CHAPTER

6

Page Layout with CSS

Chapter Objectives

In this chapter, you will learn how to ...

- Describe reasons to use CSS for page layout
- Use relative and absolute positioning
- Apply the CSS Box Model
- Configure basic page layouts using CSS
- Configure two-column page layouts using CSS
- Locate CSS page layout resources

Now that you are familiar with using CSS to format text and color, you are ready to explore using CSS to configure Web page layout. This method relies on CSS properties rather than tables to design a Web page. The technology for this layout is called CSS-P, for CSS positioning. This chapter introduces you to configuring page layouts using CSS.

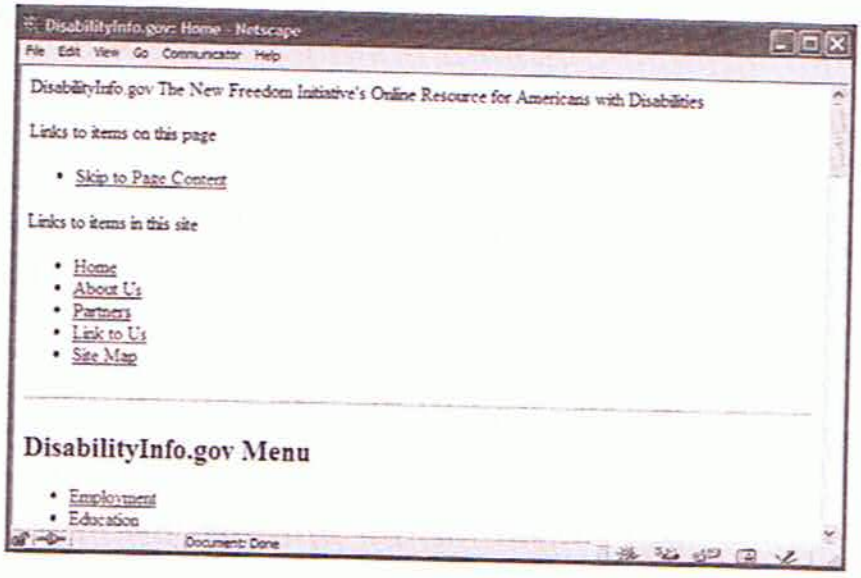


Figure 6.2
The DisabilityInfo.gov home page displayed in Netscape 4.79, an outdated browser

it is coded in the Web page. Because the site configures most of its graphics as background images within the external CSS style sheet, these graphics are unavailable to the older browser. The visitor experience is not exactly compelling when using an out-of-date browser. However, the site can still be navigated and information can be obtained.

Figure 6.3 shows the same page displayed in an Opera browser configured to simulate a text browser. The text content is displayed in the order it is coded in the Web page. Notice how even with the newer CSS page layout techniques utilized, the page is usable when rendered in a text browser.

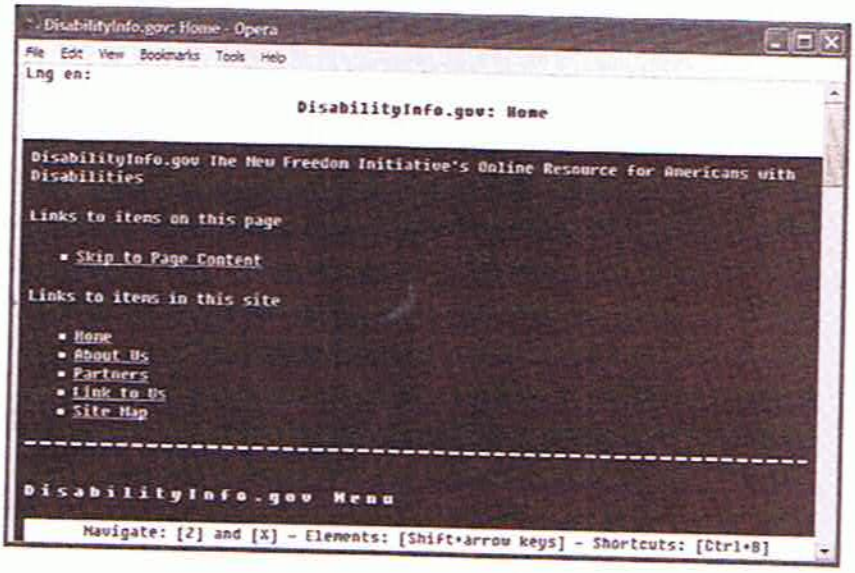


Figure 6.3
A text browser simulation of DisabilityInfo.gov's home page

Web developers have long delayed using CSS for page layout because of these rendering issues. What has recently changed? As modern browser support of CSS has increased, the number of Web visitors who use older browsers has decreased. For a typical Web site, less than 1 percent of visitors use these older browsers. Depending on your site's

target audience, this figure could be higher or lower—your Web logs (see Chapter 13) will provide this information. For example, TruGreen (http://trugreen.com) and wired (http://wired.com) are two organizations that use CSS for page layout. Although many existing Web sites use XHTML tables (see Chapter 8) to configure page layout, most Web sites developed today use CSS for this purpose.

Advantages of Using CSS for Page Layout

When CSS is used to configure page layout in addition to formatting text and color, the following advantages of using CSS for formatting are enhanced:

- **Greater Typography Control.** This includes font size, line spacing, letter spacing, indents, margins, and element positioning without using the XHTML table element (discussed in Chapter 8).
- **Style Is Separate from Structure.** The formatting and page layout can be configured and stored separately from the body section of the Web page document. When the styles are modified, the XHTML remains intact. This means that if your client decides to change something as small as the background color or as potentially huge as the page layout, you may only need to change one file that contains the styles, instead of each Web page document. For a look at how very powerful this can be, visit http://www.csszengarden.com and be amazed at how different pages with the same content and XHTML code (but different CSS) can look!
- **Potentially Smaller Documents.** Since both the formatting and page layout are separate from the document, the actual .html documents should be smaller.
- **Easier Site Maintenance.** Again, if the styles or page layout need to be changed it may be possible to complete the modifications by changing a single file only—the style sheet.
- **Increased Page Layout Control.** CSS used in conjunction with modern standards-compliant browsers provides a variety of positioning options (even down to the pixel) along with an ability to overlap elements. This gives the Web developer more control over the layout compared to the use of the previously popular XHTML tables.
- **Increased Accessibility.** Pages designed using XHTML tables for layout are easy to view with a traditional browser but can be very tedious when using a screen reader or other assistive technology. By reserving the use of XHTML tables for organizing tabular information and using CSS for page layout—the pages become more accessible.
- **Ability to Define Styles for Multiple Media Types.** Since presentation is separated from content, CSS can be used to set a separate style for printing, or possible use of a screen reader.
- **Support of the Semantic Web.** The Semantic Web is Tim Berners-Lee's vision of the future of the Internet (http://www.w3.org/2001/sw/). According to Berners-Lee, "The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." While much development is being done in this area, Web developers can take small steps, including using XHTML syntax and using CSS to separate styles from structure.

Disadvantages of Using CSS for Page Layout

If you review the screenshots of a site that uses CSS for page layout (Figures 6.1 and 6.2) you'll see a very obvious disadvantage. Visitors using older browsers will not experience your Web site in the same way as visitors using modern browsers. If you are using tables for page layout, this is not an issue. Why then are developers beginning to code mainstream Web sites using CSS for page layout? With good support of CSS by modern browsers and the increasing use of modern browsers, the advantages of using CSS to configure page layout usually outweigh the disadvantages. Of course, the target audience of a Web site should be a deciding factor. For example, if your target audience for an intranet site is a company that has standardized on Netscape 4.7 for the desktop, none of the advantages would be realized and it would be better to design the site using tables for page layout. The projected (and eventually actual) target audience should be considered when deciding on a page layout technique.

Even with the increased CSS support of modern browsers, there are still differences and bugs in their implementation of the W3C Recommendations. This is a disadvantage for Web developers, since coding and testing time is increased. Leading developers have created Web sites that document and discuss these issues (see http://www.quirksmode.org and http://www.positioniseverything.net). The CSS techniques in this chapter have been tested with Internet Explorer 7, Opera 9.21, Firefox 2, and Apple's Safari for Windows beta.

Another potential disadvantage is the fact that experienced Web developers who are adept at coding pages using XHTML tables for layout will see productivity drop temporarily as they learn about CSS techniques and properties. Using CSS positioning is different from configuring pages with tables. Time and practice are needed when learning something new.

At this point you've seen some examples of using CSS for page layout and are aware of the issues related to using this technology. The next section discusses the CSS Box Model—a crucial building block of CSS positioning.

FAQ

Should XHTML tables never be used?

Many commercial Web sites still use XHTML tables for page layout. This is for a very good reason—tables are widely supported by browsers. As a Web developer you will most likely work on sites that use XHTML layout tables and you'll work with these in Chapter 8. However, a growing trend is to configure pages using CSS (sometimes called table-less layout). This does not mean that tables are bad, ineffective, or that they are never coded on Web sites that use CSS for page layout. Even Web sites with so-called "table-less" layouts may include tables to present information in a tabular manner or facilitate design of a small portion of the page.

6.2 The Box Model

Each element in a document is considered to be a rectangular box. As shown in Figure 6.4, this box consists of a content area surrounded by padding, a border, and margins. This is known as the Box Model.

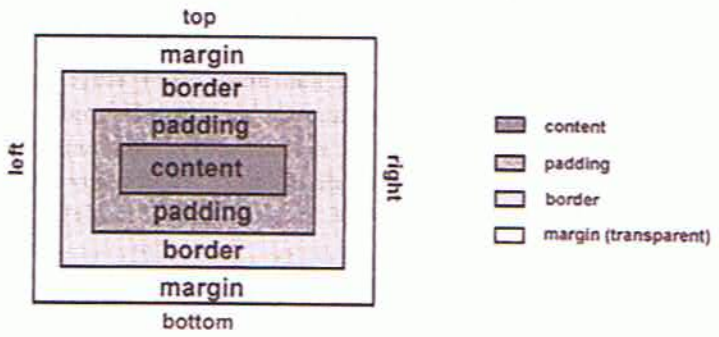


Figure 6.4
The CSS box model

- The **content** area can consist of a combination of text and Web page elements such as images, paragraphs, headings, lists, and so on. The **visible width** of the element on a Web page is the total of the content width, the padding width, and the border width.
- The **padding** area is between the content and the border. The default padding value is zero. When configuring the background of an element, the background is applied to both the padding and the content areas.
- The **border** area is between the padding and the margin. The default border has a value of 0 and does not display. You have already worked with borders in Hands-On Practice 4.2. As shown in Figure 4.2, this area can be set to various styles.
- The **margin** determines the empty space between the element and any adjacent elements. The margin is always transparent. The solid line in Figure 6.4 that contains the margin area does not display on a Web page. Keep in mind that browsers often have default margin values set for the Web page document and for certain elements such as paragraphs, headings, forms, and so on. Use the `margin` property to override the default browser values.

Figures 6.5 and 6.6 display `<div>` elements containing text content. Let's take a closer look. Figure 6.5 shows a screenshot of two `<div>` elements placed one after another on a Web page. In Figure 6.6 the boxes are nested inside each other. In both cases, the

Figure 6.5
Two `<div>` elements displaying the box model

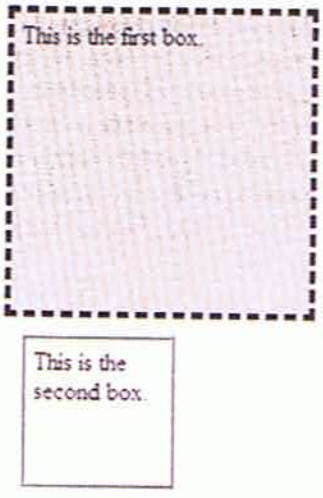
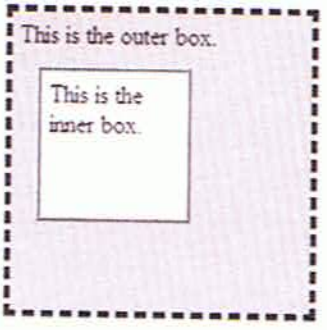


Figure 6.6
Nested elements showing the box model



browser used **normal flow** (the default) and displayed the elements in the order that they appeared in the source code. As you've worked through the exercises in the previous chapters, you created Web pages that the browser rendered using normal flow. You'll practice this a bit more in the next Hands-On Practice, then later in the chapter you'll experiment with positioning to configure the flow, or placement, of elements on a Web page.

HANDS-ON PRACTICE 6.1

You will explore the box model in this Hands-On Practice as you create the Web pages shown in Figure 6.5 and Figure 6.6.

Practice with Normal Flow

Launch a text editor and open the Chapter6/starter1.html file from the student files. Save the file with the name box1.html. This page is displayed in Figure 6.5. Edit the body of the Web page and add the following code to configure two `<div>` elements:

```
<div class="div1">
This is the first box.
</div>
<div class="div2">
This is the second box.
</div>
```

Now let's add the CSS to configure the "boxes." Add a new style rule for a class named `div1` to configure a light gray background, dashed border, width of 200, height of 200, and 5 pixels of padding. The code follows:

```
.div1 {  width: 200px;
         height: 200px;
         background-color: #cccccc;
         border: dashed;
         padding: 5px;
}
```

Create a style rule for a class named `div2` to configure a width and height of 100, ridged border, 10 pixel margin, and 5 pixels of padding. The code follows:

```
.div2 {  width: 100px;
         height: 100px;
         background-color: #ffffff;
         border: ridge;
         padding: 5px;
         margin: 10px;
}
```

Save the file. Launch a browser and test your page. It should look similar to the one shown in Figure 6.5. The student files contain a sample solution at Chapter6/box1.html.

Practice with Normal Flow and Nested Elements

Launch a text editor and open the box1.html file from the student files. Save the file with the name box2.html. This page is displayed in Figure 6.6.

Edit the code. You will not modify the CSS but you will edit the XHTML. Delete the content from the body section of the Web page. Add the following code to configure two `<div>` elements— one nested inside the other.

```
<div class="div1">
This is the outer box.
  <div class="div2">
  This is the inner box.
  </div>
</div>
```

Save the file. Launch a browser and test your page. It should look similar to the one shown in Figure 6.6. Notice how the browser renders the nested `<div>` elements— this is an example of normal flow. The student files contain a sample solution at Chapter6/box2.html.

The examples in the Hands-On Practice happened to use two `<divs>`. However, the box model applies to XHTML elements in general—not just to `<divs>`. You will get more practice using the box model in this chapter. Notice that since the CSS did not use any configurations for positioning, normal flow was used and the second box is nested within the first box because it is coded inside the first `<div>` in the XHTML. Next, we will take a look at some properties that affect positioning: `position`, `float`, `display`, and `z-index`.

6.3 CSS Positioning Properties

You've seen how normal flow causes the browser to render the elements in the order that they appear in the XHTML source code. When using CSS for page layout there are times where you will want to specify the location of an element on the page—either the absolute pixel location, the location relative to where the element would normally display, or floating on the page. There are even times when you will want to modify the way an element displays or cause an element to appear partially or completely over another element. The CSS properties used to accomplish these tasks are discussed next.

Relative and Absolute Positioning

Use **relative positioning** to change the location of an element slightly, relative to where it would otherwise appear in normal flow. Use the `position:relative` property along with either a left, right, and/or top property. The **`left` property** configures the position of the element in relation to the left side of the browser window. The **`right` property** sets the position of the element in relation to the right side of the browser window. The **`top` property** indicates the position of the element in relation to the top of the document area in the browser window.

Figure 6.7 shows a Web page (see the student files, Chapter6/relative.html) that uses relative positioning and the `left` property to configure the placement of a `<div>` (assigned to the `id mycontent`) to the left of the normal flow.

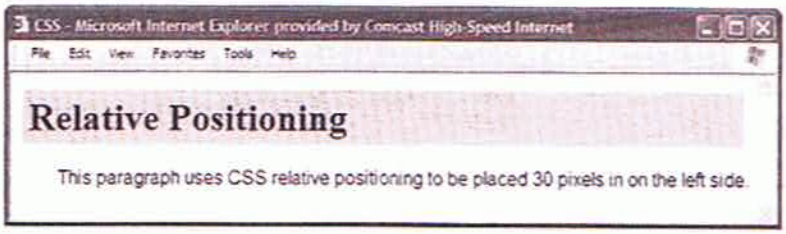


Figure 6.7
The paragraph is configured using relative positioning

The result is that the content of the `<div>`—the paragraph—is rendered 30 pixels in from the left where it would normally be placed at the browser's left margin. W3C Recommendations call for positioning to be applied to any element and Internet Explorer follows this recommendation. However, cross-browser support of positioning is more reliable when the `<div>` element is used for positioning. Notice also how the `padding` and `background-color` properties configure the heading element. The CSS follows:

```
#myContent { position: relative;
             left: 30px;
             font-family: Arial,sans-serif;
}
h1 { background-color: #cccccc;
     padding: 5px;
     color: #000000;
}
```

The XHTML source code follows:

```
<h1>Relative Positioning</h1>
<div id="myContent">
  <p>This paragraph uses CSS relative positioning to be placed 30
pixels in on the left side.</p>
</div>
```

Use **absolute positioning** to specify the location of an element precisely in a browser window. The `position:absolute` property along with either a `left`, `right`, and/or `top` property is needed to configure the placement.

Figure 6.8 shows a screenshot of a Web page that uses absolute positioning to configure a `<div>` (see the student files, Chapter6/absolute.html).

The `<div>` is assigned to the `content id` which is positioned 200 pixels in from the left margin and 100 pixels down from the top of the browser window. The result is that the paragraph contained within the `<div>` is rendered 200 pixels in from the left side and 100 pixels down from the top of the document area in the browser window. The width of the `<div>` is set to 300 pixels. Again, `padding` and `background-color` are used to configure the heading element. The CSS follows:

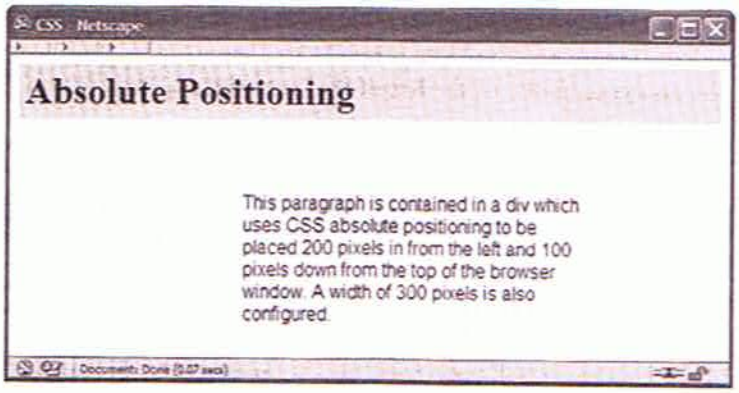


Figure 6.8
The paragraph is configured using absolute positioning

```
#content { position: absolute;
           left: 200px;
           top: 100px;
           font-family: Arial,sans-serif;
           width: 300px;
}
h1 { background-color: #cccccc;
     padding: 5px;
     color: #000000;
}
```

The XHTML source code follows:

```
<h1>Absolute Positioning</h1>
<div id="content">
<p>This paragraph is contained in a div which uses CSS absolute
positioning to be placed 200 pixels in from the left and 100 pixels
down from the top of the browser window. A width of 300 pixels is
also configured.</p>
</div>
```

When working with absolute positioning it is important to be aware that elements not absolutely positioned will be rendered following normal flow by the browser. Elements that are absolutely positioned are rendered outside of normal flow. You'll explore this behavior in the next Hands-On Practice.

HANDS-ON PRACTICE 6.2

Figure 6.9 shows screenshots of two Web pages with similar XHTML content. The Web page in the upper screenshot does not have any CSS applied. The Web page in the lower screenshot uses CSS to configure text, color, and the absolute position of a paragraph element. Launch a text editor and open the Chapter6/starter2.html found in the student files. When a browser renders the page it will use normal flow and display the XHTML elements in the same order as they are coded: `<h1>`, `<div>`, `<p>`, and `<ul>`. Launch a browser and display the page to verify.

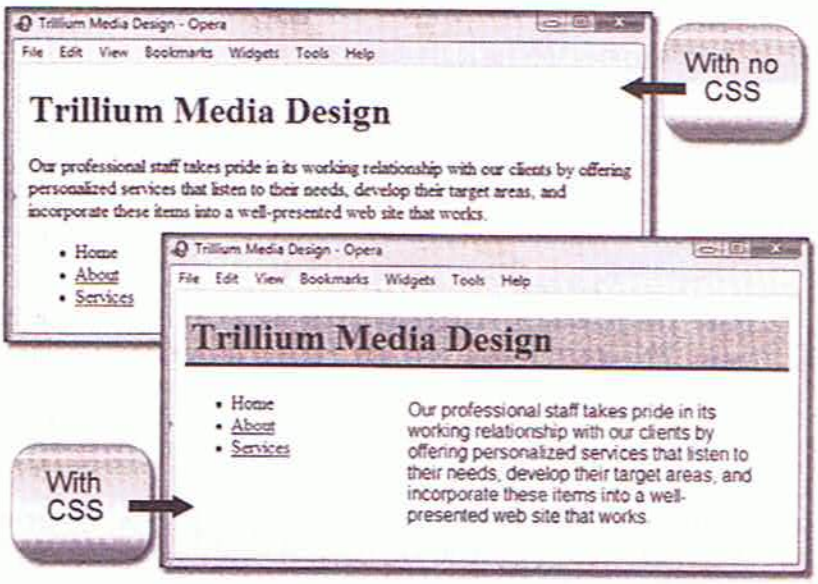


Figure 6.9
The lower Web page uses CSS absolute positioning

Let's add the CSS to make this page more "stylish" and look like the lower screenshot in Figure 6.9. Save the file with the name trillium.html. With trillium.html open in a text editor, modify the code as follows:

1. This page uses embedded styles. Code opening and closing `<style>` tags in the header section.

```
<style type="text/css">
</style>
```

2. Create style rules for the `h1` selector. Configure a background color (`#B0C4DE`), text color (`#000080`), a 3 pixel solid bottom border in the color `#000080`, and 5 pixels of padding on the bottom and left sides.

```
h1 { border-bottom: 3px solid #000080;
     color: #000080;
     background-color: #B0C4DE;
     padding: 0 0 5px 5px;
}
```

Note: The padding can be set for each side individually using the `padding-top`, `padding-right`, `padding-bottom`, and `padding-left` properties. You can use shorthand notation to set all four values in one `padding` property. The order of the numeric values determines which box side is configured (top, right, bottom, left).

3. Create style rules for a class named `content`. Configure the position to be absolute, 200 pixels from the left, 75 pixels from the top, a width of 300 pixels, and Arial or sans serif font typeface.

```
.content { position: absolute;
           left: 200px;
           top: 75px;
           font-family: Arial,sans-serif;
           width: 300px;
}
```

4. Assign the paragraph to the `content` class. Add `class="content"` to the opening paragraph tag in the body of the Web page.

Save the file. Launch a browser and test your page. It should look similar to the lower Web page shown in Figure 6.9. The student files contain a sample solution at Chapter6/trillium.html. Note that even though the unordered list is coded in the page after the paragraph, it's displayed immediately after the heading. This is because the paragraph is absolutely positioned (`position: absolute`). Browsers render absolutely positioned elements *outside* of normal flow.

Note: This Hands-On Practice used embedded CSS for ease of editing. However, for an actual Web site with more than one page the most efficient solution is to use an external CSS file. See Chapter 3 if you'd like to review using external style sheets. You'll use external style sheets later in this chapter.

FAQ **What's a good name for a class?**

A class name should be descriptive of the purpose (such as `nav`, `news`, `footer`, and so on) rather than being descriptive of the presentation (such as `redText`). According to Google's Web Authoring Statistics Study, http://code.google.com/webstats, the 10 most commonly used class names are `footer`, `menu`, `title`, `small`, `text`, `content`, `header`, `nav`, `copyright`, and `button`.

The `float` Property

Elements that seem to float on the right or left side of either the browser window or another element are often configured using the **`float` property**. The browser renders these elements using normal flow, and then shifts them as far as possible within their container (usually either the browser window or a `<div>`) to either the right or left. Other content will flow around the float. To stop this flow, use the **`clear` property**. When floating an image, the `margin` property is useful to configure empty space between the image and text on the page.

Figure 6.10 shows a Web page (see the student files, Chapter6/float.html) with an image that has been configured with `float:right`.

Figure 6.10 The image is configured with `float:right`

Notice how the text in the paragraph wraps around the image. An `id` called `yls` was created that applies the `float`, `margin`, and `border` properties. The attribute `id="yls"` was placed on the image tag. The CSS follows:

```
h1 { background-color: #cccccc;
     padding: 5px;
     color: #000000;
}
p { font-family:Arial,sans-serif;
}
#yls { float: right;
       margin: 0 0 5px 5px;
       border: solid;
}
```

The XHTML source code follows:

```
<img id="yls" src="yls.jpg" alt="Yellow Lady Slipper" height="100"
width="100" />
<h1>Floating an Image</h1>
<p>The heading and paragraph follow normal flow. The Yellow Lady
Slipper pictured on the right is a wildflower. It grows in wooded
areas and blooms in June each year. The Yellow Lady Slipper is a
member of the orchid family.</p>
```

There are times when you want to clear the effect of the float. In these cases you use the `clear` property.

The Web page displayed in Figure 6.11 uses the `clear` property set to `clear:right`. The images each clear the float that precedes them and float to the right of the browser window (see the student files, Chapter6/float2.html).

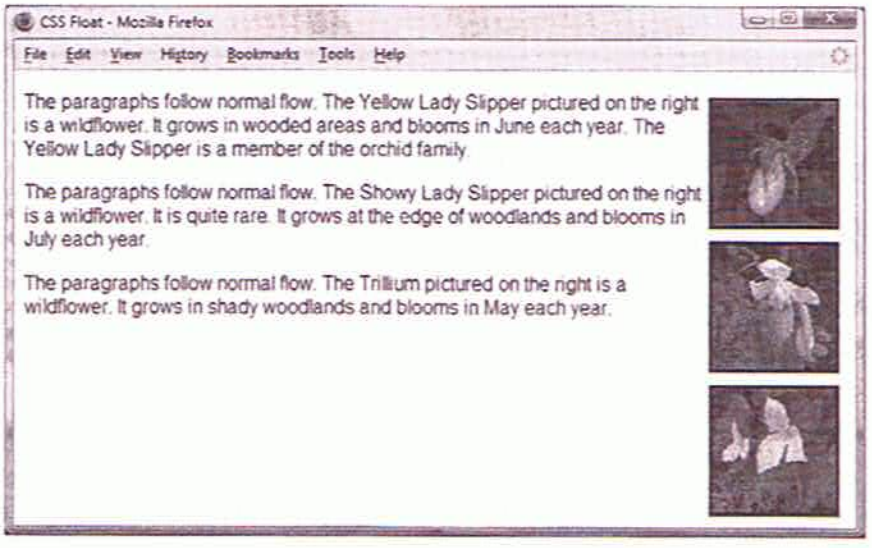


Figure 6.11
Using both the `float` and `clear` properties

The CSS follows:

```
.rightfloat { float: right;
              margin: 5px;
              clear: right;
              border: solid;
}
```

The XHTML source code follows:

```
<img class="rightfloat" src="yls.jpg" alt="Yellow Lady Slipper"
height="100" width="100" />
<p>The paragraphs follow normal flow. The Yellow Lady Slipper
pictured on the right is a wildflower. It grows in wooded areas and
blooms in June each year. The Yellow Lady Slipper is a member of the
orchid family.</p>
<img class="rightfloat" src="pls.jpg" alt="Showy Lady Slipper"
height="100" width="100" />
<p>The paragraphs follow normal flow. The Showy Lady Slipper pictured
on the right is a wildflower. It is quite rare. It grows at the edge
of woodlands and blooms in July each year.</p>
<img class="rightfloat" src="trillium.jpg" alt="Trillium" height="100"
width="100" />
<p>The paragraphs follow normal flow. The Trillium pictured on the
right is a wildflower. It grows in shady woodlands and blooms in May
each year.</p>
```

The `float` and `clear` properties will also be useful when configuring page layouts with multiple columns. In the next Hands-On Practice you'll gain experience using the CSS `float` property.

How do I know when to configure a `class` or an `id`?

If the style can be used on more than one element on a page, configure the style as a `class`. Use the `.` (dot) notation in the style sheet. Use the `class` attribute in the XHTML.

If the style is specific to only one element or if the element will be manipulated using DHTML (see Chapter 11), configure the style as an `id`. Use the `#` notation in the style sheet. Use the `id` attribute in the XHTML.

HANDS-ON PRACTICE 6.3

In this Hands-On Practice you'll practice using the CSS `float` and `clear` properties as you configure the Web page shown in Figure 6.12.

Create a folder named ch6float. Copy the starter3.html and yls.jpg files from the chapter6 folder in the student files into your ch6float folder. Launch a text editor and open the starter3.html file. Notice the order of the images and paragraphs. Notice that there is no CSS configuration for floating the images. Display starter3.html in a browser. The browser renders the page using normal flow and displays the XHTML elements in the order they are coded.

Let's add CSS to float the images and look more similar to Figure 6.12. Save the file with the name floatyls.html. With floatyls.html open in a text editor, modify the code as follows:

1. Add a style rule for a class name `float` that configures `float`, `margin`, and `border` properties.

```
.float { float:left;
         margin-right:10px;
         border:ridge;
}
```

2. Assign the image element to the class named float (use `class="float"`).

Save the file. Launch a browser and test your page. It should look similar to the Web page shown in Figure 6.12. The student files contain a sample solution at Chapter6/floatyls.html.

Figure 6.12
The CSS `float` property left-aligns the image

The `display` Property

Recall from Chapter 2 that some XHTML elements, such as the paragraph and heading elements, are block elements. A division (`<div>`) is also a block element. The browser renders these elements with 100 percent of the available width and displays a line break above and below—forming a "block." Other elements, such as anchor tags and span tags, are rendered directly inline—with no line break before or after them. These are called inline elements.

The **`display` property** configures if and how an element is displayed. An element configured with `display:none` will not be displayed. This is sometimes used when configuring styles to print a Web page. An element configured with `display:block` will be rendered as a block element (even if it is actually an inline element, such as an anchor tag). You will work with the `display` property in Chapter 7.

The `z-index` Property

The **`z-index` property** is used to modify the stacking order of elements on a Web page. When using only XHTML there is no easy way to "stack" elements other than configuring backgrounds for pages or tables. The `z-index` property provides flexibility in the display of elements. The default `z-index` value is `"0"`. Elements with higher `z-index` values will appear stacked on top of elements with lower `z-index` values rendered on the same position of the page. Figure 6.13 is configured using absolute positioning and `z-index` properties.

Figure 6.13
This page uses absolute positioning and `z-index` properties

Notice how the three flower photos and the logo are arranged. It would be difficult to recreate this just using XHTML. This type of page design may be appropriate for the splash page of a Web site. You will recreate this Web page when you complete the next Hands-On Practice . The term splash page originates from client-server applications that display an introductory (or splash) screen while the program loads. Splash pages, sometimes called splash screens, can set the tone or introduce a Web site.

You have been introduced to the `position`, `float`, `display`, and `z-index` properties. For your reference, Table 6.1 contains a list of CSS properties often used with formatting and page layout.

Table 6.1 CSS properties used with formatting and page layout

Property	Description	Commonly Used Values
`background-color`	Background color on an element	Any valid color
`background-image`	Background image on an element	`url(imagename.gif)` or `url(imagename.jpg)`
`background-position`	Position of the background image	Two percentage values or numeric pixel values. The first value configures the horizontal position and the second configures the vertical position starting from the upper-left corner of the container's box. Text values can also be used: `left`, `top`, `center`, `bottom`, `right`.
`background-repeat`	Controls how the background image will repeat	Text values `repeat` (default), `repeat-y` (vertical repeat), `repeat-x` (horizontal repeat), `no-repeat` (no repeat)
`border`	Shorthand notation to configure the `border-width`, `border-style`, and `border-color` of an element	The values for `border-width`, `border-style`, and `border-color` separated by spaces. For example: `border:1px solid #000000;`
`border-color`	Color of the border around an element	Any valid color
`border-style`	Type of border around an element	Text values `double`, `groove`, `inset`, `none` (the default), `outset`, `ridge`, `solid`, `dashed`, `dotted`, `hidden`

Table 6.1 CSS properties used with formatting and page layout (*continued*)

Property	Description	Commonly Used Values
`border-width`	Width of a border around an element	A numeric pixel value (such as `1px`), percentage value, or the text values `thin`, `medium`, `thick`
`clear`	Specifies the display of an element in relation to floating elements	Text values `left`, `right`, `both`, `none` (default)
`color`	Text color	Any valid color
`display`	Controls how and if the element will display	Text values `none`, `block`, `inline`, `list-item`. Display set to `"none"` causes the element not to display.
`font-family`	Name of a font or font family	Any valid font or a font family such as `serif`, `sans-serif`, `fantasy`, `monospace`, or `cursive`
`font-size`	Size of the text font	This varies; `pt` (standard font point sizes), `px` (pixels), the unit `em` (which corresponds to the width of the capital M of the current font), or percentages; the text values `xx-small`, `small`, `medium`, `large`, `x-large`, and `xx-large` are also valid
`font-style`	Style of the font	`normal` (default), `italic`, `oblique`
`font-weight`	Boldness or weight of the font	This varies: the text values `normal`, `bold`, `bolder`, and `lighter` can be used; the numeric values `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, and `900` can be used
`height`	Height of an element	A numeric value (`px` or `em`), numeric percentage, or `auto` (default)
`left`	Distance in from the left to display an element	A numeric pixel value or percentage
`line-height`	Spacing allowed for the line of text	It is most common to use a percentage for this value. For example, a value of `200%` is double space.
`list-style-image`	Image used to replace "bullets" in an XHTML list	`url(imagename.gif)` or `url(imagename.jpg)`
`list-style-type`	Indicates the type of list item marker	Text values `none`, `disc`, `circle`, `square`, `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`
`margin`	Shorthand notation to configure the margin surrounding an element	A numeric value (`px` or `em`) or percentage; for example: `body { margin: 10px;}` will set page margins in the document to 10 pixels. If you set a value to 0 pixels, omit the `px`. Four numeric values (`px` or `em`) can be specified. The values configure the margins in the following order (`margin-top`, `margin-right`, `margin-bottom`, `margin-left`).

continued

Table 6.1 CSS properties used with formatting and page layout (*continued*)

Property	Description	Commonly Used Values
`margin-bottom`	Size of an element's bottom margin	A numeric value (`px` or `em`) or percentage
`margin-left`	Size of an element's left margin	A numeric value (`px` or `em`) or percentage
`margin-right`	Size of an element's right margin	A numeric value (`px` or `em`) or percentage
`margin-top`	Size of an element's top margin	A numeric value (`px` or `em`) or percentage
`min-width`	The minimum width of an element	A numeric value (`px` or `em`) or percentage
`overflow`	Controls the display of a block-level element if the element exceeds its set height or width	Text values `visible`, `hidden`, `auto`, `scroll`
`padding`	Shorthand notation to configure the amount of padding—the blank space between the element and its border	Two numeric values (`px` or `em`) or percentages. The first value configures the top and bottom padding, the second value configures the left and right padding: `padding: 20px 10px;` Four numeric values (`px` or `em`) or percentages. The values configure the padding in the following order: `padding-top`, `padding-right`, `padding-bottom`, `padding-left`.
`padding-bottom`	Blank space between an element and its bottom border	A numeric value (`px` or `em`) or percentage
`padding-left`	Blank space between an element and its left border	A numeric value (`px` or `em`) or percentage
`padding-right`	Blank space between an element and its right border	A numeric value (`px` or `em`) or percentage
`padding-top`	Blank space between an element and its top border	A numeric value (`px` or `em`) or percentage
`position`	Configures the positioning of an element	The value `relative` will position the element in relation to the normal flow. The value `absolute` will position the element at the exact pixel location.
`right`	Distance in from the right to display an element	A numeric pixel value or percentage
`scrollbar-arrow-color`	Color of the arrow on the scroll bar (IE only)	Any valid color
`scrollbar-face-color`	Color of the sliding scroll bar (IE only)	Any valid color
`scrollbar-track-color`	Color of the track the scroll bar slides in (IE only)	Any valid color
`text-align`	The alignment of text	Text values `center`, `justify`, `left`, `right`
`text-decoration`	Determines whether text is underlined; this style is most often applied to hyperlinks	The text value `none` will cause a hyperlink not to be underlined in a browser that normally processes in this manner. The text value `underline` will configure hyperlink to be underlined.