Chapter 3

Advanced XHTML

HTML is the language for authoring Web pages. Chapter 2 introduced HTML/XHTML and covered many basics topics. With that foundation, we are ready to explore more advanced features of HTML.

The Web is international. Web pages may be written in the *Universal Character Set* (the UNICODE characters) that covers most known languages in use. HTML also provides ways to enter these characters from ASCII keyboards.

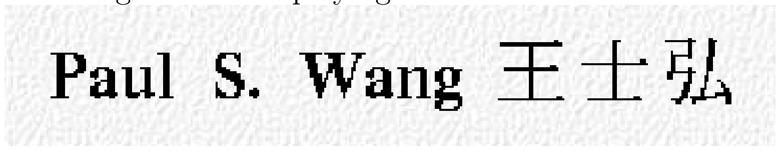
A major HTML topic is tables. HTML tables are useful for presenting information and for creating page layouts. The many aspects of table formatting and style control require careful presentation and patient experimentation to learn and master. Many practical example are provided to help you along.

Frames is a controversial construct. The pros and cons are of frames are discussed at length so you can pick the right situations to use them.

Various elements, such as `meta` and `base`, given exclusively inside the `head` element provide critical functions for Web pages. You'll get to know how to use them.

The chapter concludes by showing you how to avoid common Web page errors and how to check and validate pages.

Figure 3.1: Displaying Character References



3.1 Character Encoding

The ASCII character set contains only 128 characters. HTML uses the much more complete *Universal Character Set* (UCS) or UNICODE[1] character set, defined in ISO10646. UCS contains characters from most known languages. Characters in UCS are put into a linear sequence and each character has a *code position*. The ASCII characters are given the code positions 0-127.

An HTML document containing UCS characters can be encoded in different ways when stored as a file or transmitted over the Internet. UTF-8 is a byte oriented *Unicode Transformation Format* that is popular because of its ASCII preserving quality. UTF-8:

- represents ASCII characters (code positions 0-127) with the lower 7-bits of one byte (single-byte code)
- represents code positions 128-2047 with two bytes
- represents code positions 2048-65536 with three bytes

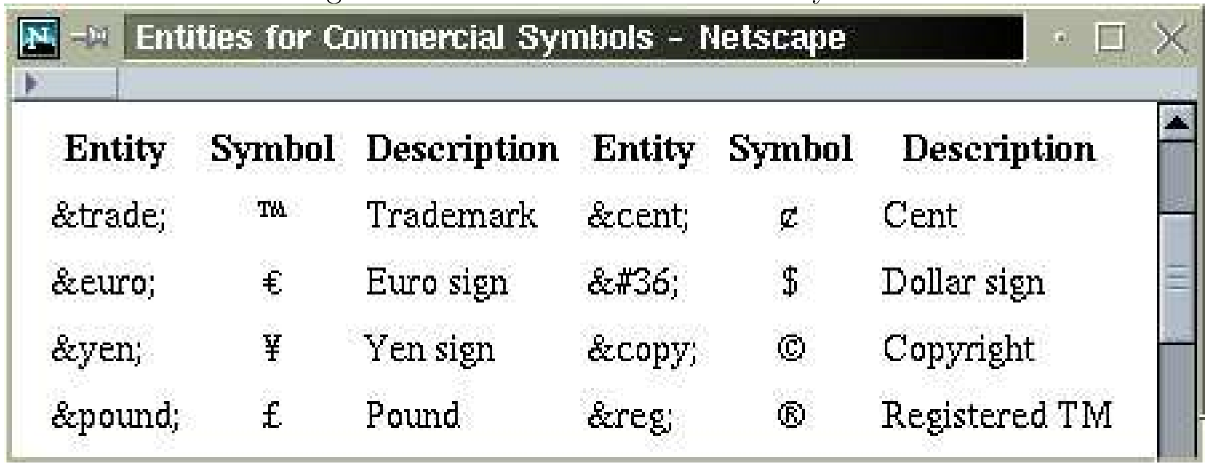
and so on up to 6 bytes to cover all 2 sup 31 Unicode characters. UTF-8 sets the most significant bit of every byte to 1 for multi-byte codes to distinguish them from single-byte codes. Other bits of the leading byte are used as a byte count. In contrast, the UTF-16 encoding uses two bytes to represent each Unicode character.
An XHTML document must specify its character encoding via a line in the form

```
<?xml version="1.0" encoding="UTF-8" ?>
```

which usually comes first in a document file.

[1] `www.unicode.org`

Figure 3.2: Entities for Commercial Symbols

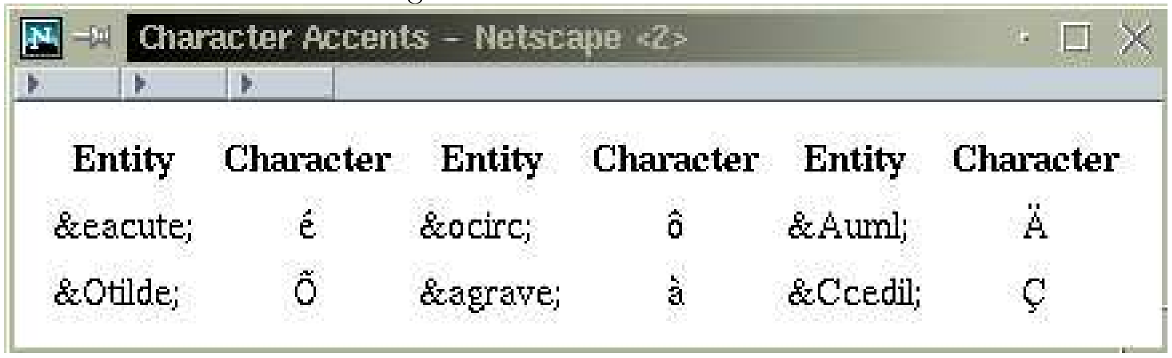


Entity	Symbol	Description	Entity	Symbol	Description
&trade;	™	Trademark	&cent;	¢	Cent
&euro;	€	Euro sign	&#36;	$	Dollar sign
&yen;	¥	Yen sign	&copy;	©	Copyright
&pound;	£	Pound	&reg;	®	Registered TM

3.2 Special Symbols and HTML Entities

On ASCII-oriented computers, it may not be easy to enter into a document characters not directly available on the keyboard. HTML provides a character-set independent way to enter Unicode characters.

Figure 3.3: Character Accents



Entity	Character	Entity	Character	Entity	Character
&eacute;	é	&ocirc;	ô	&Auml;	Ä
&Otilde;	Õ	&agrave;	à	&Ccedil;	Ç

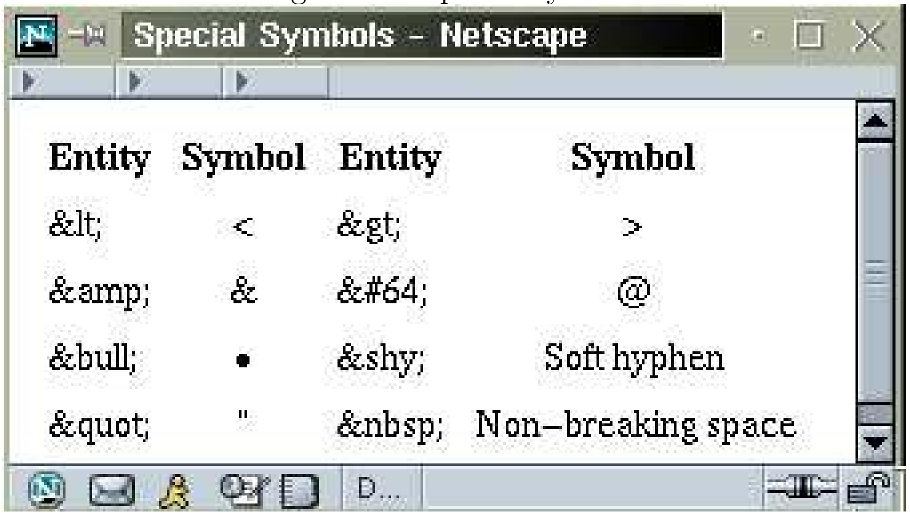
A *numeric character reference* specifies the code position with the notation

`&#`*decimal*`;` or `&#x`*hex*`;`

where *decimal* is a decimal integer and *hex* is a hexadecimal integer. A correctly configured browser seamlessly displays entities together with ASCII characters. The Chinese characters displayed in Figure 3.1 are `&#x738B; &#x58eb; &#x5f18;`, the Chinese name of Paul Wang (Ex: **Chinese**).

To make characters easier to use, HTML *character entities* provide mnemonic names for them. An *entity* is a short sequence of characters beginning with an AMPERSAND (&) and ending with a SEMICOLON (;). Figure 3.2 shows the browser presentation of some entities for often-used commercial symbols. See the Ex: **Symbols** example for the source code.

Figure 3.4: Special Symbols



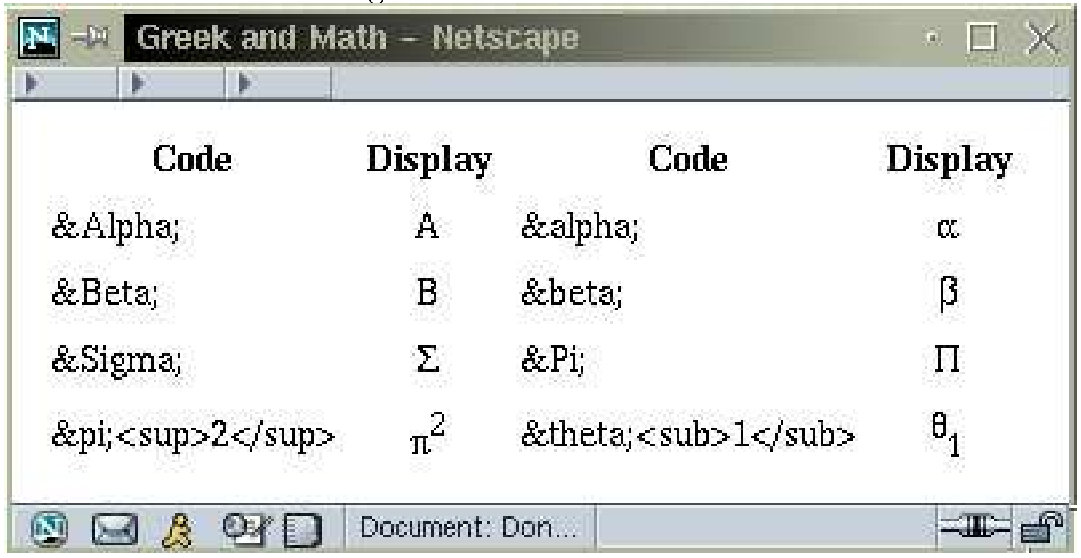
Entity	Symbol	Entity	Symbol
&lt;	<	&gt;	>
&amp;	&	&#64;	@
&bull;	•	&shy;	Soft hyphen
&quot;	"	&nbsp;	Non–breaking space

HTML entities form a subset of UCS and include Latin-1 characters, mathematical symbols, Greek letters, characters with accents, and other special characters. Figure 3.3 shows how character accents are specified (Ex: **Accents**). Other special symbols (Ex: **S**pecial) useful in HTML codes are shown in Figure 3.4.

Greek characters are also easy to specify. They are frequently used in mathematical notations. Figure 3.5 shows how to form entities for Greek characters and how to add superscripts or subscripts to them (Ex: **Greek**). Good tables for HTML entities are available on the Web and links can be found on the WDP site.

MathML is a markup language specifically designed to put mathematical formulas on the Web. More information on MathML can be found at the W3C site.

Figure 3.5: Greek Characters



Code	Display	Code	Display
&Alpha;	A	&alpha;	α
&Beta;	B	&beta;	β
&Sigma;	Σ	&Pi;	Π
&pi;²	π^2	&theta;₁	θ_1

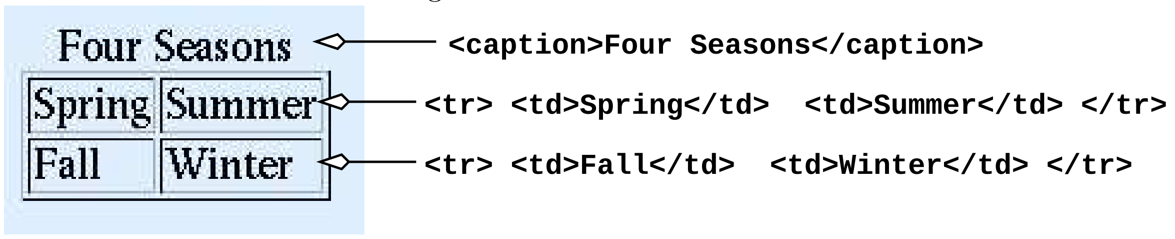
3.3 Tables

Tables let us display information in a clear and concise fashion. The block element `table` organizes and presents information in neatly aligned rows and columns. It is often used to

- present tabular data
- layout Web pages by aligning content and graphics in horizontal and vertical grids (Section 5.12) to achieve visual communication designs
- organize entries in fill-out forms for user input

In general, a table involves many aspects: rows, columns, data cells, headings, lines separating cells, a caption, spacing within and among cells, and vertical and horizontal alignments of cell contents. The rows and columns can be separated into groupings. A cell can also span several columns or rows. Thus, `table` is a complicated HTML construct and it takes some doing to understand and master. But, the ability to use `table` effectively is critical for HTML programming.

Figure 3.6: Table Structure


Four Seasons
Spring
Summer
Fall
Winter
<caption>Four Seasons</caption>
<tr> <td>Spring</td> <td>Summer</td> </tr>
<tr> <td>Fall</td> <td>Winter</td> </tr>

HTML Table Basics

In its most basic form, a `table` simply has a number of rows, each containing the same number of table entries (or cells). The following a simple table (Ex: **SimpleTable**) with two rows, two columns, and a caption:

```
<table border="1">                                (A)
<caption>Four Seasons</caption>
  <tr> <td>Spring</td> <td>Summer</td> </tr>    (B)
  <tr> <td>Fall</td>   <td>Winter</td> </tr>    (C)
</table>
```

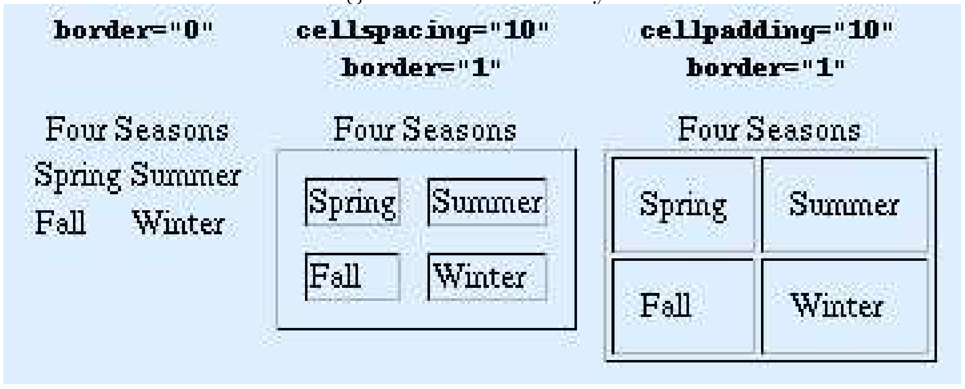
Figure 3.6 shows the displayed table along with HTML codes for its parts (Ex: **Tablestructure**).

The `tr` element gives you a table row. We have two rows in this example (lines `B` and `C`). You may only place table cells inside a table row.

A table cell is either a `td` or a `th` element. Use `th` for a cell that contains a header for a column or a row. Usually, a header will automatically appear in boldface. A `td` (`th`) may contain any inline and block elements, including `table`. Hence you can put table(s) into a table.

The table caption (`caption`) is optional. Attributes and style rules for table elements control the visual presentation of tables. In this example, the `border="1"` attribute in the `table` element (line `A`) gives the width of the table border and the rules between the cells. With `border="0"` (the default) the same table would be without border or separating lines as shown by the first table in Figure 3.7.

Figure 3.7: Table Styles



To give more spacing between table cells set the `table cellspacing` attribute. For example, the code

```
<table border="1" cellspacing="10">
```

gives the second table in Figure 3.7. To give cell contents more breathing room inside cells, set the `cellpadding` attribute. For example, the code

```
<table border="1" cellpadding="10">
```

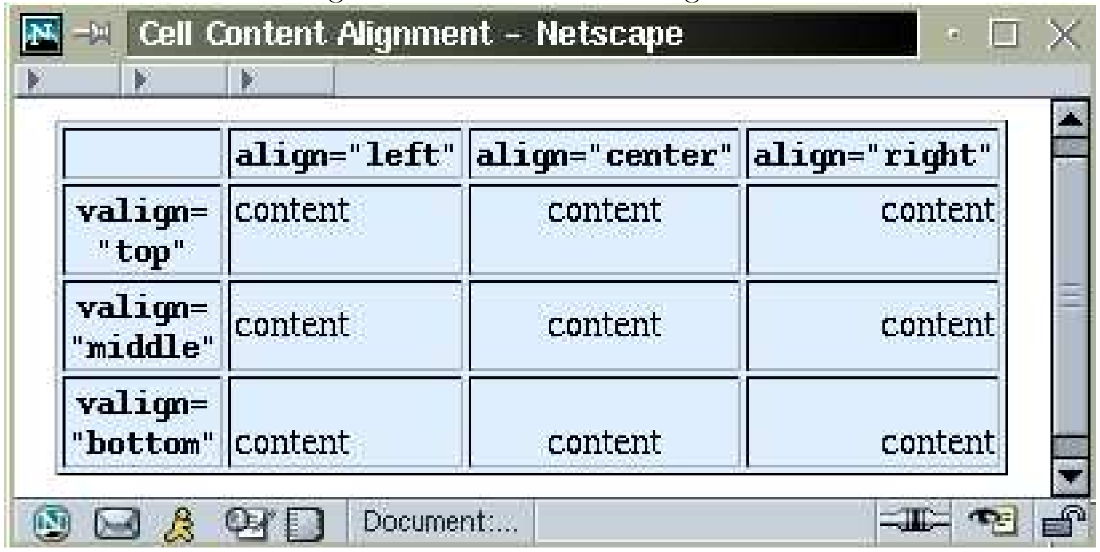
gives the third table in Figure 3.7. Notice that the contents are left aligned in the three tables of Figure 3.7. Let's turn our attention to cell content alignment.

3.4 Cell Content Alignment

By default, content in a cell is `left` aligned horizontally and `middle` aligned vertically inside the space for the cell.

The `align` (horizontal alignment) and `valign` (vertical alignment) attributes of `tr` specify content alignment for all cells in a row. Possible values for `align` are: `left`, `right`, `center`, `justify` (left and right justify lines in a cell) and `char`. The `align="char"` attribute can, for example, line up the decimal points for a column of numbers. But this feature is poorly supported by browsers.

Figure 3.8: Cell Content Alignments



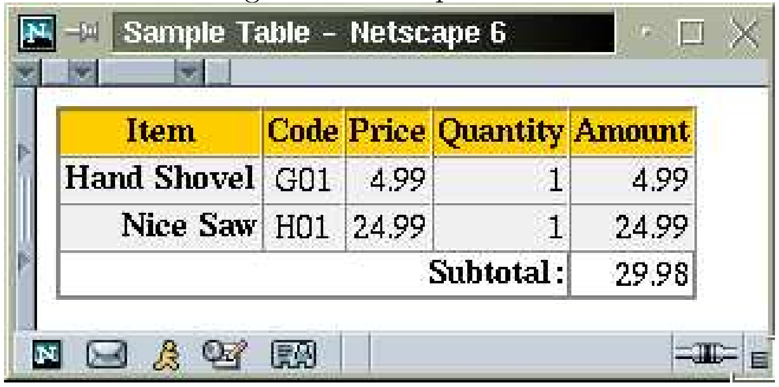
	align="left"	align="center"	align="right"
valign= "top"	content	content	content
valign= "middle"	content	content	content
valign= "bottom"	content	content	content

Possible values for `valign` are: `top` (top of cell) `middle` (middle of cell), `bottom` (bottom of cell), `baseline` (align baseline of first line of each cell in a table row).

The `align` and `valign` attributes can be placed in a `td` (defaults: `left` and `middle`) or `th` (defaults: `center` and `middle`) to control the content alignment for a single cell. Let's use an HTML table (4 rows and 4 columns) to demonstrate cell content alignments. The source code for the table shown in Figure 3.8 is as follows.

```
<table style="background-color: #def" border="1">
<tr><th></th>                                                (D)
    <th><code>align="left"</code></th>
    <th><code>align="center"</code></th>
    <th><code>align="right"</code></th> </tr>
<tr><th><code>valign=<br />"top"</code></th>                  (E)
    <td align="left" valign="top">content</td>
    <td align="center" valign="top">content</td>
    <td align="right" valign="top">content</td> </tr>
<tr><th><code>valign=<br />"middle"</code></th>               (F)
    <td align="left">content</td>
    <td align="center">content</td>
    <td align="right">content</td> </tr>
<tr><th><code>valign=<br />"bottom"</code></th>               (G)
    <td align="left" valign="bottom">content</td>
    <td align="center" valign="bottom">content</td>
```

Figure 3.9: Sample Table



Item	Code	Price	Quantity	Amount
Hand Shovel	G01	4.99	1	4.99
Nice Saw	H01	24.99	1	24.99
			Subtotal:	29.98

```
    <td align="right" valign="bottom">content</td> </tr>
</table>
```

The first row supplies headers (line `D`) indicating the horizontal alignment for cells in each column. The second row (line `E`) uses `valign=top`, the third row (line `F`) uses the default `valign=middle`, and the fourth row (line `G`) uses `valign=bottom`.

3.5 Displaying Tables

Because tables are inherently presentational, it is important to learn how to use element attributes and style properties to achieve the desired display in practice.

Let's first look at a realistic example (Ex: **Cart**) where a shopping cart is displayed as a table of purchased items (Figure 3.9):

```
<table cellspacing="0" cellpadding="1" border="1">  (1)
<thead>                                             (2)
 <tr align="center" style="background-color:#fc0">  (3)
   <th>Item</th> <th>Code</th> <th>Price</th>
   <th>Quantity</th> <th>Amount</th>
 </tr>
</thead>                                            (4)
<tbody>                                             (5)
 <tr valign="middle" align="right"
     style="background-color:#f0f0f0">              (6)
   <th> Hand Shovel</th>
   <td align="center">G01</td> <td>4.99</td>        (7)
   <td>1</td> <td>4.99</td>
 </tr>
 <tr valign="middle" align="right"                  (8)
     style="background-color:#f0f0f0">
   <th>Nice Saw</th>
   <td align="center">H01</td> <td>24.99</td>
   <td>1</td> <td>24.99</td>
 </tr>
 <tr>
    <th colspan="4" align="right">Subtotal:</th>    (9)
    <td align="right">29.98</td>
 </tr>
</tbody>                                            (10)
</table>
```

This example gives a table more structure by putting the header row in a `thead` element (lines `2-4`) and the rest of the rows in a `tbody` element (lines `5-10`). It is possible for a table to have several `thead` and `tbody` elements each containing one or more rows (Section 3.9).

To control the presentation of a table you use `table` attributes for the whole table, `tr` attributes for a whole row, and `td` or `th` attributes for a single table cell.

The next sections continue to explain this example.

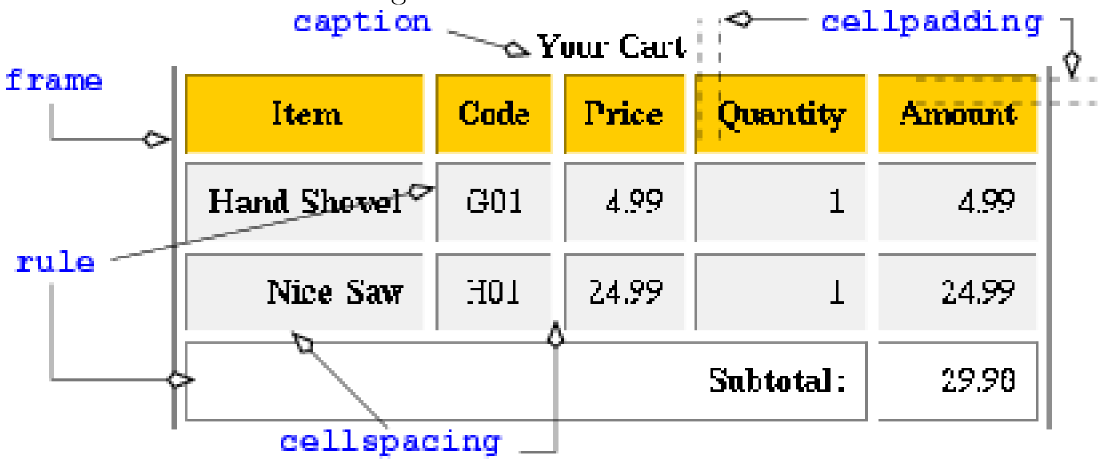
3.6 Table Formatting

Attributes for the `table` element control various formatting options for the table.

- `border`—specifies the width in pixels of the border around a table (line `1`). A non-zero `border` setting also implies horizontal and vertical rules, and a zero value also implies no rules.

- `rules`—specifies the thin 1-pixel rules separating table cells: `none` (no rules), `groups` (rules between row groups and column groups only), `rows` (rules between rows only), `cols` (rules between columns only), and `all` (rules between all cells). The default

Figure 3.10: Table Features



Your Cart

Item	Code	Price	Quantity	Amount
Hand Shovel	G01	4.99	1	4.99
Nice Saw	H01	24.99	1	24.99
			Subtotal:	29.98

is `none` if `border` is not set (set to 0) and the default is `all` if `border` is set. This attribute is poorly supported by browsers.

- `frame`—specifies the visible sides of the table's outer border: `void` (no side), `above`/`below` (top/bottom), `vsides`/`hsides` (left+right or top+bottom), `lhs`/`rhs` (left or right), `border` (all sides).
- `cellspacing`—specifies the amount of space between table cells (line `1`). The default `cellpacing` is usually 1 pixel.
- `cellpadding`—defines the amount of space between the cell border and cell contents (line `1`). The default `cellpadding` is usually 1 pixel.
- `summary`—text describing the purpose and content of the table for non-visual browsers.

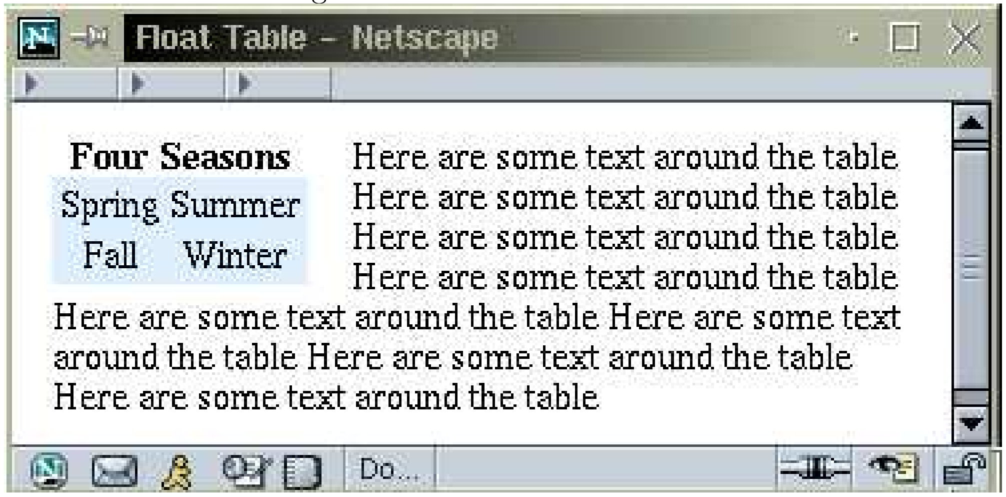
Figure 3.10 shows a table declared as

```
<table cellspacing="4" cellpadding="8" border="2" frame="vsides">
```

Because of its flexibility, you may find it convenient to use the `border` style property (Section 6.12) to display table rules, borders, and frames.

At the WDP website, the Ex: **TableStyle** example shows various table attribute settings. Note, the `rules` and `frame` attributes are poorly supported by many browsers.

Figure 3.11: Text Around Table



Four Seasons	
Spring	Summer
Fall	Winter

Here are some text around the table
Here are some text around the table
Here are some text around the table
Here are some text around the table
Here are some text around the table Here are some text
around the table Here are some text around the table
Here are some text around the table

3.7 Table Positioning

A table is normally positioned left adjusted by itself on the page. To center a table (or any block element) horizontally on the page you can use the margin style properties (Ex: **CenteredTable**):

```
<table style="margin-left: auto; margin-right: auto">
```

Add this code to Ex: **Cart** and see how it centers. If a table has a caption, you may need to add the auto margin style to the `caption` element as well[2]. You can also set the `caption-side` style property to `top` or `bottom` to place the caption above or below the table.

These margin properties can also take on a length or a percentage to control the horizontal placement of any element.

A table can also be positioned left- or right-adjusted with text flowing around it (just like text around images, Section 2.17). Figure 3.11 shows a table (Ex: **FloatTable**) coded as follows:

```
<table style="float: left; margin-right: 1em;
              background-color: #def">
```

[2]Older browsers such as IE 6 does not support auto margins and you may have to resort to the deprecated `center` element

```
<caption style="font-weight:bold">Four Seasons</caption>
<tr align="center">
<td>Spring</td> <td>Summer</td> </tr>
<tr align="center">
<td>Fall</td> <td>Winter</td> </tr>
</table>
<p>Here are some text around the table ...  </p>
```

Again, just as for floating images (Section 2.17), the `clear` property can be used to place an element clear after a floating table.

3.8 Table Width and Height

The width and height of a table is automatically computed to accommodate the contents in the table cells. You may also *explicitly suggest* a width using the `width` attribute of `table`.

The attribute `width="`*wd*`"` specifies the overall width of the table in pixels (`width="400"`) or as a percentage (`width="100%"`) of the available horizontal space. Use a percentage rather than a fixed length whenever possible.

If the suggested width is not enough, the table will be made wider automatically. If the suggested width is wider than necessary, the excess width is distributed evenly into the columns of the table. The height of a table is determined automatically by the accumulated height of cells and cannot be specified.

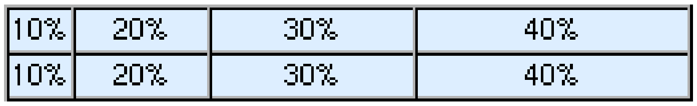
By default, all cells in the same table column have the same width. The width of the widest cell becomes the column width. Similarly, all cells in the same row have the same height. The height of the tallest cell becomes the row height. The width and height of each table cell are normally automatically computed to accommodate the cell content.

But you can also suggest a desired dimension for a table cell with the style properties `width` and `height`[3]:

- `style="width:` *wd*`"`—Sets the width of the element to *wd* which can be a fixed length or a percentage of the available horizontal space.

[3]The `width` and `height` attributes for the `td` element have been deprecated

Figure 3.12: Table and Cell Width



10%	20%	30%	40%
10%	20%	30%	40%

- `style="height: ` *ht*`"`—Sets the height of the element to a fixed length *ht*.

Incidentally, the `width` and `height` style properties apply in general to any block element and *replaced element*. A replaced element is an inline element, such as an image, that is loaded from a given URL.

For example (Ex: **TableWidth**), the following is a table taking up 60 percent of the available width, with no cell padding or spacing (line `A`), centered horizontally (line `B`), and having a light blue background (line `C`). The four columns takes up 10, 20, 30, and 40 percent of the table width respectively (lines `E`-`F`). Cell contents are centered horizontally (lines `D`). Figure 3.12 displays this table with two rows.

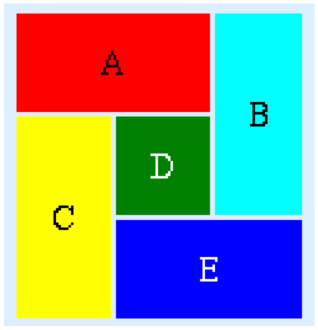
centered horizontally (line `B`), taking `60%` of the page width (line `A`), with 4 equally wide columns (lines `C`-`D`).

```
<table width="60%" border="1" cellspacing="0" cellpadding="0"    (A)
       style="margin-left:auto; margin-right:auto;               (B)
              background-color: #def">                           (C)
<tr align="center" >                                             (D)
<td style="width:10%">10%</td> <td style="width:20%">20%</td>    (E)
<td style="width:30%">30%</td> <td style="width:40%">40%</td>    (F)
</tr>
</table>
```

The height of a table cell is the minimum height required to accommodate the cell content. The cell height can be set with its `height` style property (`height: 60px`) but will always be large enough for the cell content.

If a table becomes wider than the display window, a horizontal scroll bar will appear and the end user must perform horizontal scrolling to view the whole table. Because horizontal scrolling is one of the most unpleasant tasks for uses, it is important to make sure your table fits well in a page and adjusts its width in response to different window sizes.

Figure 3.13: Row and Column Spans



In summary, you set the `width` attribute of `table` and the `width` style properties for `td`s to control table width. Note that the attribute `width="300"` and the style declaration `width: 300px` use different notations. Unlike HTML attributes, length values for style properties must be given with units (Section 2.11). The `height` style property of `td` can only be set to fixed lengths.

Width and height control are useful when applying tables to implement a layout grid for a page (Section 3.11).

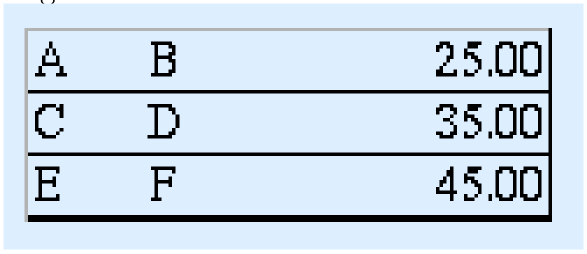
Row and Column Spans

A table cell can *span multiple rows and/or columns*. You use the `rowspan` (`colspan`) attribute to specify the number of rows (columns) a table cell spans.

We have already seen the use of `colspan` (line `9` Section 3.5). For a more complete example (Ex: **Spans**), let's look at how the table in Figure 3.13 is coded.

```
<table width="120">                                         (i)
<tr align="center">
 <td colspan="2"
    style="background-color: red; height: 40px">A</td>     (ii)
 <td rowspan="2" style="background-color: cyan">B</td>     (iii)
</tr>
<tr align="center">
 <td rowspan="2"
    style="background-color: yellow">C</td>                (iv)
 <td style="background-color: green;
     color: white; height: 40px">D</td>                    (v)
```

Figure 3.14: Table with Horizontal Rules



A	B	25.00
C	D	35.00
E	F	45.00

```
</tr>
<tr align="center">
 <td colspan="2"
     style="background-color: blue;
     color: white; height: 40px">E</td>                  (vi)
</tr>
</table>
```

The table has a total width of 120 pixels (line `i`). On the first row, the cell `A` which spans two columns (line `ii`) is followed by cell `B` that spans two rows (line `iii`). Thus, the table has a total of three columns. On the second row, the cell `C` which spans two rows (line `iv`) is followed by cell `D` that stays in a single row and column (line `v`). We don't specify the third cell for row 2 because that cell has already been taken. On the third row, cell `E` spans columns 2 and 3 (line `vi`) and that completes the table.

This example shows again control of the background color (lines `ii` and `iv`) and the foreground color (lines `v` and `vi`) for individual cells.

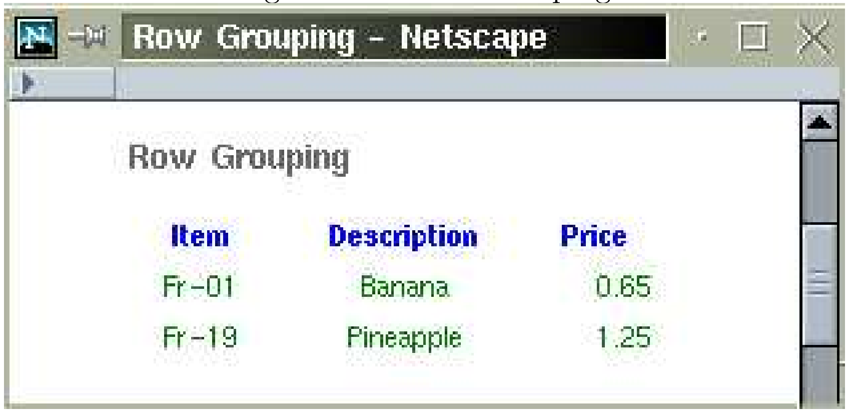
You can practice row and column spanning by making the mirror image of Figure 3.13 at the WDP website.

Rules between Cells

The `rules` attribute of `table` is not supported well at all by browsers. But you can still manage to place horizontal and vertical rules in tables without setting `border="1"`. The `frame="box"` attribute gives a simple box around the table. The `border` style properties (Section 6.12) on individual cells can be used to piece together desired rules in a table.

For example (Ex: **Rules**), the table

Figure 3.15: Row Grouping



Row Grouping

Item	Description	Price
Fr-01	Banana	0.65
Fr-19	Pineapple	1.25

```
<table width="150" cellspacing="0" frame="box">
<tr>
<td style="border-bottom: thin #000 solid">A</td>
<td style="border-bottom: thin #000 solid">B</td>
<td style="border-bottom: thin #000 solid"
    align="right">25.00</td>
</tr><tr>
 ...
</tr><tr>
 ...
</tr></table>
```

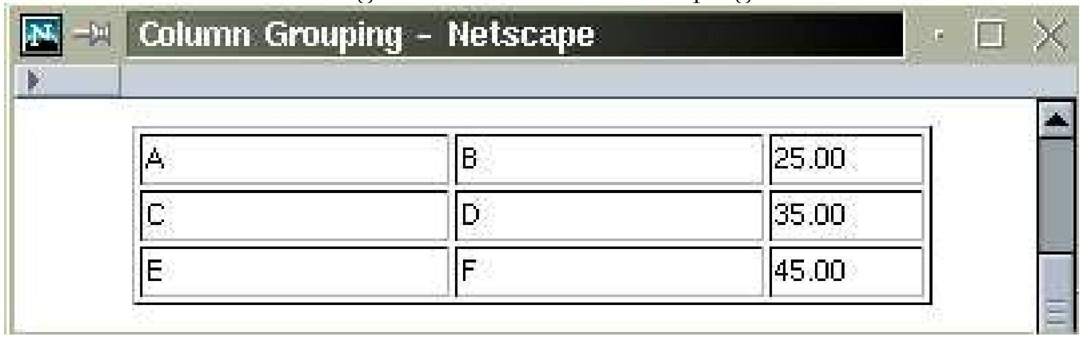
with three rows coded entirely the same way gives the display shown in Figure 3.14.

3.9 Row and Column Grouping

Rows can be grouped by `thead` (a group of header rows), `tbody` (a group of table rows), or `tfoot` (a group of footer rows to be displayed at the end of the table). For example (Ex: **RowGroup**),

```
<table width="200">
<thead align="center" style="color: blue">
  <tr><th>Item</th><th>Description</th><th>Price</th></tr>
</thead>
<tbody align="center" style="color: green">
  <tr><td>Fr-01</td><td>Banana</td><td align="right">0.65</td></tr>
```

Figure 3.16: Column Grouping



A	B	25.00
C	D	35.00
E	F	45.00

```
  <tr><td>Fr-19</td><td>Pineapple</td><td align="right">1.25</td></tr>
</tbody>
</table>
```

has a blue `thead` row group and a green `tbody` row group (Figure 3.15). Thus, row grouping allows you to specify alignment and style attributes for a group of rows at once.

Column grouping is also possible with the `colgroup` and `col` elements. Column groupings are given right after any table caption and before everything else in a table. With column grouping, width and alignment settings can be done for all cells in one or more columns at once. But, support for column grouping is varied among browsers.

For example (Ex: **ColGroup**),

```
<table border="1" width="300">
<colgroup span="2" width="40%" />              (I)
<colgroup span="1" width="20%" />              (II)
<tr> <td>A</td><td>B</td><td >25.00</td></tr>
<tr> <td>C</td><td>D</td><td >35.00</td></tr>
<tr> <td>E</td><td>F</td><td >45.00</td></tr>
</table>
```

specifies `40%` width (line `I`) for the first two columns and `20%` width (line `II`) for the third column (Figure 3.16).

The colgroup `width` gives the width for each column in the group. The value is a pixel number, a percentage of the table width, or a *relative length* expressed as `n*` where `n` is an integer. A width "2*" is twice as wide as "1*". The width `0*` specifies the minimum width needed by the column content. For example,

```
<table>
<colgroup span="4" width="2*" />           (III)
<colgroup span="3" width="1*" />           (IV)
<tbody>
   . . .
</tbody></table>
```

specifies each of the first four columns (line`III`) to be twice as wide as each of the next three columns (line `Iv`).

You may also use the closing tag `</colgroup>` and enclose zero or more `col` elements within. A `col` element supplies alignment and width attributes for one or a span of columns.

Table Structure

A `table` element follows a well-defined structure. It contains an optional `caption`, followed by zero or more `col` and `colgroup` elements, followed by an optional `thead`, an optional `tfoot`, and then one or more `tbody`. The `tbody` tags are optional when a table has one `tbody` and no `thead` or `tfoot`.

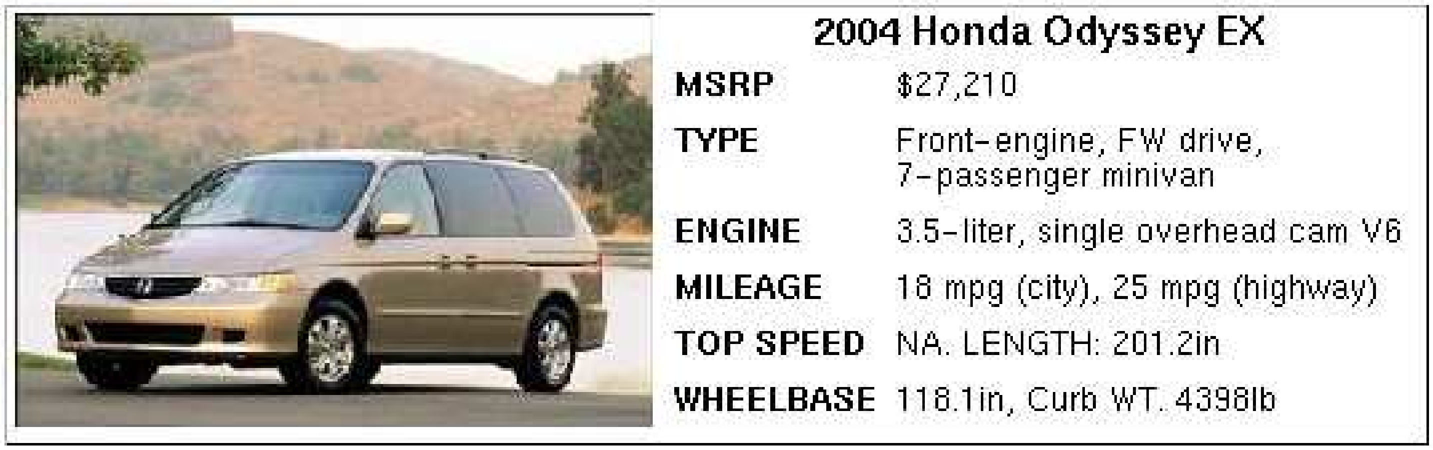
3.10 Table Nesting

You can put one table inside another easily. All you do is to enclose a table in a `td` element of another table. All the complicated formatting and positioning will be done automatically for you.

Figure 3.17 shows a table (Ex: **Nest**) with one row containing two cells: a picture and an information table. The superstructure of the nested table is:

```
<table frame="box" cellpadding="0">
<tr valign="top">                                          (A)
<td><img src="honda.jpg" alt="honda odyssey"
         width="240" height="155" /></td>
<td>
   <!-- place the nested table here -->
</td></tr>
</table>
```

Figure 3.17: Table Nesting



2004 Honda Odyssey EX

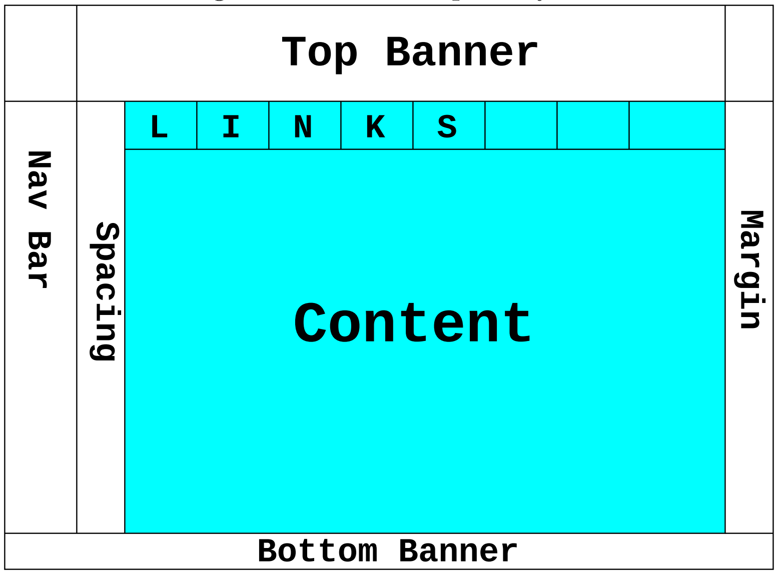
MSRP	$27,210
TYPE	Front-engine, FW drive, 7-passenger minivan
ENGINE	3.5-liter, single overhead cam V6
MILEAGE	18 mpg (city), 25 mpg (highway)
TOP SPEED	NA. LENGTH: 201.2in
WHEELBASE	118.1in, Curb WT. 4398lb

Note the vertical alignment used (line `A`) to make sure the image and the inner table will be placed correctly.

And the table nested inside is just another table structure:

```
<table  cellspacing="5">
  <caption style="font-size: larger; font-weight: bold">
       2004 Honda Odyssey EX</caption>
  <tr valign="top">
     <th align="left">MSRP</th><td>$27,210</td></tr>
  <tr valign="top">
     <th align="left">TYPE</th>
     <td>Front-engine, front-wheel
          drive, seven-passenger minivan</td></tr>
  <tr valign="top">
     <th align="left">ENGINE</th>
     <td style="whtie-space: nowrap">3.5-liter,          (B)
         single overhead cam V6</td></tr>
  <tr valign="top">
     <th align="left">MILEAGE</th>
     <td>18 mpg (city), 25 mpg (highway)</td></tr>
  <tr valign="top">
     <th align="left">TOP SPEED</th>
     <td>NA. LENGTH: 201.2 inches</td></tr>
  <tr valign="top">
     <th align="left">WHEELBASE</th>
     <td>118.1 inches</td></tr>
  <tr valign="top">
```

Figure 3.18: A Sample Layout



```
     <th align="left">CURB WT.</th>
     <td>4,398 pounds</td></tr>
</table>
```

Note the top vertical alignment for rows and left alignment for `th`. We also want to prevent line wrapping and preserve enough table width for the inner table. We can do this by using the `nowrap` declaration on a long line (line `B`).

3.11 Using Tables for Page Layout

A clear and attractive Web page usually follows an underlying *layout grid* (Section 5.12) to position various visual components in the page. The `table` element offers a practical way to create the underlying grid and to make it work for variable window sizes and screen resolutions. The ability to nest tables makes even complicated page layout easy to implement. Figure 3.18 shows a sample page layout with top and bottom banners, left navigation bar, links, spacing, and right margin. Follow these rules of thumb to define `table`-based alignment grids:

- Use `width="100%"` to make sure a table takes up all the available horizontal space.

- Use `style="width:`*xx*`%"` for all cells in some selected row so the column widths follow a designed proportion (See Section 6.15). Pick a row with the least row spanning in it. Don't provide widths in more than one row to avoid inconsistent settings.

- Use attributes `border="0"`, `cellpadding="0"`, and `cellspacing="0"` to make sure that the underlying grid stays *unseen*.

- Use `valign="top"` so contents in cells on a table row align along the top edge of the row.

- Do not make the grid into one big table when using nested tables can simplify the overall structure.

- Mark the beginning and end of different parts with HTML comments such as,

  ```
  <!-- Top Banner Begin -->
  <!-- Top Banner End -->
  <!-- Content Begin -->
  <!-- Content End -->
  ```

  to make revising the page much easier.

- When piecing together a bigger picture with images in neighboring cells of a layout table, intervening gaps must be eliminated (Section 6.17).

To implement the grid shown in Figure 3.18 we can use a table of three rows and four columns. The third cell of the second row can contain another table for the cyan-colored part. The bottom banner may span all four columns.

3.12 Page-wide Style with `body`

The `body` element contains the content of an HTML page. To organize page content, you place block elements inside the `body` container.

Style properties attached to the `body` tag affects the presentation of the entire page. For example

```
<body style="color:navy; background-color: white"}
```

asks for `navy` characters over white background for all child elements in `body`. A child element can override parent style settings by specifying its own style properties, as we have seen in the preceding sections and in Chapter 2.

The page or element background can also be given by an image. Usually the background image is automatically repeated horizontally and vertically (like floor tiles) to cover the entire page. The background image may be stationary or scroll with the content. Use these style properties to manage the background image:

- `background-image:` *URL*—the *URL* links to an image (GIF, JPG, PNG)
- `background-attachment:` *how*—`scroll` or `fixed`
- `background-repeat:` *how*—`repeat-x` (horizontally) `repeat-y` (vertically), `repeat` (both ways), `no-repeat`

A search on the Web will turn up many sites with background image collections. Use background images only as part of the site's visual design to enhance communication, to increase readability, or to complement functionality. Make sure `background-color` is set to a color similar to the background image in case a browser does not support images.

To set page margins, use the style properties:

`margin-top:` *length*

`margin-bottom:` *length*

`margin-left:` *length*

`margin-right:` *length*

Similar to a printed page, margins give the content breathing room and are important page layout considerations. It is a good idea to use appropriate margins consistently for all pages in a site. For example Ex: **BodyStyle**,

```
<body style="color:navy; background-color: white;
             margin-top: 40px; margin-bottom: 40px;
             margin-left:50px; margin-right: 50px">
```

sets the left and right margins to 50 pixels and the top and bottom margins to 40 pixels.

Other `body` style properties to set in practice include `font-family`, `font-size`, `color`, and hyperlink styles

Consistently setting the style of the `body` element (Section 6.2) is important for the design unity (Section 4.5) of a site.

3.13 Head Elements

So far we have focused on element you place inside `body` to create page content. But HTML also allow you to attach administrative information, or *meta data*, for a page within the `head` element. The `head` element is required for each page and it encloses administrative elements for a page. Let's now turn our attention to these elements.

The `head` must contain exactly one one `title` element and zero or more optional elements:

- `base`—for page location (Section 3.16)
- `meta`—for various page-related information (Section 3.15 and 3.14)
- `style`—for in-page style sheet (Chapter 6)
- `link`—for links to related documents such as favicons (Section 3.17) or external style sheets (Chapter 6)
- `script`—for in-page or external scripts such as a Javascript program (Chapter 9).

The `meta` element has the general form

`<meta name="`*some_name*`" content="`*some_text*`" />`

and the values for *some_name* and *some_text* can be arbitrary. Well-established conventions for using `meta` are included in this chapter. The `meta` element is also used to place HTTP header information inside a page as illustrated by the page forwarding mechanism next.

3.14 Search Engine Ready Pages

Search engines use *robots* to continuously visit sites on the Web to collect and organize information into indexed and easily searchable databases. This gives users a quicker and easier way to find sites they want. To attract the right visitors to your site, it is important to provide search engines with the correct information about your site. There are a few simple but important steps you can take to make sure your site is effectively and correctly processed by search engines.

Use these elements inside the `head` element to make your Web page search engine ready:

- `<title> ... </title>`—Supply a precise and descriptive title.
- `<meta name="description" content=" ... " />`—Give a short and concise description of the content of the page.
- `<meta name="keywords" content="word1, word2,  " />`—List keywords that a person looking for such information may use in a search.
- `<meta name="robots" content="key1, ..." />`—Tell visiting robots what to do: `index` (index this page), `noindex` (do not index this page), `follow` (follow links in this page), `nofollow` (do not follow links in this page), `all` (same as `index, follow`), `none` (same as `noindex, nofollow`).

  For a regular page, you invite the robot to collect information from the page and follow links to recursively index all subpages with

  ```
  <meta name="robots" content="index, follow" />
  ```

For a page such as terms and conditions of service, copyright notices, advertisement for others, etc. that you do not want indexed as content of your site then use:

```
<meta name="robots" content="noindex, nofollow" />
```

to prevent such extraneous information to be indexed as representative of the type of information supplied by your site.

To further help visiting robots, you can place a `robots.txt` file at the *document root* directory of your domain. In the `robot.txt` file you can indicate that certain parts of your server are off-limits to some or all robots. Commonly the file is in the form

```
User-agent: *
Disallow: URL-prefix1
Disallow: URL-prefix2
```

List the URL prefix of any directory that is useless for a robot to index. Such a directory may be for images, cgi programs, administration of your site, internal documentations, password restricted files, etc.

The full details on the *Standard for Robot Exclusion* can be found at

```
www.robotstxt.org/wc/norobots.html
```

3.15 Page Forwarding

Web pages sometimes must move to different locations. But visitors may have bookmarked the old location or search engines may still have the old location in their search database. When moving Web pages, it is prudent to leave a *forwarding page* at the original URL, at least temporarily.

A forwarding page may display information and redirect the visitor to the new location automatically. Use the `meta` tag

```
<meta http-equiv="Refresh" content="8;  url=newUrl" />
```

The `http-equiv` `meta` tag actually provides an HTTP response header (Section 1.14) for the page. The above `meta` element actually give the response header

```
Refresh    8; url=newUrl
```

The effect is to display the page and load the *newUrl* after 8 seconds.

Here is a sample forwarding page

```
<head><title>Page Moved</title>
<meta http-equiv="Refresh" content="8; url=target_url" />
</head><body>
<h3>This Page Has Moved</h3>
<p>New Web location is: ... </p>
<p>You will be forwarded to the new location
   automatically.</p>
</body></html>
```

The `Refresh` response header (`meta` tag) can be used to refresh a page periodically to send updated information to the end-user. This can be useful for displaying changing information such as sports scores and stock quotes. Simply set the refresh target url to the page itself so the browser will automatically retrieve the page again after the preset time period. This way, the updated page, containing the same `meta refresh` tag, will be shown to the user.

3.16 Portable Pages

Sometimes, a page is downloaded from the Web and saved on your PC. But, in all likelihood, the saved page won't display correctly in a browser because it needs images and other files that now can't be found. This happens because the hyperlinks to the needed images and files are relative to the original location of the page.

You can fix this easily by adding the `base` element in the header:

```
<base href="Full_URL_of_original_location" />
```

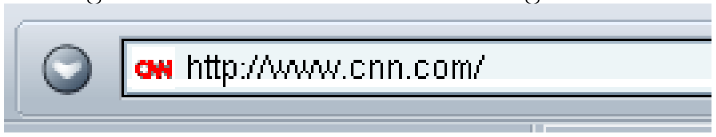
The `base` provides a browser an explicit base upon which to interpret all relative URLs in the page. Now your saved page works as if it were in its original location.

If a site expects certain pages to be used after being downloaded, then those pages should have the `base` tag in place already.

Pages generated dynamically by CGI programs can also use this technique to easily use images and links independent of the program location (Section 13.15).

3.17 Website Icons

Figure 3.19: Location Box Showing Favicon



A website may define a small icon, a tiny logo or graphic, that is displayed by browsers in the URL location box. It is called a *favicon* and is also used in bookmarks and favorite-site listings. Figure 3.19 shows the favicon of the CNN stie.

Favicons can help brand a site and and distinguish its bookmarks. To install a favicon for your site, follow these two steps.

1. Create a `favicon.ico` file for the icon and place it in the *document root* directory of your Web server. If the document root is not accessible to your site, then place `favicon.ico` at the top directory of your site.

2. Add the header element

   ```
   <link rel="shortcut icon" href="/favicon.ico" />
   ```

   in the `head` element for all `index.html` and other key pages on your site. It tells browsers where the favicon is located, if it is not at document root. It also allows different pages to use different favicons (seldom done). You do not need to insert the `shortcut` link tag in pages if you are able to place the favicon in document root.

To create the `favicon` file, you may use any raster graphics editor to get a 16-color 32×32 graphic. Then save it as an ICO (Microsoft icon format) file. Tools are available to convert `.bmp`, `.gif`, and other image formats to `.ico`.

3.18 Frames

Normally a browser window displays one Web page. The `frameset` element, used instead of `body`, allows you to divide a window into rectangular regions, called *frames*, and display a different page in each frame. The identification, size, and scroll ability of each frame can be configured by attributes as well. Hyperlinks in one frame can change the displayed content in another frame.

Frequently, frames are used to divide a window into fixed-display and variable-content regions. Constant materials, such as a top banner, a table of contents, a logo, a navigation bar, and a copyright notice, can be placed in their own frames and made always visible. Clicking on a link in a fixed frame can update the information displayed in the content-display frame. This situation is not new. Look at the top and bottom of your browser window. The always visible include the title bar, the menu bar, the tool bar, and the status bar.

Frames can give a page a "broken-up" look or a monotonous look. When the content frame shows a page not designed to display together with the fixed frames, the page can look down right terrible. Most style experts will tell you to avoid frames and others will even say "misuse of frames is one of the ten worst problems on the Web today."

Frames can be effective in certain situations. But it should be used carefully and it is not for every situation. Make sure you use the correct XHTML `DOCTYPE` declaration

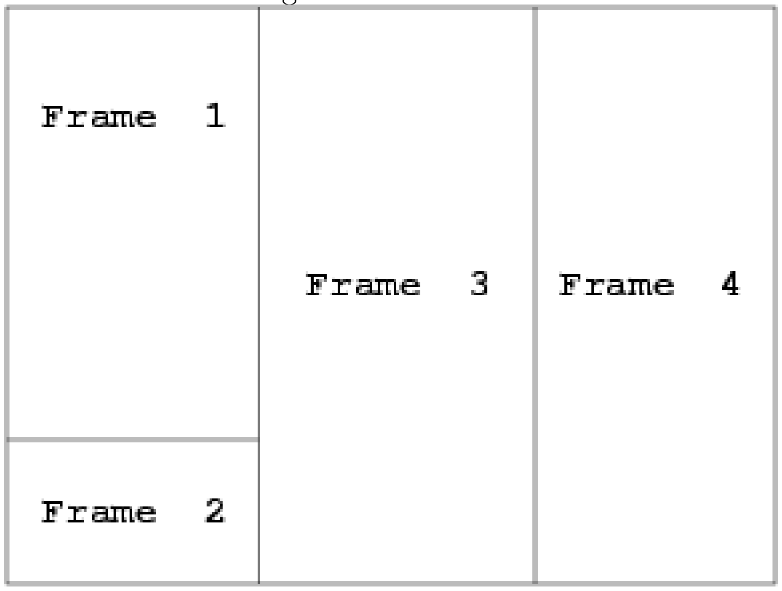
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

for any frameset page. In such a page the `body` element is not allowed. XHTML Strict, the future of HTML, does not support frames.

Using Frames

Use the `rows` and `cols` attributes of `<frameset>` to subdivide the browser window into frames. For example,

Figure 3.20: Frames



`cols="25%,75%"` (1st column width 25%, 2nd column 75%)

`rows="100,*"` (1st row height 100 pixels, 2nd row the rest)

The `cols` values specify the width of each column from left to right. The `rows` values give the height of each row, from top to bottom. If both `rows` and `cols` are specified, a grid, filled in row-major order, is defined.

The value for `rows` (`cols`) is a comma-separated list of lengths in pixels (e.g. 100), percentages of the available window height (width), or relative units (e.g. `3*`). For example, `cols="*, 3*, 200"` gives 200 pixels to the third column and three fourth of what's left to the 2nd column.

If a row or column is given a dimension of zero, for example,

`rows="0,*"` (first row is hidden)

then it is hidden. If you also get rid of frame borders (see next subsection), then even a single-window looking page can be a frame.

Within a `frameset` you may place `frame` and `frameset` elements to fill the defined grid in row-major order. For example,

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
   <title>A frameset example</title>
</head>
<frameset cols="*,*,*">                               (1)
  <frameset rows="*,200">                             (2)
     <frame name="frame1" src="a.html" />
     <frame name="frame2" src="graphics.gif" />
  </frameset>                                         (3)
  <frame name ="frame3" src="b.html" />               (4)
  <frame name ="frame4" src="c.html" />               (5)
</frameset>                                           (6)
</html>
```

The outer `frameset` (line `1`) gets you three equal columns. The first column is occupied by an inner `frameset` (line `2-3`) that has two rows. The second column is filled by `frame3` (line `4`) and the third column (line `5`) filled by `frame4` (Figure 3.20).

Note that the `frameset` is used instead of a `body` in an HTML document. The `<noframes> ... </nofra`
element (placed between lines `5` and `6`) can be used to supply alternative `body` for browsers that do not support frames. For example,

```
<noframes>
<body>
<p>This page uses frames, but your browser doesn't support them.
Here is a <a href="regular.html">no-frames version</a>. </p>
</body>
</noframes>
```

Frame Borders

Frames in a frameset are usually separated by borders which can be used to resize the frames.

In certain applications, you need to remove the borders around frames. This means not drawing the frame borders and eliminating the space between the frames. According to HTML, all you need to do is to set

```
frameborder="0"
```

for all frames involved. Unfortunately, this is not enough for the current NN or IE. You also need to include the non-standard `frameset` attributes:

```
<frameset  ...
   border="0"                     (for NN)
   frameborder="0"                (for IE)
   framespacing="0"               (for IE)
>
```

The `frame` Element

The `frame` is an element to supply attributes. Neither end tag nor content is allowed.

- `name="`*id*`"`–identifies the frame
- `src="`*URL*`"`–supplies content
- `frameborder="`*flag*`"`–enables/disables frame border; *flag* is `1` or `0`.
- `marginwidth="`*pixels*`"`–sets left and right margin width
- `marginheight="`*pixels*`"`–sets top and bottom margin height
- `noresize`– disables frame resizing
- `scrolling="`*flag*`"`–controls the display of scroll bars; (*flag* is `yes`, `no` or `auto`. The `auto` setting displays a scroll bar only when necessary.

The frame name must begin with a letter. The name allows you to specify a `target` frame for any element involving an `href` attribute, namely `a`, `area`, `base`, `form`, and `link`.

Targets

The `target` attribute can be used in any element that provides a hyperlink. If the target is the name of a frame, then the referenced information is destined to that frame. Thus, in `frame1` the link

```
<a href="index.html" target="frame3">homepage</a>
```

displays `index.html` in `frame3` when clicked.

When displaying frames, the browser `Location` box shows the URL of the frameset document, not the URL of any page contained in any of its frames. Right clicking a frame and selecting `Frame Info` in the pop-up menu shows you the title and URL of the page in that frame. The the browser `Back` and `forward` buttons work on a linear history of pages visited. In the context of frames, the history includes the frame targets of the pages as well as their URLs. Thus, these buttons always work, frames or not.

Without an explicit `target`, the default target for a hyperlink is the frame containing the link. `target` in the `base` element can set the default target for all un-targeted links in a page.

In addition to named frames, there are also *known targets*:

- `_blank`—A new, unnamed top-level window
- `_self`—The same frame containing the link, overriding any `base` specified target
- `_parent`—The immediate frameset parent of the current frame
- `_top`—The full, original window (thus canceling all other frames)

The `_blank` target is often useful for displaying links external to a site. If you do use these targets, make sure the page is declared XHTML Transitional.

Frames Pros and Cons

While frames are tempting, their use does present a series of problems for a website. The combined weight of these problems is serious enough for many to consider frames to be the number one mistake in website implementation.

Here are the top-10 reasons.

1. A significant number of browsers do not support frames. Having to create the *noframes* version of the pages essentially negates the convenience frames offer. If the noframes versions are not provided, non-frame browsers will display a blank page.

2. Because a frameset uses a different navigation model than the generally understood hyperlink, many Web surfers can get confused by the behavior of frames.

3. The strength of the Web is hyperlinks. But an outside link to an internal page of a frame-based site do not work. The link leads to a page without its associated frames making it an incomplete page.

4. When visiting a frames site, the browser location box does not indicate the displayed page, just the frameset. When you visit different pages in such a site, the location box stays the same.

5. Bookmarking internal pages of a frames site is difficult and the saved links do not work due to reason number 3.

6. Links from a frames page to off-site pages always require a new top-level window. Otherwise, the outside page will be framed in the wrong context.

7. Search engines have trouble with frames since they don't know what composites of frames to include as navigation units in their index.

8. Browsers are likely to have page printing problems with a frames page. For example, each part on the screen that belongs to a different frame may be printed on a separate page.

9. Links to file types such as PDF that uses browser plug-ins may have problems or be limited to a small area of the window.

10. Applications of frames mostly use them for banners and navigation bars. Such design gives a very ordinary and boxy look with one rectangle on top and one rectangle on the left. From a design standpoint that is far from innovative or attractive. Fixed banners and navigation bars can be achieved with CSS positioning (Section 6.20) which is more flexible and does not limit you to the boxy look. Besides, the CSS approach has no frames-induced problems.

The frames construct has its advantages. It updates only the contents making pages appear quicker. The different frames can be resizable and scrollable making very lengthy navigation bars possible. Viewers can also reduce the decorative top banner to give more screen space for the content. Certain self-contained sites may wish to discourage outside links into the middle of the site. In that case, a frames organization can really help. There are situations that call for the use of frames. But, a Web designer must think twice before deciding on a frame-based design.

3.19 Server-Side Includes

The basic idea of *server-side includes* (SSI) is simple–You put special instructions, called *SSI directives*, inside a Web page for the Web server to include files or other information into the page. The page delivered to the client side is the result after the inclusions have been made.

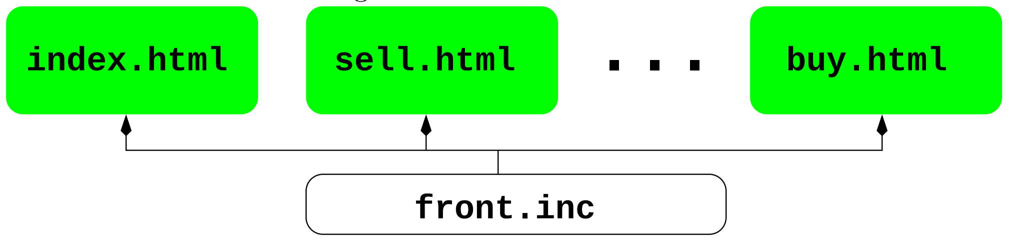
To use SSI, you usually need to give the file the suffix `.shtml` or make the `.html` file executable:

```
chmod +x page.html
```

It depends on your server configuration and you need to check with your server administrator.

SSI directives are placed inside HTML comments. For example,

Figure 3.21: File Inclusion



```
<!--#echo var="DATE_LOCAL" -->       (Server-side Date and time)
<!--#echo var="LAST_MODIFIED" -->    (Last modified time of page)
```

can be handy to include such changing information in a page. Make sure there is no space before the # character.

To include another file, use either of these two directives:

`<!--#include file="`*relative-path*`" -->`

`<!--#include virtual="`*local-URL*`" -->`

The *relative-path* is a file pathname relative to the directory containing the page. The `local-URL` is a URL relative to the page or to the server root (`/footer` or `/cgi-bin/prog.cgi` for example).

Often a website will have many pages who share a common front part and perhaps also a back part. It is convenient to keep them in header and footer files included in pages with SSI (Figure 3.21). This way any changes in the header/footer will be very easy to make.

Modern hosting services also support PHP, a much more powerful server-side include and programming language, that can be used for including files. Typically PHP directives can be placed in regular `.html` files. This is an advantage over SSI because you don't have to use a different file suffix or to add execution permission.

It is beyond the scope of this text to discuss PHP. But we do need to cover a few points. PHP directives are given in a Web page as follows.

```
<?php  . . . ?>         (Full syntax)
<?  . . . ?>            (Shorthand)
```

To include another file with PHP use

```
<? include("filename"); ?>
```

The shorthand PHP syntax conflicts with the very first line required by an XHTML file:

```
<?xml version="1.0" encoding="UTF-8"?>
```

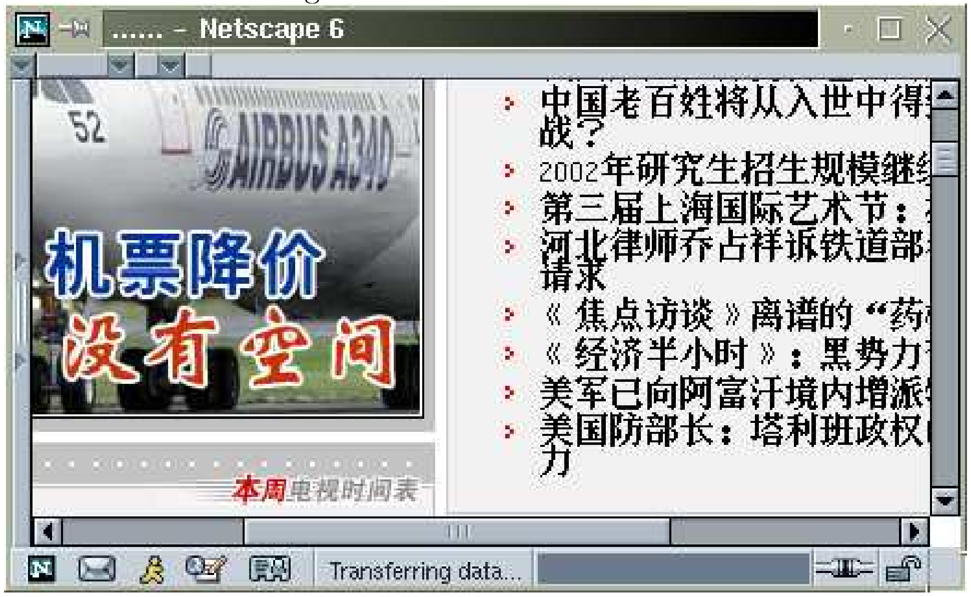
which will cause a PHP syntax error. To server XHTML pages from a server that supports PHP, you need to send the first line this way

```
<? print('<?xml version="1.0" encoding="UTF-8"?>');  ?>
```

which instructs PHP to produce the first line verbatim.

3.20 Internationalization

Figure 3.22: A Site In Chinese



- 中国老百姓将从入世中得
  战？
- 2002年研究生招生规模继
- 第三届上海国际艺术节：
- 河北律师乔占祥诉铁道部
  请求
- 《焦点访谈》离谱的“药
- 《经济半小时》：黑势力
- 美军已向阿富汗境内增派
- 美国防部长：塔利班政权
  力

The Web is international and XHTML documents allow all ISO10646 (Unicode) characters (Section 3.2). The *primary language* for an HTML file is set by the `lang` attribute of the `html` tag (Section 2.1):

```
<html     ...     lang="language">
```

where *language* is a two-character (case insensitive) language code specified by the International Organization for Standardization (ISO) *Code for the representation of names of languages* (ISO 639). Table 3.1 shows the two letter codes of some languages. The `lang`

Table 3.1: Some Two-Letter Language Codes

Code	**Language**	**Code**	**Language**
`AR`	Arabic	`DE`	German
`ES`	Spanish	`EN`	English
`FR`	French	`IT`	Italian
`IW`	Hebrew	`JA`	Japanese
`RU`	Russian	`ZH`	Chinese

attribute can be included with any element to indicate a language which may be different from the primary language of the page. Support for the `lang` attribute from browsers is increasing. The `dir` attribute specifies the base direction of *directionally neutral text*, that is text without an inherent directionality as defined in UNICODE, or the directionality of tables: `dir="LTR"` (left-to-right), `dir="RTL"` (right-to-left). A `RTL` table has its first column on the right side of the table.

An international page may also use a different character set encoding. To specify the character encoding for a document, use both the encoding attribute specification on the `xml` declaration and and a `meta http-equiv` element. For example,

```
<?xml version="1.0" encoding="gb2312"?>
```

and the `meta` element

```
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
```

indicate the *simplified Chinese* `gb2312` character set (Figure 3.22). The two values ought to be the same. In case they are different, the `encoding` setting in the `xml` declaration takes precedence.

Examples of other `charset` settings are:

`charset=utf-8`	(Unicode)
`charset=big5`	(Traditional Chinese)

You can see many examples in the character set option menu of your Web browser.

3.21 Common Page Errors

Before publishing a Web page we must check and make sure it is free from errors. Testing a page with a browser is a start. Of course it is best to avoid errors in the first place.

We list some common Web page errors here to help you avoid them.

- Missing or extra end tags—If you use a regular editor, insert and delete the begin and end tags of each element at the same time. This way open and close tags always come and go in pairs.
- Missing `/` for elements without end tags—Always use the three-character " `/>`" to close elements. without end tags.
- Overlapping tags—For example, `<li><p> ...</li></p>` is overlapping and incorrect. If you insert begin and end tags together, you can largely avoid tag overlapping as well.
- Missing quotation marks for attributes—For example, `width=50` or `width="50` is incorrect. Remember to always use quote marks for attribute values. Insert both begin and end quote marks at the same time to avoid missing quote marks.
- Upper-case tag and attribute names—XHTML requires all tag and attribute names to be in lowercase.
- Incorrect Element placement—Placing an inline element where only block elements are allowed, or visa versa, is largely tolerated by browsers but not allowed in XHTML. For example `br` or `img` can't be placed as a direct child of `body` and a list can't be a direct child of another list.

- Table rows with different number of cells—This is a hard error to detect. When constructing a table with multiple rows, consider duplicating the first row a number of times then edit the rows for content and formatting. This way at least the cell count will be correct for all rows.

- Missing required attributes—For example, `alt` (for `img` and `area`), and `id` (for `map`) ?img? are required.

- Missing, incorrect or useless title—The `title` element is required. Use a correct and descriptive `title` for each page. Keep in mind browser bookmarks are named with page titles. Titles such as "homepage" are useless.

- Broken links—Incorrectly entered links or links to pages that have been moved.

- Spelling and grammar errors—Proof read, copy edit, and use a spell checker.

- Misspelled entities—These result in strange characters on the page. Make sure your entities are correctly typed and that pages don't contain extraneous characters from copy and paste or file transfer.

- Illegal characters—Characters such as `<` and `&` are not allowed directly. Use `<` only to start a tag. Use `&` only to start an entity. Everywhere else, use entities to introduce such characters.

3.22 Page Checking and Validation

After creating or extensively editing a page, you should first pass it through a spelling checker. The Netscape Composer has a spell checker that understands HTML. On the PC, word processors such as MS/Word can be used to check spelling.

On UNIX systems **spell** is usually available for spelling checking. The **aspell** checker has an HTML mode so it will ignore tags and focus on textual content.

The UNIX command

aspell `-l -H <` *file*`.html >` *output-file*

will produce a list of suspect words.

To see if your XHTML code meets standards, you can use the free HTML/XHTML code validator available at W3C:

`http://validator.w3.org/`

You pick a document type and enter the URL of the page you wish to check. It is that easy. The result of the check lists any errors, their location in the source code, and their possible causes. The source code lines numbers is also displayed to aid debugging. If the page to check does not have an URL, then you can use the "check uploaded file" feature to check files on your local hard drive. The WDP website offers hands-on experiments to guide you through the page validating process. Other validators are also available on the Web. Make sure you choose one that understands XHTML.

When validating your XHTML pages, it is necessary to keep in mind that there are three XHTML standards: `Strict`, `Traditional` and `Frameset`. Each has its own `DOCTYPE` declaration (Section 2.4). If the page contains any deprecated tags (e.g. `center` or `font`) or disallowed attributes (e.g. `bgcolor`, `topmargin`, `background`, and `target`), then your `DOCTYPE` is XHTML Traditional. If the page uses `frameset` then the `DOCTYPE` needs to indicate `Frameset`. Only when a page is free of such constructs, can it have a chance to validate successfully under XHTML `Strict`.

Comprehensive page checkers examine many aspects of a Web page including spelling, broken links, coding errors, coding improvements, and load time, then make suggestions for improvement. Commercial versions are available at places such as DoctorHTML

`www2.imagiware.com/RxHTML/`

and NetMechanic

`netmechanic.com/toolbox/html-code.htm`

These websites typically also offer to check your pages as a trial of their software or services. It is a good idea to check your pages thoroughly before placing them on the Web.

3.23 For More Information

XHTML 1.0 is stable but XHTML is evolving. The W3C site (`www.w3c.org`) has up-to-date and complete specifications of XHTML recommendations.

The following websites are good references:

- XHTML 1.0 Recommendation: `www.w3.org/TR/xhtml1/`
- HTML 4 Elements: `www.w3.org/TR/html4/index/elements.html`
- HTML 4 Element Attributes: `www.w3.org/TR/html4/index/attributes.html`
- HTML 4 Reference at WDG: `www.htmlhelp.com/reference/html40/`

The WDP website has additional resources for HTML.

3.24 Summary

HTML documents are international and may contain characters from a specified character set using a designated encoding. HTML entities use ASCII character sequences to represent non-ASCII characters. They make it easy to enter non-ASCII characters from ASCII keyboards.

The `table` element can be used for presenting information in a tabular form and to provide page layout grids that contain cells where graphics and textual contents are placed.

The full table super structure looks like this:

```
<table>
<caption> ... </caption>                 (0 or 1)
<colgroup> ... </colgroup>               (0 or more)
<col />                                  (0 or more)
<thead> <tr> ... </tr>   ...  </thead>   (0 or 1)
<tfoot> <tr> ... </tr>   ...  </tfoot>   (0 or 1)
<tbody> <tr> ... </tr>   ...  </tbody>   (0 or more)
</table>
```

Table rows (`tr`) contain cells (`td` or `th`). A cell can span multiple rows and/or columns. Each cell has a padding and a spacing from neighboring cells. The table may have an outside frame as well.

Table cell contents can be aligned vertically (`top`, `middle`, `bottom`, `baseline`) and horizontally (`left`, `center`, `right`, `justify`, `char`) and have individual styles. A `td` may contain inline and block elements including a `table`. Thus, tables can be nested. By default, a table is left aligned. However it can easily be centered (`margin-left: auto; margin-right: auto`), floated to the left/right (`float: left`), or otherwise positioned on a page.

The `body` element provides a place to define page-wide styles such as margins, background and foreground colors. fonts, and background images.

Elements for the `head` can be used for page refreshing, page forwarding, favicon provision, and for making a page search engine ready. Other head elements attach style sheets and Javascript programs to a page, as you'll see in later chapters.

Frames present several Web pages in designated areas in the browser window. It represents a departure from the window-per-page navigation model and can be confusing to users. The pros and cons discussed in this chapter allow you to choose the right situations to apply Frames.

SSI (`<!--# ... -->`) or PHP (`<? ... ?>`) codes enable you to include other files into an HTML file making it easy to share common parts of pages your site. This can be important for the maintainability of a site.

Avoid common HTML coding mistakes listed in this chapter and check/validate your pages before putting them on the Web.

Exercises

Review Questions

1. Which part of an XHTML file specifies the document language? the document character set? the document character encoding? Show the XHTML code.

2. Consider UTF-8 and UTF-16 character encodings? Why do we say UTF-8 is ASCII preserving?

3. UTF-8 seems to use more bits than UTF-16 for character at higher code positions. What advantage does UTF-8 has over UTF-16?

4. Name four ways to enter the Greek character π in a Web page.

5. Consider HTML tables. How do you center a table? Float a table to the left or right?

6. Are there situations when a correctly specified `table` has rows containing different numbers of `td` elements? Explain.

7. What attributes are available for `table`? `tr`? `td`?

8. Is it possible to specify the height of a table? The height of a table cell? How?

9. Consider Frames. Why is it difficult to link to an internal page of a Frames site?

10. List the elements and information you need to place in a page to make it search-engine ready.

11. List some common HTML coding errors not listed in Section 3.21.

Assignments

1. Enter the mathematical formula $sin(\pi \,\mathrm{sup}\, 2)$ into a Web page and display the results.

2. College fraternities often use names with Greek letters. Enter your favorite fraternity name into a page.

3. Take a simple table with a caption and center the table. Does the caption automatically go with the table or do you need to center the caption separately? Experiment with different browsers and see for yourself.

4. Consider specifying table cell `width` with percentages. What happens if the total adds up to more than `100%`? What happens if one or more cells have no `width` specified while others have percentages? Experiment and find out.

5. By using background colors for the table and the cells, you can make thin rules appear between cells that are sometimes more appealing than what you get with settings such as `border="1"`. Experiment.

6. Consider `cellspacing` and `cellpadding`. Is it true that they are set to zero if you do not set them? Explain.

7. When centering a block element such as `table`, what is the difference between

   ```
   style="margin-left: auto; margin-right: auto"
   ```

   and

   ```
   style="margin-left: 20%; margin-right: 20%"
   ```

   Experiment and explain.

8. Practice row and column spanning by creating an HTML table to present the mirror image of Figure 3.13.

9. Access the W3C XHTML validator and use it on some of your pages. Look at the results and fix any problems.