** and **cellspacing** attributes to **0**. Table cell widths should correspond to the width of the image inside the cell, and the width of the table should be the sum of the cells in a table row. It also is important to include the **height** and **width** attributes of the images used. Figure 7-9 shows a browser rendering of this layout, with an overlay to show where the image is broken up.

While the images in the preceding example are all GIFs, JPEGs could also be used. "Photographic" areas of an image should be saved as JPEGs while areas with limited color, such as simple text, should be saved as GIFs. By saving each area in the appropriate format, it is possible to reduce the overall file size and optimize performance. (This is discussed in more detail in Chapter 5.)

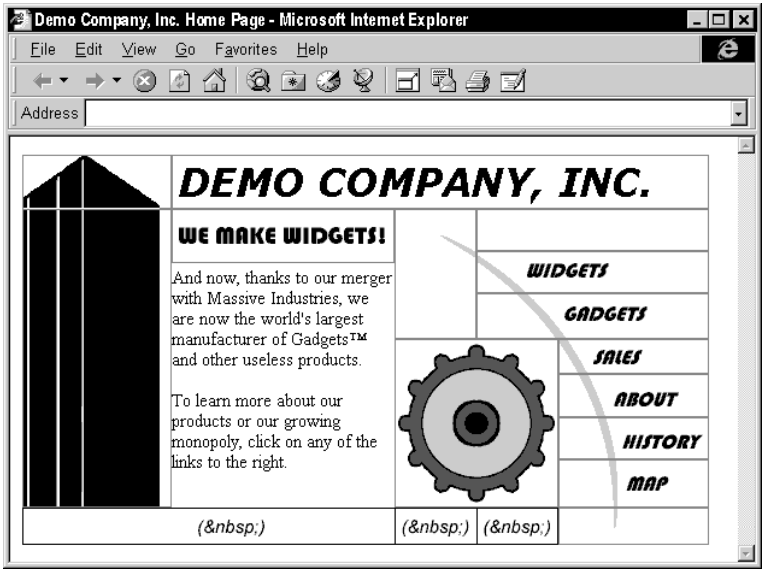
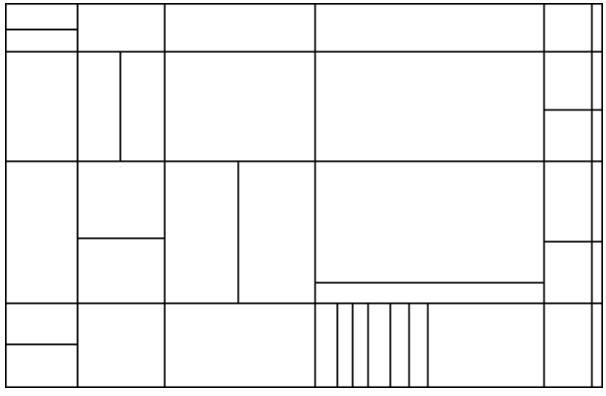


FIGURE 7-9 Rendering of layout with "jigsaw"

Graphic tools such as Adobe PhotoShop or Macromedia's Fireworks enable Web designers to create a single image layout design, beginning with a file such as a large image that is then sliced up to create an HTML/XHTML layout using a table and pieces of the image converted into GIF and/or JPEG files, as appropriate. The use of such a program is shown in Figure 7-10.

Graphic programs such as Fireworks will not only create markup but even special "shim" images to maintain cell widths and heights throughout the table. Although these tools are certainly very convenient in layout preparation, they often produce overly complicated tables, laden with wildly varying **colspan** and **rowspan** attributes, as suggested by the following grid:



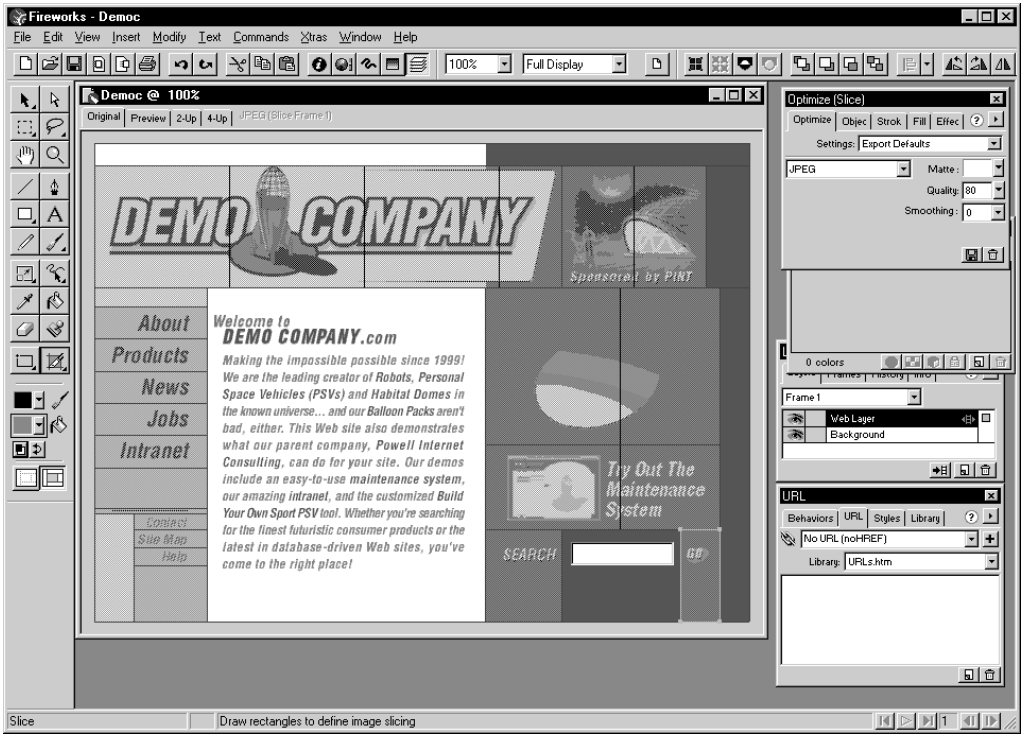
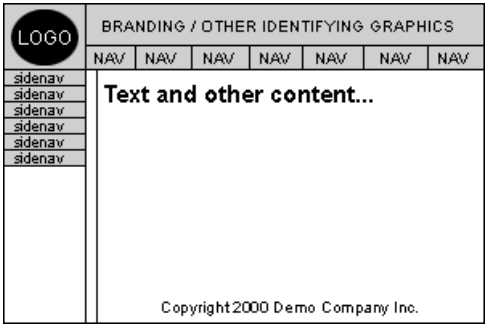
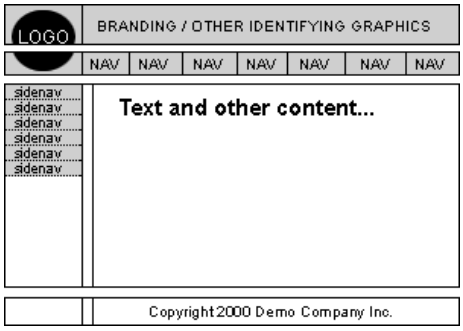


FIGURE 7-10 Slicing visual composite to make a template

Some designers also might generate similar table layouts on their own. Whether using a graphics tool or building such layouts by hand, it is always useful to step back and consider a simpler approach. Consider a basic layout like this:



Although it certainly would be possible to create a single table to hold all of the graphic and text elements of this design, it might be simpler to think of the layout as a layer cake, as shown here:



The row featuring the navigation buttons could be one table, and the part of the layout featuring the side navigation could be another one. Whereas the copyright information at the bottom could be included in a cell with the **colspan** attribute, it too could be split off into its own table, simply to keep different parts of the page separate. As long as the tables are all the same width, each has column widths that add up properly to the overall table width, and do not have **
** tags or any other elements between them, they should stack up perfectly.

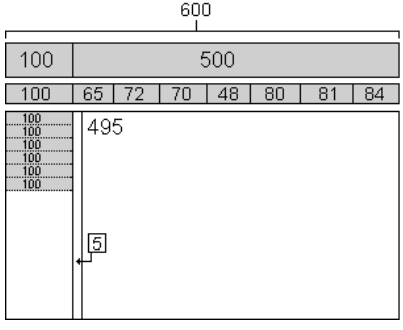


Table Tips

As you saw in the last section, tables can get quite complicated quickly. Here are a few useful rules to make things go smoother when creating layouts with tables.

- Consider using indentation or multiple returns between table rows and cells for easy readability. This is particularly useful if you are hand editing and the extra white space can always be removed before page delivery.
- Always use these three basic attributes with the **table** element: **border**, **cellpadding**, and **cellspacing**. Even if you don't want any of these things, set them to zero; **border** is useful for checking your work, and browsers generally throw in a little bit of unwanted **cellpadding** and **cellspacing** if those attributes are not present.

- Always use the closing tags for every element; this will prevent browser display problems and maintain XHTML compatibility as well.
- Make certain that column widths, as defined by **<th>** or **<td>** cells, add up to the overall defined width of the table. Faulty addition has ruined more than a few seemingly perfect tables.
- Don't forget that use of the **colspan** or **rowspan** attributes in one row will require the removal of some table cells in other rows in the table.
- Certain Microsoft-created attributes, such as **bordercolor**, **bordercolordark**, and **bordercolorlight**, are not suitable for cross-browser usage. Avoid them unless designing for an IE-only environment such as an intranet.
- Try to simplify layouts; don't go crazy with excessive **rowspan** and **colspan** usage when you can stack tables much more simply. It might help to visualize the layout as a layer cake, with a separate table as each layer.
- Don't nest tables deeply. Besides making things overly complicated, you may actually reduce page rendering speed.
- Always comment complex tables so they will make sense to you later. When commenting, at the very least, it is handy to indicate the beginning and end of any table.

```
<!-- begin top nav table -->
   ... table elements ...
<!-- end top nav table -->
```

If you are nesting tables, be certain to mark them as such.

```
<!-- begin nested table 1 -->
   ... table elements ...
<!-- end nested table 1 -->
```

Use comments to any extent you desire, from comments noting the start of new table rows to comments explaining the purpose of a table cell. As long as your comments are meaningful, it won't hurt to add a few kilobytes to your document size in the interest of good coding practice, and you can always strip these comments out before page delivery.

Advanced Data Tables

So far, the discussion of tables has mentioned five elements: **table**, **caption**, **tr**, **th**, and **td**. These are the most commonly used elements. HTML 4 introduced several new elements that provide increased control over table formatting: **col**, **colgroup**, **thead**, **tfoot**, and **tbody**. A variety of attributes to control table data formatting are also supported in standards-aware browsers.

Rather than the simple structure presented earlier, a full HTML/XHTML table is defined by the specification using the following structure:

- An opening **<table>** tag.
- An optional caption specified by **<caption>** … **</caption>**.
- One or more groups of rows. These might consist of a header section specified by **<thead>**, a footer section specified by **<tfoot>**, and a body section specified by

<tbody>. Although all these elements are optional, the table must contain at least a series of rows specified by **<tr>**. The rows themselves must contain at least one header or data cell, specified by **<th>** and **<td>**, respectively.

- One or more groups of columns specified by **<colgroup>** with individual columns within the group indicated by **<col>**.
- A closing **</table>** tag.

The main difference between strict standards-oriented tables and the more basic table form is that rows and columns can be grouped together. The advantage to grouping is that it conveys structural information about the table that might be useful for rendering the table more quickly or keeping it together when displaying on the screen. For example, specifying the **<thead>** or **<tfoot>** might allow a consistent header or footer to be used across larger tables when they span many screens (or sheets of paper when printed). The use of these elements is mandatory when working with dynamically populated tables that incorporate databinding as introduced by Microsoft and discussed later in this chapter.

The example that follows illustrates the use of the relatively uncommon HTML/XHTML table elements just presented.

```
<!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" lang="en">
<head>
<title>Standard HTML/XHTML Tables</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>

<table border="1" frame="box" rules="groups">
<caption>Fun with Food</caption>
<colgroup>
    <col />
</colgroup>

<colgroup>
    <col align="center" />
    <col align="char" char="." charoff="2" />
</colgroup>

<thead>
<tr>
    <th bgcolor="yellow">Fruit</th>
    <th bgcolor="yellow">Color</th>
    <th bgcolor="yellow">Cost per pound</th>
</tr>
</thead>

<tfoot>
<tr>
   <th colspan="3">This has been another fine table example.</th>
</tr>
```

```
</tfoot>

<tbody>
<tr>
    <td>Grapes</td>
    <td>Purple</td>
    <td>$1.45</td>
</tr>

<tr>
    <td>Cherries</td>
    <td>Red</td>
    <td>$1.99</td>
</tr>

<tr>
    <td>Kiwi</td>
    <td>Brown</td>
    <td>$11.50</td>
</tr>
</tbody>

</table>
</body>
</html>
```


The first thing to notice in this code is the use of the **frame** and **rules** attributes for the **<table>** tag. The **frame** attribute specifies which sides of the frame that surrounds the table will be visible. In this example, the value is set to **box**, which means that the frame around the outside of the table is on. Other values for this attribute include **above**, **below**, **hsides**, **vsides**, **lhs**, **rhs**, **void**, and **border**. The meaning of all these values is discussed in the table syntax section of Appendix A.

Do not confuse the idea of the **frame** attribute with that of **rules**. The **rules** attribute defines the rules that might appear between the actual cells in the table. In the example, the value of **rules** is set to **groups**; this displays lines between the row or column groupings of the table. The **rules** attribute also takes a value of **none**, **groups**, **rows**, **cols**, and **all**.

The other major difference in the preceding table example is the inclusion of the **<thead>** and **<tbody>** tags. **<thead>** contains the rows (**<tr>**), headings (**<th>**), and cells (**<td>**) that make up the head of the table. In addition to organization and the application of styles, the advantage of grouping these items is that it might be possible to repeat the elements over multiple pages (under certain browsers). Imagine printing out a large table and having the headers for the rows appear on every page of the printout. This is what **<thead>** might be able to provide. Similarly, the **<tfoot>** tag creates a footer to use in the table, which also might run over multiple pages. Lastly, the **<tbody>** tag indicates the body of the table, which contains the rows and columns that make up the inner part of a table. Whereas there should be only one occurrence of **<thead>** and **<tfoot>**, there can be multiple occurrences of **<tbody>**. Multiple bodies in a document might seem confusing, but these elements are more for grouping purposes than anything else. When a table is specified without **<thead>**, **<tfoot>**, or **<tbody>,** it is assumed to have one body by default.

Notice that one of the **<col>** tags in the example uses the **char** value for **align** in conjunction with the **char** attribute:

```
<col align="char" char="." />
```

This is meant to make the contents of the cells in that column line up with a certain character; in this case, a decimal point. The intended effect would be useful for aligning numbers with decimal points:

Fun with Food

Fruit	Color	Cost per pound
Grapes	Purple	$1.45
Cherries	Red	$1.99
Kiwi	Brown	$11.50
This has been another fine table example.		

Unfortunately, this doesn't seem to work in just any browser:

Fun with Food

Fruit	Color	Cost per pound
Grapes	Purple	$1.45
Cherries	Red	$1.99
Kiwi	Brown	$11.50
This has been another fine table example.		

Although data tables are becoming more difficult to code, you can take heart from the variety of tools that can be used to create them. Most HTML editing tools can easily add the elements needed to make tables; Macromedia Dreamweaver and Homesite offer tools for table creation. This is good, because the combination of standard table elements along with various proprietary extensions introduced by Microsoft results in a dizzying array of elements and attributes for the individual table cells. For those inclined to delve into all the details, the complete syntax for the **table** element and all associated elements can be found in Appendix A.

Databinding: Tables Generated from a Data Source

Tables often contain row after row of identically formatted data that originates in a database. There are two basic methods to create these data-dependent tables. Neither one is ideal:

- If the table data is relatively static, it is common to build a long table by hand or with a tool, individually coding each data cell.
- If the table data is dynamic, it is common to generate the entire page containing the table using a server-side technology such as CGI, ASP, ColdFusion, or PHP as discussed in Chapter 13.

The first approach is difficult for an HTML author. The second, which does not really qualify as HTML authoring, usually requires programming and server access. *Databinding*, while a proprietary client-side browser technology introduced by Microsoft, can be much simpler to use. The basic idea is to dynamically bind HTML elements to data coming from an external source such as a text file, XML file, or database. Although not technically restricted to HTML tables, it does represent a simpler, more powerful approach for generating large data-dependent tables.

In HTML databinding, a data source that provides information is associated with a data consumer that presents it. The data source is a control with some means to access external information that is embedded in an HTML document using the **<object>** tag. This tag is briefly introduced in Chapter 9, and is further explained in Chapter 15. For now, it will be useful to understand that **<object>** adds a small program to the page that can be used to access an external data source. The document also contains a data consumer, an HTML element that uses special attributes to ask the ActiveX control for data that the element subsequently displays. Data consumers come in two sorts: those that present single data values, and those that present tabular data. Tables obviously fall into the latter category.

Creating an HTML table using databinding is a very simple process. It is necessary to define only one table row. The rest are generated automatically according to the template defined by the first row. Think of each row in a tabular data set as corresponding to a database record, and each column as corresponding to a database field. A template table row is defined in HTML that associates **<td>** or **<th>** tags with field names in the data set. A table will subsequently be generated with one row for each record in the data set, and with cell values filled in from the appropriate record fields. The data source control might support processing capabilities such as sorting or filtering the data set. If so, the table can be dynamically regenerated on the client side in response to updated information from the data source. For example, a data source might contain a tabular data set for product price information. One field might contain the name of the product and another its price. By default, a table could present this information sorted alphabetically by product name. In response to a button on an HTML page, the data source could sort the data set by price. The table that displays the information would be dynamically regenerated.

To better understand the idea of databinding, consider the following simple example. An external data file contains two or more columns of comma-delimited data. The first line contains the names of the data set fields corresponding to the columns. The following lines contain the actual data for the appropriate fields. The sample external data file called alphabet.txt is shown here:

```
Letter, Thing
A, Apple
B, Boy
C, Cat
D, Dog
E, Elephant
F, Fox
G, Girl
H, Hat
```

To access the data, an HTML document references an object for a data source control and a related table definition. The following is an example of how this would be accomplished:

```
<!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" lang="en">
<head>
<title>Data Binding Example</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
<!--   validation not possible due to datasrc and datfld attributes -->
</head>
<body>
<object id="alphabet"
        classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <param name="DataURL" value="alphabet.txt" />
    <param name="UseHeader" value="True" />
</object>

<table datasrc="#alphabet" border="1">
<thead>
    <tr bgcolor="yellow">
        <th>Letter</th>
        <th>Reminder</th>
    </tr>
</thead>
<tbody>
    <tr align="center">
        <td><span datafld="Letter"></span></td>
        <td><span datafld="Thing"></span></td>
    </tr>
</tbody>
</table>
</body>
</html>
```

This HTML code generates a table from the file alphabet.txt in which each table row contains a letter of the alphabet and the name of a thing that can remind the reader of that letter. The rendering of this example under Internet Explorer is shown in Figure 7-11.

Let us examine a little more closely the pieces needed to make this databinding example work. First, the data source uses the Tabular Data Control (TDC) object: an ActiveX control provided by Microsoft and identified by the lengthy class identifier. This particular control locates and manipulates text data files in a tabular format. Other controls supporting databinding could have been used instead. These can support different data access capabilities such as access to remote relational databases. The Microsoft ActiveX Data Objects control (ADO), however, is a representative example. The TDC supports several parameters, two of which are used in this example. The **"DataURL"** parameter tells the TDC the name and location of the data file it is to use. In this case, because only a filename is provided, the TDC looks in the same directory containing the Web page. By default, the TDC treats every line in a data file as data. The **"UseHeader"** parameter tells the TDC that the first line in the data file does not contain data but rather the names of data fields.

As a data consumer, the **table** element uses its **datasrc** attribute to connect to a data source. Note in the example how this attribute is set to the name of the **<object>** tag invoking the data source control. Like anything referenced by an **id**, the object name must be preceded

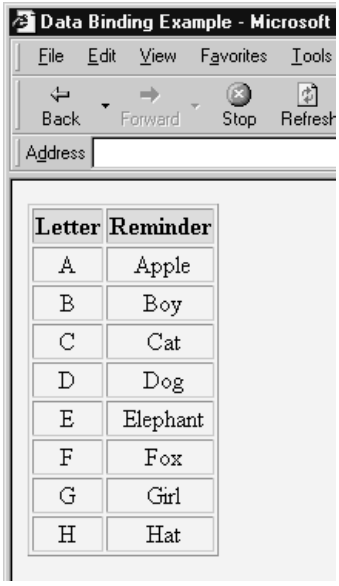


FIGURE 7-11 Databinding example under Internet Explorer

by the **#** symbol and the **<object>** tag must declare a name using the **id** attribute in order to be accessed by a data consumer. In summary, the **datasrc** attribute identifies a data source to be used in generating a table.

The next step is to associate cells in the template table row with particular fields in the data set. This is done using the **datafld** attribute of appropriate elements. It contains the name of the field in the data set that its element is to be bound to. If data set–specific names are not defined, fields can be identified using default positional names: "Column1", "Column2", and so forth. The **<td>** tag, commonly used for cell data, does not support the **datafld** attribute. To bind a field to a table cell, the **<td>** tag needs to contain one of the elements that supports **datafld**. The tags that make the most sense in the context of a table are ****, ** **, **<object>**, and **<img>**. The latter two tags illustrate that databinding is not confined to textual data. For example, a column of images can be created by using a tag declaration such as **<img datafld="Picture">** inside a table cell. Note that the usual **src** attribute would not be required. Instead, the **datafld** attribute identifies a field inside the data set that contains a valid image filename, such as mypict.gif, and binds the image to that value.

Microsoft provides one additional attribute, **datapagesize**, which can be used to limit the number of records displayed from the datasource document. For example, if the **<table>** tag in the preceding example were revised to read

```
<table datasrc="#alphabet" border="1" datapagesize="3">
```

the rendering of the table would display only the first three rows (A, B, and C) of the information in alphabet.txt.

If a table does not explicitly declare header or footer section elements, then implicitly all table content is in the body section. In static tables, this usually does not have visual consequences, but it does in tables generated by databinding. All body rows are included in the template for table row generation, not just the rows containing databound fields. To prevent header or footer information from being repeated for every row in the table, it is necessary to enclose it with the **<thead>** or **<tfoot>** tag. The **<tbody>** tag then can be used to signal the beginning of the template to be databound.

Such a brief example scratches the surface of databinding and merely shows the importance of tables in relation to dynamic data. For more information on databinding, visit the Microsoft MSDN site at http://msdn.microsoft.com and the Remote Data Service site at http://www.microsoft.com/data/ado/rds/.

Summary

The **table** element and its associated elements have become the most commonly used means of creating Web page layouts using HTML/XHTML markup. Although positioning through style sheets (see Chapter 10) should provide more precise layout capacities, browser support is still somewhat inconsistent, there is often an issue of backward compatibility, and developers and Web design tools still do not embrace CSS fully. For better or worse, for many developers tables are still the best way to create layouts that work across multiple browsers, especially when considering older browsers. Even if tables lose their layout duties, they are very useful; remember, they are meant to present tabular data.