CSS Layout

by **Ross Shannon**

One of the major benefits of using CSS is that you're no longer forced to lay your sites out in **tables**. The layout possibilities of CSS give you complete control over the positions and dimensions of all page elements.

If you've tried laying out a page with tables, you have probably been irritated in the past by the inability of your browser to render your page exactly as you had wanted. Table structures aren't the most flexible of page layout devices, as they weren't really designed for this purpose. Now however, with some reliable browser support in the current generation of browsers, you have a new and much improved option.

This page was last updated on 2009-04-27

Working with divs

The element is well-suited to take over from tables as a layout tool. It is a **block-level** element that is used to *div*ide the page into logical sections, and can hold whatever you need inside it. You can have **blocks** of text in divs and then put them together in a layout. You have immense freedom, with the ability to add these blocks, or "layers", on top of each other. Check out this **example**.

The div tag has few attributes of its own (save for align="left | right | center"), with **all of its formatting applied through stylesheets**. To set up a simple navigation block, we would use code like this (with the CSS being in an **external .css file** or style block):

```
div#navigation {width: 200px; background: gray; padding: 10px; }

<div id="navigation">...navigation links...</div>
```

This example code uses some very simple CSS code. All block-level elements can have a width property, specified in units or as a percentage, and then I just added a **background colour** and some **padding** space around the div content.

Floating Elements

Since divisions are block-level (i.e., they default to 100% of the available screen width and add line breaks between each other), they will all just stack up underneath one another unless you position them in some way. The simplest way to do this is to use the CSS float property, the backbone of most CSS layouts. You can float any element left or right, and it will align itself over to the side of whatever element it is contained within.

```
#column1 {float: left; width: 200px; padding: 10px; }
#column2 {float: left; width: 200px; padding: 20px; }
```

To create **columned layouts, you simply float all of the column divisions to the same side** and they will line up beside each other, as long as they fit. Laying out content in this way has immediate benefits such as *progressive downloading* (as the text is loaded it is displayed onto the page immediately, so your visitor can read as the page is forming around the text). You can also give each column specific **margins and padding**, giving you greater freedom to space your content out. Below is an example of code like the CSS above, with both div elements given the float: left; property:

Column 1 Column 2

With these floating elements you can mimic a table structure, and have your page in a traditional layout without all the drawbacks of tables. Floating elements takes a little bit of practice (especially if the columns are not the same height), but can result in many traditional and non-traditional layouts. But CSS wasn't content to merely emulate the layout mechanisms of the past — now you can control the position of elements on the page down to the pixel.

CSS Positioning

There are two other types of positioning beyond floating: **absolute** and **relative**. The codes you'll be using are

```
tag {position : choice; top : 0px; bottom : 0px; left : 0px; right : 0px; }
```

Browser Compatibility Note:

Absolute and relative positioning is a feature of the CSS2 specification and so is supported by **» Internet Explorer 4+**, **» Mozilla**, **» Firefox**, **» Opera** and **» Safari**. For best results use the newest browsers available, as they will have improved and more accurate rendering capabilities. Do not use these if your users may be using older browsers. Netscape 4.7's

positioning support is full of glitches, and is largely crap (though better than the horrendous mess it makes of floated elements...).

Absolute Positioning

If you position an element (an image, a table, or whatever) absolutely on your page, it will appear at the exact pixel you specify. Say I wanted a graphic to appear 46 pixels from the top of the page and 80 pixels in from the right, I could do it. The CSS code you'll need to add into the image is

```
img {position: absolute ; top: 46px; right: 80px; }
```

You just add in which method of positioning you're using at the start, and then push the image out from the sides it's going to be closest to. You can add the CSS directly into the tag using the style attribute (as shown in the **introduction to stylesheets**), or you can use **classes and ids** and put them into your stylesheet. It works the same way. The recommended method is to add classes for layout elements that will appear on every page, but put the code inline for once-off things.

The image would appear **like so**. As you can see, it is possible to have things **overlapping** with absolute positioning.

Positioning Layers

To create what we call **layers** with the div tag, use code like this:

```
<div style="position: absolute; left: 610px; top: 80px; height: 400px; width: 100px; padding: 1em;">layer stuff</div>
```

First you specify the **position** of the layer, then its **dimensions** (which is optional, the layer will resize itself). If you want to give colour to the layer's background, add the background-color: red; attribute in with the rest of your CSS code. As usual, you can use **websafe colours**, or **named colours**.

Anything that could go on a normal page can be positioned with divs. They **load far faster** than, say, a table layout. Tables do not display on-screen until they have been downloaded in their entirety. With layers, all the information a browser needs is contained in the style attributes you've added. Therefore, it will display as soon as any part of it is downloaded.

To get layers **overlapping** each other you need to use the z-index command. Add z-index: 1 in with the positioning code and this element will appear above everything without this command. If you want something else to go over this layer too, add z-index: 2 and so on. The higher the index, the closer the div will appear to the front of the page.

Put the layer that holds your page's **content at the top** of your code. This is what readers want to see immediately. Your navigation and other presentational components can then load *around* this, allowing your reader to begin reading as soon as possible and making your navigation available when it is most likely to be used: after the page has been read.

*To see some examples of designs laid out with both float and absolute positioning, have a look-see at my **redesigns** section.*

Relative Positioning

When you position something relatively, you are modifying its position from where it *would* have been if you hadn't changed anything. I find that to be the easiest way of thinking about it. For instance, in the next sentence, I'll offset some words 12px down and 22px right relative to their start position.

Well, here are (some words)
some words

The words in parentheses are the words in their original positions, and the bold ones are the moved words. The CSS code to move them was

```
<span style="position: relative ; top: 12px; left: 22px;">some words</span>
```

You should notice that if you want to move something left, you use the right code, and push it away from that side and vice-versa.

To override an inherited position property, and make the element just a normal part of the page again, set it to position: static.

Horizontal Centering

Centering a div or any other **block-level** element horizontally is a special case for CSS layout, even moreso because there is a bug in Internet Explorer's implementation of the standard way of doing it. The standard way is to set the element's horizontal margin values to auto, like so:

```
#wrapper {width: 760px; margin: 0 auto; }
```

That will work in **browsers** like Firefox, Safari or Opera. However, this will have no effect in versions of Internet Explorer below 7. There is a hack we can use though, so that we get horizontal centering in all browsers. To whit, IE incorrectly centers **block-level** elements if the element that they're contained in has text-align: center applied. So we can apply this property to the body element, and all the elements within it will be centered.

```
body {text-align: center; }
```

One final step is then necessary. The line above will, of course, center all the text *inside* the centered elements as well, which is generally not what we want, so we need to align the text within back to the left. So here's all the code:

```
body {text-align: center; }
#wrapper {width: 760px; margin: 0 auto; text-align: left; }
```

Easy when you know how, eh? Be careful if you're planning on mixing absolute positioning and this centering method in the same layout. If you want other elements to be absolutely positioned inside the wrapper, make it relatively positioned first.

```
#wrapper {position: relative; width: 760px; margin: 0 auto; text-align: left; }
```

This will make an inner element that you absolutely position at, for example, top: 0; left: 0; appear at the top left corner of the wrapper, and not of the the top left of the entire window.

Next stop: adding **borders** to your elements.