because the z-index settings within each element are relative only to the other descendants of that element. In effect, the strong element in Paragraph 2 shares the z-index value of its parent in relation to its parent's siblings.

Absolute Positioning

There have been examples of absolute positioning throughout this chapter, but this section examines this popular method of positioning in more detail.

An absolutely positioned element has these basic characteristics:

- It is declared using {position: absolute;}.
- It is positioned relative to the edges of its containing block using one or more of the offset properties (top, right, bottom, left). Properties that are not specified are set to auto (the default). The offset values apply to the outer edge of the element box (including the margin value, if there is one).
- It is completely removed from the document flow. The space it would have occupied in the normal flow is closed up and it no longer has an affect on other elements (for instance, text won't wrap around it).

These points are demonstrated in this simple example of an absolutely positioned list element (Figure 21-15).

```
div {position: absolute; background-color: #999; width: 440px;}
ul {position: absolute; left: 60px; top: 30px; background-color: #CCC;
margin: 0px;}

<div>
  <p>Phasellus feugiat eros at mi. Integer leo tellus, hendrerit non,
euismod non, condimentum in, sem. </p>
   <ul>
       <li>Lorem ipsum dolor</li>
       <li>Sit amet, consectetuer</li>
      <li>Adipiscing elit</li>
      <li>Vel nonummy ligula</li>
      <li>Tempus dignissim</li>
   </ul>
  <p>Fusce suscipit, ligula eget tempus ...</p>
</div>
```

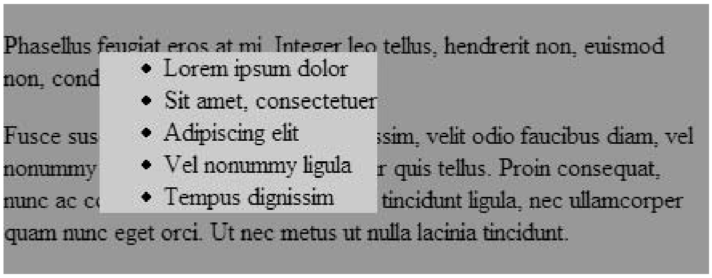

Phasellus feugiat eros at mi. Integer leo tellus, hendrerit non, euismod
non, cond
Lorem ipsum dolor
Sit amet, consectetuer
Adipiscing elit
Vel nonummy ligula
Tempus dignissim
Fusce sus ssim, velit odio faucibus diam, vel
nonummy r quis tellus. Proin consequat,
nunc ac c tincidunt ligula, nec ullamcorper
quam nunc eget orci. Ut nec metus ut nulla lacinia tincidunt.

Figure 21-15. Absolute positioning

In all of the previous examples, elements have been positioned using length measurements for the offset property values. The auto value has some interesting behavior that bears attention. When any of the offset properties other than bottom are set to auto, the edge of the element box is positioned in its “static” position, that is, where it would have been in the normal document flow. In Figure 21-16, the dollar sign slug will always stay next to its line of origin, because its top offset property is set to auto.

```
p {position: relative; margin-right: 10px; left: 10px;"}

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque
<span style="position: absolute; top: auto; left: -1em; background-color:
#CCC;">$</span>pharetra, urna in laoreet tincidunt,...</p>
```

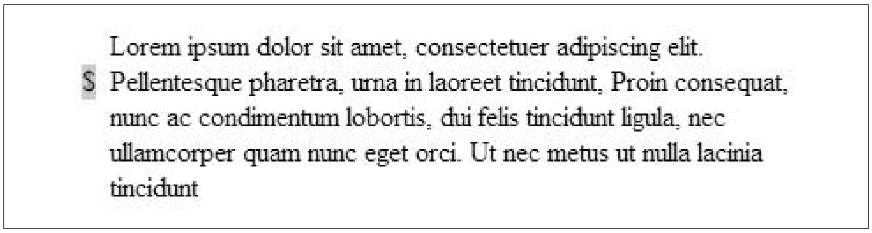

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
$ Pellentesque pharetra, urna in laoreet tincidunt, Proin consequat, nunc ac condimentum lobortis, dui felis tincidunt ligula, nec ullamcorper quam nunc eget orci. Ut nec metus ut nulla lacinia tincidunt

Figure 21-16. Setting offset properties to auto

Notice that the top of the positioned element is in the vertical position that it would have had if the element were still in the line. Only its horizontal position has been changed, as specified. Notice also that the space that the element occupied on the line has been closed up because it has been absolutely positioned. If the left offset property had been set to auto as well, the left edge of the element would be placed in the spot at which the content originated, but it would overlap with the following text (because its space is closed up).

This can be a useful method for adding margin notes that stay with their respective text. Just be sure that there are few or no constraints on the other positioning and sizing properties that might override the auto placement.

Absolute Positioning and Containing Blocks

The first step to absolutely positioning an element is to identify or create its containing block. The containing block is critical to positioning because all absolute measurements are based on its sides. Containing blocks were discussed in more detail earlier in this chapter, but it’s worth a brief refresher.

For an ancestor element to be a containing block, it must have a position value of absolute, relative, or fixed (in other words, it must not be static, either declared or by default). If no ancestor element qualifies as a containing block, then the initial containing block is used (html, body, or the viewport, as determined by the user agent).

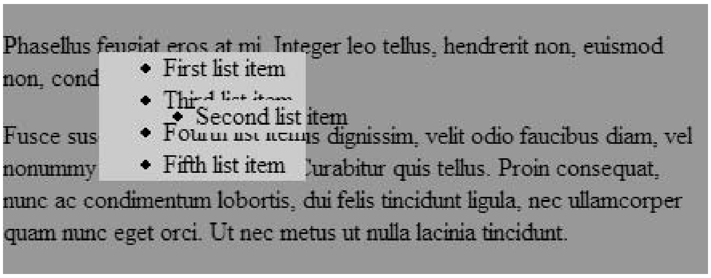
In the example in Figure 21-15, the containing block for the list is a div that has its position set to relative (but its position has not been altered). It is common practice to declare the position of an ancestor element as relative explicitly and leave it in place, or to insert a new positioned element (like a div) to set up the containing block for absolutely positioned elements.

> To force the browser to use the body element as the initial containing block, add this style rule:
>
> ```
> body {position: relative;}
> ```

Another important thing to note is that by setting the position of the unordered list element (ul) to absolute, it thereby becomes the containing block for its descendant elements. If an li element were to be absolutely positioned, its offset properties become relative to the sides of the ul, as shown here and in Figure 21-17.

```
div {position: absolute; background-color: #999; width: 440px;}
ul {position: absolute; left: 60px; top: 30px; background-color: #CCC;
margin: 0px;}
li#callout {position: absolute; left: 60px; top: 30px; background-color:
#CCC; margin: 0px;}

<div>
  <p>Phasellus feugiat eros at mi. Integer leo tellus, hendrerit non,
euismod non, condimentum in, sem. </p>
   <ul>
       <li>First list item</li>
       <li id="callout">Second list item</li>
      <li>Third list item</li>
      <li>Fourth list item</li>
      <li>Fifth list item</li>
   </ul>
  <p>Fusce suscipit, ligula eget ...<p>
</div>
```


Phasellus feugiat eros at mi. Integer leo tellus, hendrerit non, euismod non, cond

- First list item
- Thir
- Second list item
- Fo
- Fifth list item

Fusce sus is dignissim, velit odio faucibus diam, vel nonummy urabitur quis tellus. Proin consequat, nunc ac condimentum lobortis, dui felis tincidunt ligula, nec ullamcorper quam nunc eget orci. Ut nec metus ut nulla lacinia tincidunt.

Figure 21-17. The absolutely positioned list becomes the containing block for the positioned list item

Calculating Position

While specifying a position using the offset properties is a fairly straightforward affair, things can get complicated when offset measurements are combined with the margins and content width of the element and the width constraints of the containing block. In fact, the CSS 2.1 specification provides a dizzyingly detailed list of rules and constraints for dealing with conflicting and unspecified values.

In the interest of brevity, this section provides a general and practical summary of those rules that should serve you well in most instances.

The CSS 2.1 specification provides a formula for all the values that make up the width of a containing block. It is presented in Figure 21-18 in graphical form because it is helpful to visualize the values that span across a containing block. Bear in mind that the calculated sum of all the interior values must be equal to the width of the containing block. This same structure applies in the vertical direction as well.

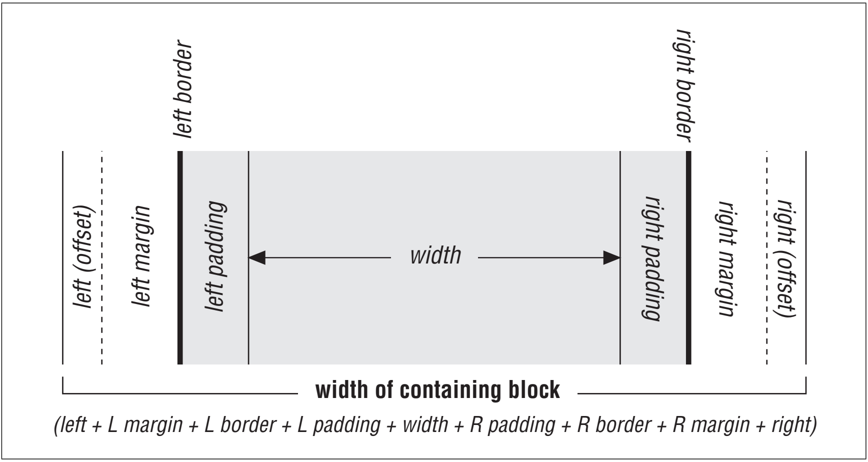

left border
right border
left (offset)
left margin
left padding
width
right padding
right margin
right (offset)
width of containing block
(left + L margin + L border + L padding + width + R padding + R border + R margin + right)

Figure 21-18. The sum of values in the containing block

In very generalized terms, when values are conflicting or unspecified, the space tends to be adjusted on the right side for left-to-right (ltr) languages (or the left side for right-to-left languages). Height issues are resolved by adjusting the space at the bottom of the positioned element.

- In instances where all values have been specified (i.e., none of them are `auto`), and the values do not add up to the width of the containing block, user agents are instructed to just ignore the right (for `ltr` languages) and bottom offset values and make up the discrepancies on those sides.
- If the width of the positioned element is specified, it is not altered. However, if the width for a text (non-replaced) element is set to `auto`, the content area will “shrink to fit” and be just wide enough to accommodate the contents. For replaced elements such as images, the inherent pixel dimensions of the object are used when `width` is `auto`.
- The width of an element’s content area gets resized only when it is set to `auto` and all the other properties have specific measurement values. As the only parameter set to `auto`, the element `width` is the last resort and gets resized.

- User agents look for an auto value (on the margin or offset) on the right side first (for ltr languages) to make necessary space adjustments. For vertical adjustments, adjustments are made to properties set to auto on the bottom.
- When the top and left properties are set to auto, the element is placed in its "static" position (as mentioned above). This is overridden only as a last resort when all of the other parameters have specific values and left (for horizontal placement) and top (for vertical placement) are the only available auto values. Only then is space adjusted on those sides.

Given these constraints and behaviors, the most simple and predictable approach to absolute positioning is to provide a specific width for the positioned element and specific top and left offsets. That way, the margins on the positioned object will be preserved and the space on the right and bottom can flex as necessary to fit in the containing block. Granted, this won't work for all situations, but it's a starting point. It usually involves a bit of math to get it right.

These positioning rules are based on the correct behavior as defined in the CSS 2.1 spec and describe the basic behavior of standards-compliant browsers. Be aware, however, that because of a problem with the box model implementation in Internet Explorer for Windows (all versions except IE 6 and 7 running in Standards mode), these browsers have a different method of calculating position based on applying the padding, borders, and margin within the specified width.

Fixed Positioning

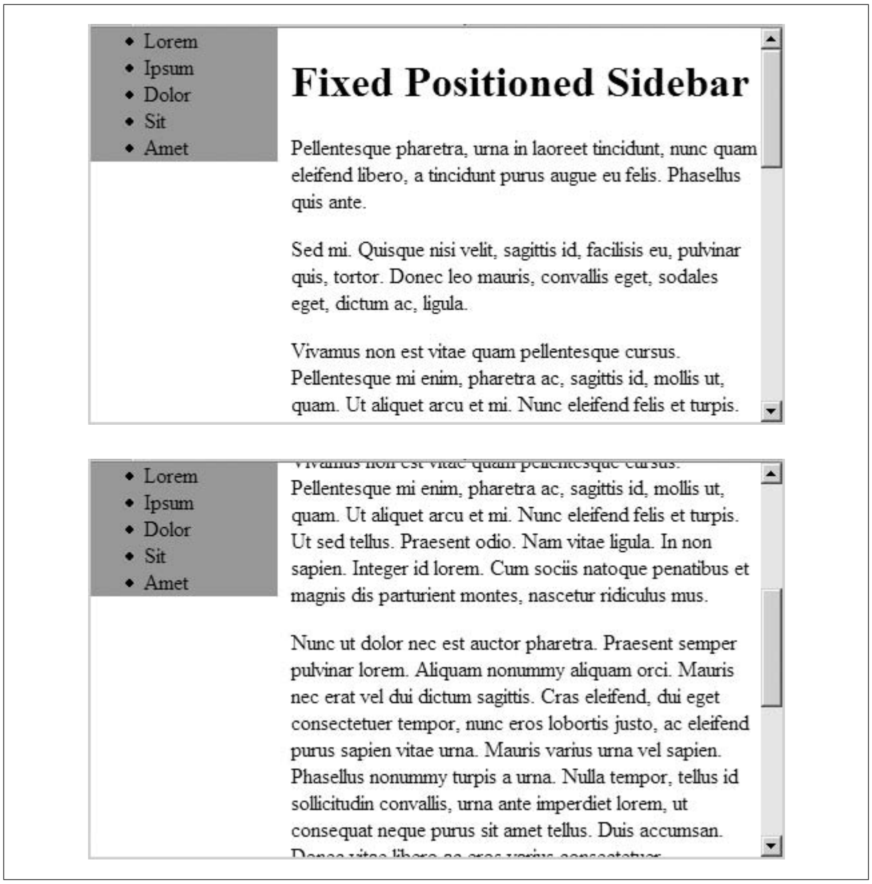
Fixed positioning is essentially the same as absolute positioning, only the containing block is the viewing area (or viewport; typically the browser window). The distinguishing feature of fixed elements is that they do not scroll with the document, but are persistent on the page. On printed pages, fixed elements may appear in the same place on all pages.

In addition to not scrolling, fixed elements share these basic characteristics:

- They are declared using {position: fixed;}.
- They are positioned relative to the edges of the viewport (browser window) using one or more of the offset properties (top, right, bottom, left). Properties that are not specified are set to auto (the default). The offset values apply to the outer edge of the element box (including the margin value, if there is one).
- Like absolutely positioned elements, they are completely removed from the document flow, and the space they would have occupied is closed up.

Fixed elements can be used to create frame-like interfaces or to place persistent elements on the page. In this example, a fixed element is used as a short sidebar that stays put as the document scrolls (Figure 21-19).

```
ul {position: fixed; top: 0px; left: 0; width: 100px; background-color:
#999; margin: 0; padding: 10;}
p, h1 {margin-left: 150px;}
```



- Lorem
- Ipsum
- Dolor
- Sit
- Amet

Fixed Positioned Sidebar

Pellentesque pharetra, urna in laoreet tincidunt, nunc quam eleifend libero, a tincidunt purus augue eu felis. Phasellus quis ante.

Sed mi. Quisque nisi velit, sagittis id, facilisis eu, pulvinar quis, tortor. Donec leo mauris, convallis eget, sodales eget, dictum ac, ligula.

Vivamus non est vitae quam pellentesque cursus. Pellentesque mi enim, pharetra ac, sagittis id, mollis ut, quam. Ut aliquet arcu et mi. Nunc eleifend felis et turpis.

- Lorem
- Ipsum
- Dolor
- Sit
- Amet

Vivamus non est vitae quam pellentesque cursus. Pellentesque mi enim, pharetra ac, sagittis id, mollis ut, quam. Ut aliquet arcu et mi. Nunc eleifend felis et turpis. Ut sed tellus. Praesent odio. Nam vitae ligula. In non sapien. Integer id lorem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Nunc ut dolor nec est auctor pharetra. Praesent semper pulvinar lorem. Aliquam nonummy aliquam orci. Mauris nec erat vel dui dictum sagittis. Cras eleifend, dui eget consectetuer tempor, nunc eros lobortis justo, ac eleifend purus sapien vitae urna. Mauris varius urna vel sapien. Phasellus nonummy turpis a urna. Nulla tempor, tellus id sollicitudin convallis, urna ante imperdiet lorem, ut consequat neque purus sit amet tellus. Duis accumsan. Donec vitae libero ac eros varius consectetuer.

Figure 21-19. Fixed positioning

Internet Explorer 6 and earlier for Windows does not support fixed positioning. Objects with fixed positioning are treated as though they are static, and therefore behave as though they have not been positioned at all. There are workarounds available; to find them, search for "CSS fixed position in IE" or something similar in your favorite search engine. Support in IE 7 for Windows (in beta as of this writing) has not been confirmed.

Relative Positioning

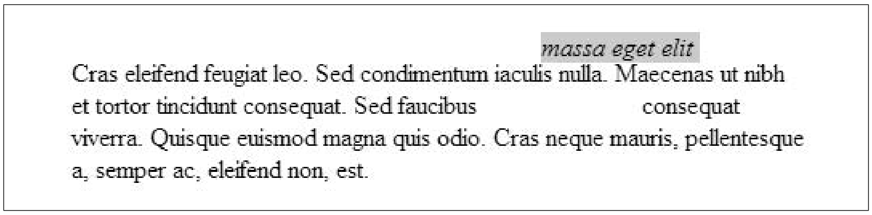
Relative positioning works differently than absolute and fixed positioning. The critical difference is that although the element is moved around, the space where it would have appeared in the normal flow is preserved and continues to influence the elements that surround it.

Relatively positioned elements have these characteristics:

- They are declared using `{position: relative;}`.
- They are positioned relative to their initial position in the normal flow using one or more of the offset properties (`top`, `right`, `bottom`, `left`). Properties that are not specified are set to `auto` (the default).
- Their original space in the document flow is preserved.
- Because they are positioned elements, they can potentially overlap other elements.

This example of a relatively positioned emphasized (`em`) element demonstrates the basic syntax and behavior of relative positioning (Figure 21-20). Notice that when the element is moved, its space is left behind and the surrounding elements behave as though it is still there.

```
em {position: relative; top: -36px; right: -36px; background: #ccc; }
```



Cras eleifend feugiat leo. Sed condimentum iaculis nulla. Maecenas ut nibh et tortor tincidunt consequat. Sed faucibus *massa eget elit* consequat viverra. Quisque euismod magna quis odio. Cras neque mauris, pellentesque a, semper ac, eleifend non, est.

Figure 21-20. Relative positioning

In relative positioning, the `top`, `right`, `bottom`, and `left` properties move the element relative to its original position. Specifying a positive value for `top` moves the element down by that amount. Specifying a value for `left` moves the element to the right, and so on, such that a positive value for one side is equivalent to a negative value on the opposite side (the computed values are `right=-left` and `bottom=-top`).

The CSS 2.1 specification advises that when conflicting values are provided, the provided value for `right` is ignored in left-to-right languages (`left` is ignored for right-to-left languages) and is understood to be `-left`. When top and bottom values conflict, the provided `bottom` value is ignored and reset to `-top`. As such, this overconstrained style rule:

```
em {top: 10; bottom: 50; left: 50: right -4;}
```

would be rendered as though it had specified like this:

```
em {top: 10: bottom: -10; left: 50; right: -50;}
```

Relative positioning is often used to establish a containing block by specifying the position of the element as `relative`, but not altering its position. The result is that its child elements can then be absolutely positioned relative to the rectangle created by the element.