CHAPTER 33

Understanding CSS Positioning Options

In the last few chapters of Part IV, you learned much of what you need to know to use CSS. In this chapter, you learn about positioning options available to you in CSS. This chapter focuses on the five properties that give you complete control over how your elements are positioned on the page. The positioning properties — `display`, `float`, `clear`, `position`, and `z-index` — give you the power to position your text and graphics relatively, absolutely, or using some combination of the two.

In This Chapter

The display property

The float property

The clear property

The position property

The z-index property

Understanding relative positioning

Understanding absolute positioning

Combining relative and absolute positioning

The display Property

The `display` property is one you already take for granted. All elements have a `display` property. You can use the default `display` property or set your own. You can also use the `display` property to make elements appear and disappear on your page. Dynamic HTML, which is just JavaScript and CSS with the Document Object Model, takes advantage of the `display` property to have objects appear and disappear without reloading the page.

There are 17 valid values for the `display` property:

- **block.** All `block` elements have a default display value of block. You can make an `inline` element behave like a `block` element by setting the value of the `display` property to block.
- **inline.** All `inline` elements have a default display value of inline. You can, of course, change this.

- **list-item.** List items are sort of a hybrid between `block` elements and `inline` elements. You probably noticed the chapter that discusses lists beats around the bush on what type of element lists and list items are. A list item is treated as a `block` element with an added list-item marker.
- **compact.** This value creates either block or inline boxes, depending on the context of its use. Properties apply to run-in and compact boxes based on their final status (inline or block level).
- **marker.** This value declares generated content before or after a marker box. This value should only be used with `:before` and `:after` pseudo-elements that are attached to block-level elements. In other cases, this value is interpreted as `inline`.
- **none.** An element with a `display` value set to none does not render on the page. The border, if there is any around the element, is not rendered either. The element doesn't take up any space on the page; thus, it doesn't affect the layout of any other elements. Finally, any child elements of the element whose `display` value is set to none are not rendered (they inherit this `display` value), *even if* their own `display` values are explicitly set to something other than none.

Chapter 48, which covers JavaScript, explains why you might set the `display` property to none.

- **table, inline-table, table-row-group, table-column, table-column-group, table-header-group, table-footer-group, table-row, table-cell, and table-caption.** These values cause an element to behave like a table element (subject to table restrictions).
- **run-in.** No elements have a default value of run-in. Setting the display value of an element to run-in has this effect: If the following element is not of type block, is floating, or is positioned absolutely, the run-in element renders as a `block` element. Otherwise, the run-in element renders as if it were part of the following element, which means inline with the following element. This doesn't work in Version 5 or earlier browsers.

```
H6.run-in {

        display: run-in;
        font-variant: bold;
        }
H6:after.run-in {
        content: ". ";
        }
```

```
<H6 class="run-in">Run-in.</H6>

<P>No elements have a default value of run-in. Setting the
display value of an element to run-in has the effect you see
in this example.</P>
```

The previous example, using the style sheet rules immediately preceding it, renders as follows:

Run-in. No elements have a default value of run-in. Setting the display value of an element to run-in has the effect you see in this example.

The float Property

It is not unusual to want to arrange your page so that one or more elements, most commonly images, appear next to (rather than above or below) other elements. The following HTML and CSS create the effect you see in Figure 33-1:

```
IMG.icon {
       padding: 20px;
       float: left;
}

<HTML>
<HEAD>
  <LINK rel="stylesheet" type="text/css" href="ch33.css">
  <TITLE>Chapter 33 Examples</TITLE>
</HEAD>

<BODY>
<H1>Chapter 33 Examples</H1>

<IMG class="icon" src="fir.gif">

<P>It is common to want to have images appear on the same line
as your text. In this example, the float property of the IMG
class icon is set to "left" to achieve this effect. More text
is required in this example so that you can see that the text
continues to wrap around the image. This is that additional
text that is required. By now, the example should have enough
text to make the point. Just in case, another sentence will
provide additional text for purposes of making this point. This
will be the last sentence. OK, maybe just one more, but I mean
it this time.</P>

</BODY>
</HTML>
```

The `float` property has three values: left, right, and none. Setting the value of the `float` property to none (the default value) results in the element not floating at all. In the previous example, the page would instead be rendered as seen in Figure 33-2.

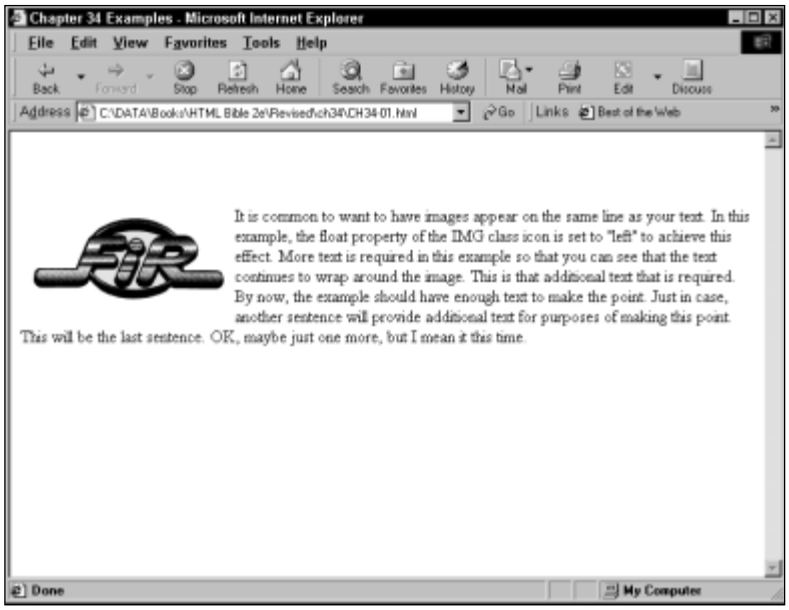

Figure 33-1: Using the float property to wrap text around an image

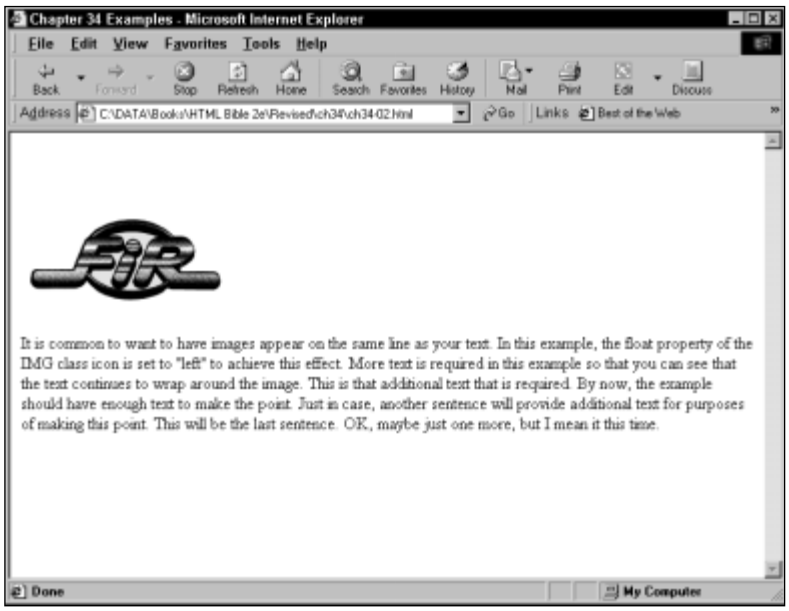

Figure 33-2: Setting the float property to none results in no floating

The clear Property

The `clear` property works in conjunction with the `float` property. While the `float` property is applied to the object you want to wrap around, the `clear` property is applied to the object you want to do the wrapping. The default value for the `clear` property is *none*. This means both the right and the left sides of the element are allowed to have a floating object next to them.

In the following example, there are two icon classes. One positions an image as floating to the left of the nonfloating element. The other positions an image as floating to the right of the nonfloating element: in this case, the `P` element. Notice that both the images must appear *before* the nonfloating element. If they don't, the nonfloating element only wraps around the first image and the second image appears on a line by itself after the nonfloating element.

```
IMG.icon {
       padding: 20px;
       float: left;
}
IMG.icon2 {
       padding: 20px;
       float: right;
}
P {
       clear: none;
}

<BODY>
<H1>Chapter 33 Examples</H1>
<IMG class="icon" src="../public_html/itclogo.gif">
<IMG class="icon2" src="../public_html/itclogo.gif">

<P>It is common to want to have images appear on the same line
as your text. In this example, the float property of the IMG
class icon is set to "left" to achieve this effect. More text
is required in this example so that you can see that the text
continues to wrap around the image. This is that additional
text that is required. By now, the example should have enough
text to make the point. Just in case, another sentence will
provide additional text for purposes of making this point. This
will be the last sentence. OK, maybe just one more, but I mean
it this time.</P>

</BODY>
```

The results of the preceding example can be seen in Figures 33-3 and 33-4. The same example is shown in both Netscape and Internet Explorer so you can see that both browsers render the same thing slightly differently. Specifically, Internet Explorer seems not to respect the padding property. Netscape graduates the text around the image.

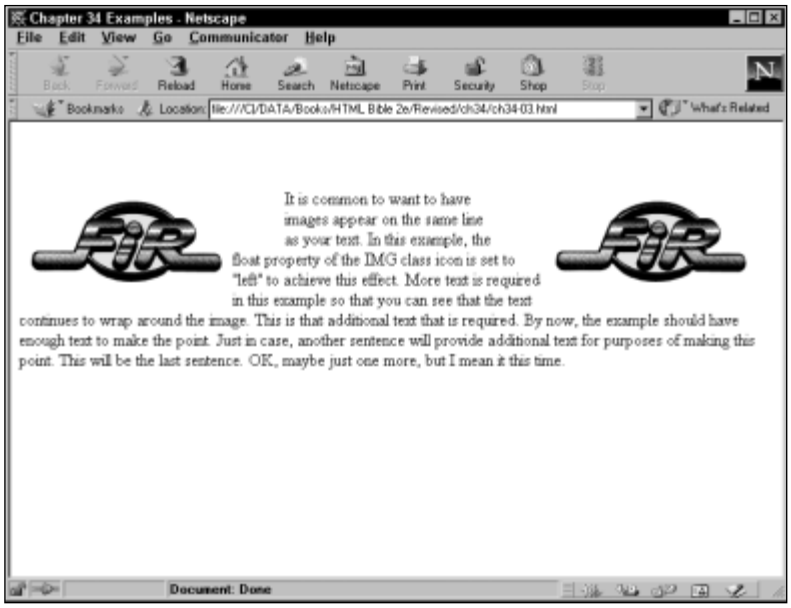

Figure 33-3: Using the float and the clear properties together (in Netscape)

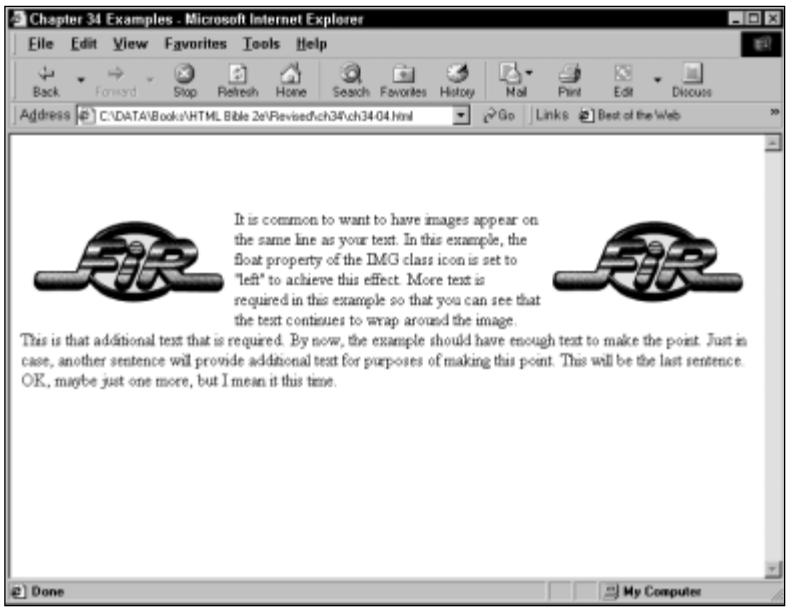

Figure 33-4: Using the float and the clear properties together (in Internet Explorer)

In Figure 33-5, the `clear` property of the nonfloating element (`P`) is set to left. This prevents anything from floating to the left of the `P` element. In this example, however, because both images (being the same image) are the same height, the second image (with a class of icon2) floats to the right on the page, parallel to the first image.

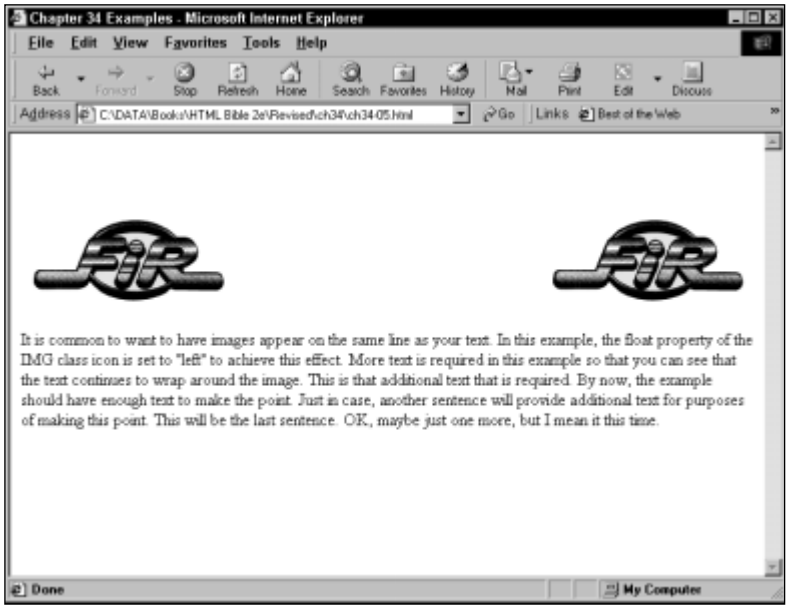

Figure 33-5: Floating with the clear property set to left

The results in Figure 33-5 are not exactly what were intended. If what was wanted was for the second image to float to the right of the text and the first image to be positioned above the text, then the `float` property of the first image must also be turned off (set to none, commented out, or deleted from the style sheet). The resultant style sheet would look like this:

```
IMG.icon {
       padding: 20px;
       /* float: left; */
}

IMG.icon2 {
       padding: 20px;
       float: right;
}

P {
       clear: left;
}
```

And the results of this style sheet, with the HTML previously given, can be seen in Figure 33-6.

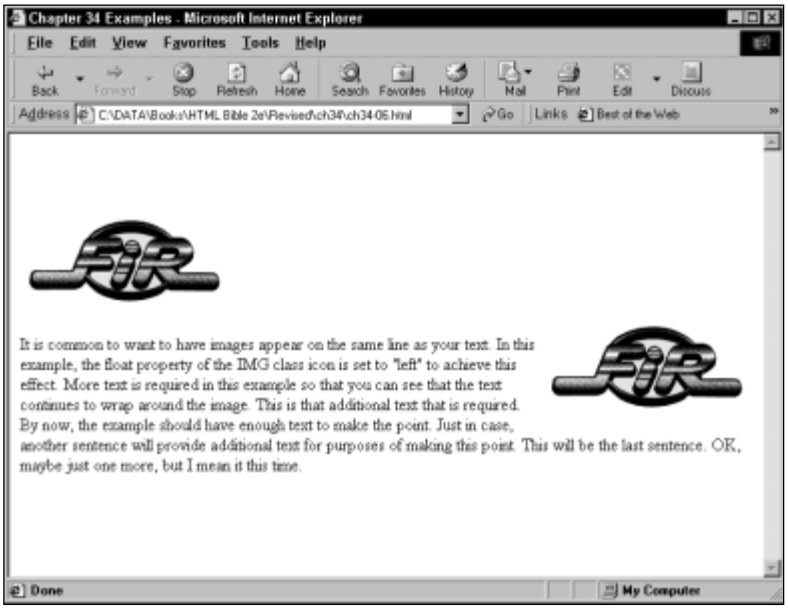

Figure 33-6: Using the float and clear properties properly to achieve desired results

The position Property

The fourth of the five properties you can use to control positioning with CSS is the `position` property. The `position` property takes one of three values:

- **normal.** This is the default value. Normal position simply means the rules you have learned thus far about CSS apply to your element, and it renders in a position based on the values you assign to it and the values of the previous normal elements.
- **relative.** This means relative to the position the element would have if it were defined as a statically positioned element. All children elements are positioned with this new relative position as their starting point. When positioning elements with the relative position, overlaying or obscuring other elements is possible.
- **absolute.** When you define an element as absolutely positioned, it is not included in the calculations used to position other elements on the page. The element is simply put where you tell it to be put. All children elements of this element are positioned relatively to this absolute position. When using absolute positioning, it is possible to overlay or obscure other elements.

The z-index Property

The `z-index` property is a wonderful addition to CSS. It enables you to position elements in the third dimension: depth.

Why would you want to position elements with the `z-index` property?

- To layer graphics (each of which are links), to save space, or to create an interesting visual effect
- To layer text over an image
- To hide graphics for use later, using JavaScript, when you can simply change the `z-index` of an object and have it magically appear

Understanding Relative Positioning

In CSS, relative positioning has two possible meanings. Most Web designers think of relative positioning as when they position an element using the float property (formerly the `valign` and `align` attributes) or by changing the margins or padding on an element to scoot it one way or another. Generally speaking, *relative positioning* is positioning the element using any means other than absolute positioning. All the properties you have learned about CSS in this chapter and in previous chapters have been about relative positioning, which CSS2 now confusingly calls *static positioning.*

To make matters more interesting, CSS introduced the `position` property, which you can set to *relative*. This enables you to move an element by some offset from the position at which it would have been positioned if you had allowed the browser to position it using all the previous properties you defined for it and for preceding elements.

To unconvolute the previous sentence, these are the plain-English steps the browser goes through to position an element not defined as absolutely positioned:

1. Calculate the internal size of the box associated with each element
2. Calculate the size of each element's box, including padding, borders, and margins
3. Calculate the position of each element's box on the page based on the amount of space taken up by each previous box

If you choose to set the value of the position property to *relative*, then the browser takes a fourth step to calculate the element's position. After calculating the starting point for that box, the browser uses the relative offset you have indicated to calculate a new position, relative to that starting point.

Understanding Absolute Positioning

When you position an element with absolute positioning, the browser only takes the first two steps previously listed. After it has calculated the size of the box, it renders that box exactly where you tell it. You risk overlaying other elements, if you are not careful, but it can be used to create stunning layout effects.

Combining Relative and Absolute Positioning

You can combine relative and absolute positioning in CSS, but you must do so thoughtfully or you will end up with your text running over your graphics, or your graphics overlaying your text.

The easiest way to combine relative and absolute positioning is to define the first element on your page — say, an image of known dimensions — as absolutely positioned near the top-left of the page, or wherever you want it positioned, and then to define the rest of your page as being relative to that offset (the dimensions of the absolutely positioned element).

Other options for combining relative and absolute positioning include positioning an element to the right or left of the body of your page and limiting the width of the rest of the elements in your page by using the width attribute.

You learn more specific applications of both relative and absolute positioning in Chapter 35.

From Here

Jump to Chapter 47 and learn about the Document Object Model (DOM), which is essential to dynamic HTML.

Proceed to Chapter 34 to learn about positioning and sizing graphics.

Summary

In this chapter you learned advanced CSS. You learned the five essential layout properties: `display`, `float`, `clear`, `position`, and `z-index`. You learned how relative positioning differs from static positioning and how they both differ from absolute positioning. You also learned how to combine relative and absolute positioning.

✦ ✦ ✦