Dreamweaver Article

Designing with CSS – Part 1: Understanding CSS Design Concepts

Table of Contents

Creating a Cascading Style Sheet

First, I'd just like to touch on the outmoded use of font tags. By default, Dreamweaver 8 uses styles rather than font tags to redefine the appearance of your text content. This is a good move in the right direction that it is tantamount to sacrilege to make the changes to revert back to font tags. Let me explain why—you can see the argument in process in a little while.

To begin with, font tags are *deprecated*, which means they may not be supported by browsers in the future. They are no longer a part of the XHTML specification. That is a good enough reason not to use them.

The real problem with font tags begins when your design client requests changes. Maybe the font color needs to be changed or perhaps the font face needs to change from Verdana to Arial. If the content has been generated using font tags, making these changes can take a huge amount of time. By contrast, you can make changes such as these—and indeed far more complicated ones—in a very short time by editing an external style sheet because it applies the changes on a sitewide basis instantly. You also get easier document manipulation and lighter, faster-loading pages. The benefits are many.

Taking the time to learn CSS, even if you wish to use it only for redefining elements rather than full-fledged page layout, is well worth enduring the small learning curve it takes to get you there. Dreamweaver 8 has made giant strides in its implementation of CSS. Take advantage of these changes. I guarantee you won't be sorry!

How Do You Use CSS?

Let's start at the beginning of the CSS trail and look at the methods available to you when you want to apply a Cascading Style Sheet to your documents. You can use an external style sheet, an embedded style sheet, or inline styles. External style sheets are best because they give you the most control over styles.

Using an External Style Sheet

By using an external (linked) style sheet, you exercise global control over the appearance of every page in your site that is linked to that CSS file. This is a very powerful method of using CSS. By using a linked style sheet, you have the ability to make sitewide changes on countless numbers of pages from a single location. This can be completed in what may amount to no more than a few seconds' work. Powerful indeed!

Dreamweaver makes setting up an external style sheet just about as easy as it can be. In the first exercise, I'll review the procedure of creating an external style sheet and how to link it to your web pages:

1. Select File > New or press Control+N to open the New Document dialog box (see Figure 1).

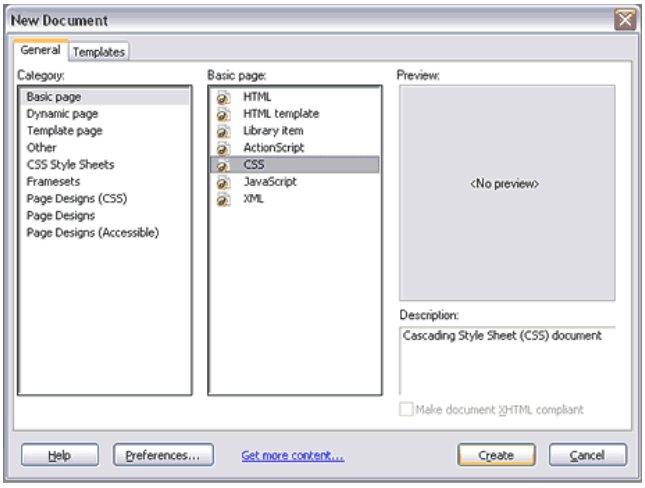

Figure 1. The New Document dialog box

2. In the Category column, select Basic Page.
3. In the Basic Page column, select CSS.

4. Click the Create button.

Your newly created Cascading Style Sheet will open in Dreamweaver. If you are completely unfamiliar with CSS, you will notice that style sheets don't have a Design view. Your style sheet is little more than a text file that contains your CSS rules and their properties and values.

Alternatively, if you select the CSS Style Sheets option in the Category column of the New Document dialog box, the Basic Page column will show a list of "starter" style sheets that already contain some of the CSS rules you may use. You won't be using these starter pages at the moment. Instead, you will be building your own style sheet, and I will discuss why you are setting the properties and values you will use.

Now that you have created your first CSS file, use the following steps to save it in a defined site:

1. If you haven't already defined a site, do so now. If you need help defining a site, see "How to Define a Site in Dreamweaver" (TechNote 14028).
2. Create a new folder called **CssFiles** in the root of your site.
3. Save the CSS file in the new CssFiles folder and name it **external.css**.

Note: Normally, I suggest saving CSS files in their own folder just for the sake of good organization. This keeps your site neat and tidy as it grows.

In this section, you will create two pages that will demonstrate the power of using an external style sheet:

1. Open the New Document dialog box (File > New).
2. Select Basic Page from the Category column.
3. Select HTML from the Basic Page column.
4. (Optional step) Click the Make Document XHTML Compliant option.
5. Click the Create button.
6. Save the page to your site root and name it **external.html**.
7. Repeat Steps 1–5 to create a second page.
8. Save the second page to your site root and name it **external2.html**.

You now have all the documents you need to complete the first section of this tutorial.

Linking the Cascading Style Sheet

You should now have Dreamweaver open and have the following three documents displayed, saved, and ready to work on:

- external.css
- external.html
- external2.html

If you don't have these files open in Dreamweaver, open them now.

The following demo walks you through the process of linking your style sheet to your HTML documents.

Play the demo: Attaching an External Style Sheet

Here are the steps reviewed in the above demo:

1. Open external.html, external2.html, and external.css in Dreamweaver 8.
2. Press Shift+F11 to open the CSS Styles panel.
3. Make sure external.html is the active document, and click the Attach Style Sheet button in the lower right corner of the CSS Styles panel. The Attach External Style Sheet dialog box opens. Dreamweaver remembers your last selection.
4. Select the Link option.
5. Click the Browse button.
6. The Select Style Sheet File dialog opens.
7. Select Document from the Relative To pop-up menu.
8. Select the external.css file you created earlier. The path is inserted in the File/URL field.
9. Click OK to add your style sheet name to the CSS Styles panel.
10. Click the Code view button. You will see that Dreamweaver has inserted the necessary code linking your document to your style sheet.
11. Select File > Save to save the file.
12. Repeat the process to link external2.html to your style sheet.

Before you begin to add rules into your style sheet, there are one or two things you should cover first. In the next section, you will learn why it is important to use a complete doc type. I will then give a quick introduction to margins and padding before discussing browser default settings and why it is a good practice to zero off these defaults so you start on a level playing field.

Dreamweaver Article

Designing with CSS – Part 1: Understanding CSS Design Concepts

Table of Contents

Margins, Padding, and Doc Types

Dreamweaver 8 (and MX 2004) adds a complete doc type to every page you create. This was another big step in the right direction from Macromedia because an incomplete doc type can cause many problems with your CSS.

A complete doc type is shown below. In this case, the doc type is a Transitional XHTML doc type:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

FEEDBACK

An incomplete doc type will cause your browser to drop into what is called "quirks mode." This will very quickly cause you to lose your hair and any sanity you may have if you are aiming for a pixel-perfect design. Ensure that your doc type is a complete doc type at all times. If you are using a version of Dreamweaver prior to MX 2004, you may want to view the available doc types you can use at the World Wide Web Consortium (W3C).

You should also be aware that if you have anything above the doc type in Internet Explorer—even a comment—the browser will revert to quirks mode. This, of course, does not include the use of any server-side code you may have above the opening XHTML tag. Server-side code is, of course, processed on the server and not returned to the browser in the manner I am talking about here.

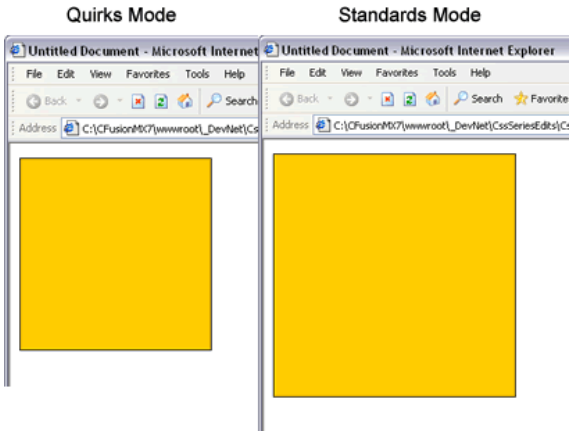

Figure 2. The differences between quirks mode and standards mode can be quite considerable

In figure 2, you are looking at Internet Explorer 6 (IE6). When this browser—and earlier versions—drops into quirks mode, it has a unique problem in that it gets the CSS box model completely wrong. It includes the padding within the defined width rather than adding the padding to the width.

You can clearly see the results in Figure 2. While this may not mean too much to you at the moment, you will be able to see the importance of ensuring your pages are rendered in standards mode. The small box in Figure 2 shows IE6 in quirks mode. In this instance, quirks mode was triggered by placing a simple HTML comment above the doc type.

What Are Margins and Padding?

Margins and padding exist within many of the XHTML elements you will use within your design work. Figure 3 represents a <p> element. The text is clearly visible and the red area surrounding it indicates the presence of a padding value. The black border around the red area indicates the boundary of the <p> element. Increasing or decreasing the padding value causes the red area to expand or contract to suit the value you provide in your CSS rule for the <p> element.

The blue area represents the margin value. The margin value pushes away other elements surrounding the <p> element as long as they are in the document flow. (I will talk more about the document flow later in this series.) Increasing and decreasing the margin value determines how closely the elements surrounding the <p> element are allowed to encroach upon it.

Figure 3. Margin and padding example

So to recap: Padding resides within an element and margins reside outside the element.

The Syntax of CSS

CSS is a simple language. It is easy to read and takes very little time to grab the basics and start building your own style sheets. After you complete this section of the tutorial, you will have a good understanding of the syntax and various elements that exist within CSS rules. I begin by creating a simple rule that clearly lays out the structure for you to see:

```
selector {
property: value;
}
```

Selector

The first part of the rule is called a *selector*. The selector represents a structure that can be used as a condition to define how an element looks in the browser.

Property

The *property* section of the CSS rule defines a specific area of the structure, such as padding or margin properties.

Value

The *value* section of the CSS rule defines a measurement—in general terms. If you are defining how a <p> element may look, the property might be a reference to the <p> element's padding and the value might be 10 pixels:

```
p {
padding: 10px;
}
```

A CSS rule may, of course, contain many property and value pairs. Each would perform a specific task within the structure to provide you with the look and feel you want for the rule.

Property and value pairs are separated by a colon (:). The space after the colon is optional. Immediately following the value, you have a semicolon (;). The semicolon is said to be optional after the last property/value pair in your CSS rule. However, I prefer to leave it in place.

That is really all there is to CSS syntax; it is as simple as that. While it is always good to know your code, it is also good to know that Dreamweaver writes the CSS rules for you, as you shall see shortly.

Longhand or Shorthand?

When you write your CSS rules, you have an option of two different writing styles: longhand and shorthand. Dreamweaver allows you to set your own preference for the writing style. Here are the steps to access the Dreamweaver CSS preferences:

Select Edit > Preferences, or use the Control+U shortcut. In the Preferences dialog box, select the CSS Styles option in the Category list (see Figure 4).

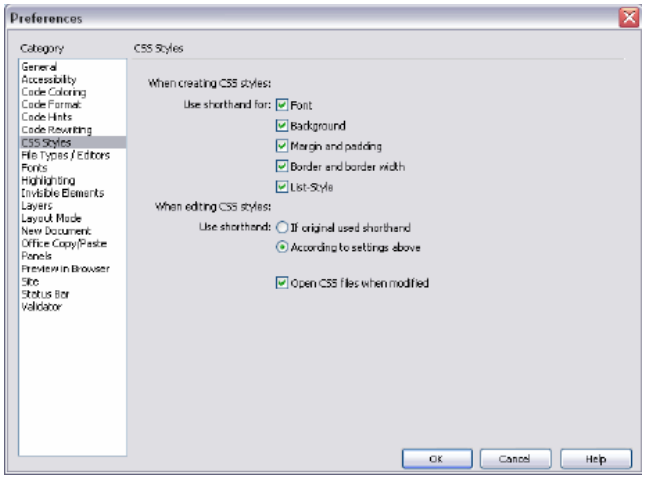

Figure 4. Setting your CSS writing preferences

By selecting all the Use Shorthand For options, as well as the According to Settings Above option, you tell Dreamweaver to write your CSS rules in

shorthand. That's fine, but what is the difference between longhand and shorthand writing styles?

The obvious answer is that shorthand creates a smaller file. Shorthand allows you to condense your rules in such a way that you can assign multiple properties and values in a single line. More importantly, you can write your rules and modify them faster—less work all around.

To show you how this is achieved, I will write the same CSS rule in longhand and then again in shorthand. The difference will be obvious. Once you are familiar with shorthand, I will show you how it can be condensed further to reduce the amount of required information.

Let's begin by creating a CSS rule that redefines the margins of the `body` element, first in longhand and then in shorthand. Go to your external.css file and type the following longhand code to set the body margins:

```
body {
margin-top: 0;
margin-right: 10px;
margin-bottom: 0;
margin-left: 10px;
}
```

Right under that code, type the following shorthand version of the same code:

```
body {
margin: 0 10px 0 10px;
}
```

There's quite a bit of difference here. Using the shorthand method, you eliminate the need to describe each side of the body you want to affect in the property side of the rule. By simply declaring "margin," you avoid having to declare each `margin` property within your CSS rule. You also are able to apply the values in a single line of code. This reduces the CSS property from a four-line affair to a single line.

Delete both the longhand and shorthand code you just wrote.

How does the browser know which value is for which side of the `body` element? Simple, the browser takes the order of the values into consideration and then applies them accordingly. Reading from left to right, the values are applied to the top, right, bottom, and left margins. In this scenario you would have a 0 margin to the top and bottom of your page, while the left and right margins would have a value of 10 pixels.

You can further reduce the shorthand rule as follows:

```
body {
margin: 0 10px;
}
```

When the browser comes across a rule like this one, it knows that the values are pairs of values. The first value is taken and applied to the top and bottom margins while the second value—10px in this instance—is applied to the left and right margins.

If all the margins of the body are the same value, you set the value only once:

```
body {
margin: 0;
}
```

In the rule shown above, a value of 0 is applied to all four sides of the `body` element. Zeroing the margins on the `body` element is a good practice, at least initially. Browsers tend to apply default margins on the body (and other elements) if they are not explicitly stated. Explicitly zeroing the defaults and setting the margins you require is the best way to guarantee that browsers apply the margins you want. Leaving the defaults can result in unexpected displacements within your page because default margins vary greatly from browser to browser.

The Opera browser is a special case because it also applies a default padding setting on the `body` element. This means you need to zero off the padding on the body as well as the margins to get a consistent cross-browser starting point. To achieve this, your final `body` rule would look like the one below:

```
body {
margin: 0;
padding: 0;
}
```

The above rule provides you with a cross-browser zeroed `body` element—a good base from which to start when laying out pages.

Now that you understand about zeroing your page margins and the importance of a full doc type, return to your external.css file and add the `body` rule to your style sheet using the Dreamweaver user interface. The demo below walks you through the process.

Play the demo: Setting the Margin and Padding Values

Here are the steps reviewed in the demo:

1. Click the New CSS Style button in the CSS Styles panel. The New CSS Style dialog box opens.
2. Select the Tag Selector Type option.
3. Select Body from the Tag pop-up menu.
4. Select the `body` element.
5. Ensure that Dreamweaver has selected the correct style sheet in the Define In pop-up menu.
6. Click OK to open the CSS Style Definition dialog box.
7. Select the Box category option.
8. Ensure that the Same for All options are selected for both Padding and Margin.
9. Enter **0** in the Padding field.
10. Enter **0** to the Margin field.
11. Click the OK button.

12. Look at the code that Dreamweaver enters in the external.css file. Although Dreamweaver has added a value for the measurement (px), this is not necessary for a zero value.
13. Select the `body` rule and delete it. You will next use the Dreamweaver code hints to write your CSS `body` rule.
14. Type **body {** . As you begin typing, the code hints appear.
15. Select Margin from the code hint list.
16. Enter a value of **0** after the `margin property: margin:` value.
17. Press Enter to go to the next line of code and type **p**.
18. Select Padding from the code hint list.
19. Enter a value of **0** after the `padding property: padding:` value.
20. Select File > Save to save the file.

It is possible to reduce the amount of characters in a color value within your style sheet rules. If a color is defined in three pairs (for instance, red is #FF0000), that could be written as #F00, where each character represents a matching pair. The F represents FF and the single zero in the second and third positions of the value indicate 00 and 00. This technique can only be applied if the color value contains three pairs. Other examples of this would be #FFFFFF (white) written as #FFF and #000000 (black) written as #000.

Alternatively, each color value could be set using the color's name:

- `color: red;`
- `color: black;`
- `color: white;`

The syntax of CSS is a flexible one. In many cases, it allows you to use different syntax to achieve the same results.

In the next section, we will look at how you can lay out your web pages correctly by using semantic markup.

Dreamweaver Article

Designing with CSS – Part 1: Understanding CSS Design Concepts

Table of Contents

Semantic Markup

> "The Semantic Web approach develops languages for expressing information in a machine processable form" (W3C)

The line above is a quote, so it resides in a pair of <blockquote> tags. This tag lets a screen reader know that the text is a quote and not just another paragraph. Screen readers are machines that enable visually impaired people to surf the web. Creating a semantically correct document in its simplest form is just one case of using the XHTML elements supplied with the specification to lay out your document correctly.

Each element provided in the XHTML specification has been designed to be used in a specific way and each has a specific meaning. An `h` el a title and any text between title tags is instantly recognized as a title by a machine reader, just as text within <p> tags is recognized as being a paragraph. Not all tags are semantic in their makeup. Take the tag, for instance. A tag has no meaning associated with it; it is simply a container element. If you were using a tag and styled it to emphasize text within a paragraph, you would be better off using the tag. This is what it was designed to do. It says to the machine reader *this text should be emphasized*; in turn, the machine reader emphasizes the text within the tags.

Semantic Improvements in Dreamweaver 8

This is another area in which Dreamweaver progressed greatly. In earlier versions of Dreamweaver (prior to MX 2004), pressing Control+I or clicking the I button in the Property inspector inserted a pair of <i> tags to italicize the text. While this looked fine to the human eye, the <i> tag means absolutely nothing to a machine reader. Dreamweaver 8 (and MX 2004) now inserts tags when you press Control+I or click the I button in the Property inspector. A machine reader understands that the tag means to emphasize. In much the same way, the Control+B shortcut (or B button in the Property inspector) now inserts tags rather than <b> tags. The resetting of these keyboard shortcuts in Dreamweaver MX 2004 was a move in the right direction for creating semantically correct documents.

Hierarchical Structure

Creating text in a `p` element and then using font tags—or CSS for that matter—to increase the text size so it looks like a heading and then applying a bold tag to it does not make it a header. The finished result might look like a header to the human eye. However, that's where the association of the text with a header ends.

If you want to create a header, you should use one of the header elements. The header element you use depends on where you are in your document. The main heading on your page would be an `h1`. Subheadings would be `h2`, and so on. You have no need to use all the headings within your document but they are available if you require them:

- `h1`
- `h2`
- `h3`
- `h4`
- `h5`
- `h6`

Each header element is hierarchical in nature, with `h1` being the most important header and `h6` being the least important header. All of them have a natural place within your documents. Commonly you will see designers typing text directly into a <td> tag set within a table—not good. This practice gives machine readers no indication at all as to how to read the text. If you are dealing with copy text below a header, then you should probably set it within a `p` element. Once the text is set within the <p> tags, machine readers know what type of content they are looking at and how it should be read. The idea is to set your document out in a hierarchical manner that is easily read not only by humans but by machine readers (see Figure 5).

```
<body>
<hl>Use hl elements for your main title</hl>
<p>We should use p tags for creating paragraph text</p>
<h2>Use the correct element for sub headings</h2>
<p>and the content should still be inside p tags</p>
</body>
```

Figure 5. Structuring your document correctly

I can hear you saying, "OK, that sounds fine but `h1` elements are *huge*! I understand how making semantically correct documents is a good thing, but I also want my documents to look good."

Enter CSS

CSS provides you with the ability to redefine elements to take on the appearance *you specify*. This is cool because you can set the size of `h1` to whatever you want. You can set the font face, margins, padding, color, background color, and a whole list of other properties all from within a single CSS rule. This keeps your document semantically correct.

Not only can you redefine the `h1` element, but once its rule is safely in your style sheet and the pages in your website are linked to that style sheet, you can change its appearance globally from a single location in a matter of seconds. The demo below walks you through adding an `h1` element and demonstrates the ease with which you can control your documents from an external style sheet.

Play the demo: Redefining an h1 Element

Here are the steps reviewed in the above demo:

1. Click the New CSS style button in the CSS Styles panel. The New CSS Style dialog box opens.
2. Ensure that the Tag Selector Type option is selected.
3. Select h1 from the Tag pop-up menu.
4. Select external.css from the Define In pop-up menu.
5. Click OK.
6. Select "Georgia, Times New Roman, Times, Serif" from the Font pop-up menu.
7. Enter **100** in the Size text box and select **%** from the unit of measure pop-up menu.
8. Enter **#003366** in the Color text box.
9. Select the Box category.
10. Click the Same for All option in both the Padding and Margin sections.
11. Enter a value of **0** for both the Margin and Padding.
12. Click OK.
13. Open external.css. You can see your newly added `h1` rule.
14. Select File > Save to save the external.css file.

As you can see in the CSS Styles panel, your `h1` rule is now shown with the body element in the external.css tree.

Zoe Gillenwater wrote an excellent article for Community MX that goes into more detail on semantic markup. It also includes information on how semantic markup can help your search engine positioning.

Dreamweaver Article

Designing with CSS – Part 1: Understanding CSS Design Concepts

Table of Contents

ID, Class, and Descendant Selectors

In the previous section I reviewed the parts of a CSS rule. I now take you through some of the more commonly used selectors and look at how they are used.

The ID Selector

I begin by defining a div with an *ID selector*. The div you define here is a container or wrapper div. Using a wrapper div is a common practice in CSS positioning, and we will look deeper into how they work later in Part 3. The ID selector is preceded by the pound sign (#) within your style sheet (see Listing 1). The margins you set should be somewhat familiar to you from the way you set up your body margins earlier. I am using a pairs value to declare the settings. As you can see, the top and bottom will be zero.

```
#wrapper {
width: 770px;
margin: 0 auto;
}
```

Listing 1. Wrapper ID selector

The `auto` value may be new to you, so let me explain. When you progress to laying out your first CSS positioning document in Part 3, you will create a fixed-width layout. The width of your container or wrapper div will be 770 pixels to ensure that you have no horizontal scrolling for your viewers who have set their browsers to an 800-pixel-wide resolution. The margin value of `auto` is applied to the left and right sides of the wrapper div. This means that regardless of the users' browser window width, the page content is always centered horizontally. The space on either side of the fixed-width wrapper div is distributed evenly by the *auto pairs value* in your style sheet. You will see how these values are set in the Creating an ID Selector demo at the end of this section.

The pound sign (#) in your style sheet indicates that your wrapper div is an ID selector. However, it is not used when you reference the ID selector from within your XHTML code. If you were inserting the wrapper div into your XHTML document, you would reference it as shown in Listing 2.

```
<div id="wrapper">
Our wrapper div content
</div>
```

Listing 2. ID referenced in HTML, as opposed to how it is written in the CSS file

As you can see, you reference the wrapper by using `id="wrapper"`. The properties and values you set for your wrapper div within your style sheet are now applied to the wrapper div in your XHTML code.

Caution: You can only use an ID selector name *once* in your XHTML document. For example, if you defined an ID selector in your style sheet and then reused that selector name on an image swap behavior within the same document, you will get some very odd behavior.

Play the demo: Creating an ID Selector

Here are the steps reviewed in the demo:

1. Click the New CSS Style button in the CSS Styles panel. The New CSS Style dialog box opens.
2. For Selector Type, click the Advanced option.
3. Name the ID selector **#wrapper**. When you create an ID selector, you type the name into the Selector field. An ID selector is always preceded by the pound (#) sign.
4. Ensure that you have external.css selected in the Define In pop-up menu.
5. Click OK.
6. Select the Box category.
7. Deselect the Same for All option in the Margin section.
8. In the Top text box, enter **0**.
9. In the Right text box, enter **auto**.
10. In the Bottom text box, enter **0**.
11. In the Left text box, enter **auto**.
12. Select the Border category.
13. Deselect the Same For All option in the Style area. You are going to add a 1 pixel solid black border to the left and right sides of your #wrapper ID.
14. Select Solid from the Right and Left pop-up menus for the border style.

15. Deselect the Same For All option in the Width section.
16. Enter **1** in the Right and Left text boxes for the border width.
17. Deselect the Same For All option in the Color section.
18. Enter **#000000** in the Right and Left text boxes for the border color.
19. Select the Positioning category.
20. Select Relative from the Type pop-up menu.
21. Click OK.
22. Open the external.css file. You will now see the newly created #wrapper rule.
23. Select the File > Save to save the external.css file.

Your `#wrapper` selector is now in the CSS Styles panel.

The Class Selector

Unlike the ID selector, the *class selector* can be used as often as you need within your XHTML document. The class selector is preceded by a period (.) within your style sheet (see Listing 3).

```
.leftimage {
margin-right: 15px;
margin-bottom: 5px;
}
```

Listing 3. Class selector

The period (.) in the style sheet indicates that the rule is a class selector. However, the period is not used when you reference the class selector from within your XHTML code. For example, if you were inserting the `leftimage` class into your XHTML document, you would reference it as shown in Listing 4.

```
<img src="images/mypic.jpg" alt="some text " width="145" height="180" class="leftimage" />
```

Listing 4. Referencing a class selector within XHTML

The final outcome is the same whether you use a class selector or an ID selector. Because you will need to apply the settings you have declared within your rule on many occasions, and often more than once in the same document, you really need this rule to be a class selector instead of an ID selector.

Play the demo: Creating a Class Selector

Here are the steps from the demo:

1. Click the New CSS style button in the CSS Styles panel. The New CSS Style dialog opens.
2. For Selector Type, click the Class option.
3. Name your class selector **.leftimage**. A class name is always preceded by a period.
4. Ensure that the external.css style sheet is selected.
5. Click OK.
6. Select the Box category.
7. Deselect the Same For All option in the Margin section.
8. Give the Right margin a value of **15**.
9. Give the Bottom margin a value of **5**.
10. Click OK.
11. Open the external.css file and view your `.leftimage` class rule.
12. Select File > Save to save the external.css file.

The `.leftimage` class appears in the external.css tree in the CSS Styles panel.

Pattern Matching and the Descendant Selector

The *descendant selector* is a powerful way to target specific elements within specific areas of your XHTML documents. Using a descendant selector, you could, for instance, easily target any `em` elements that reside within `p` elements. There is no need to create a separate class and then apply it to the tags. You can simply target them from the style sheet. There is a space between each element within the descendant selector (see Listing 5). A descendant selector works in much the same way as the forward slash (/) does in a folder hierarchy within a URL.

```
p em {
color: #990000;
}
```

Listing 5. Descendant selector applying to `em` elements within `p` elements

The rule shown in Listing 5 says, Find `p` elements and if those `p` elements contain `em` elements, change the text color to #990000.

For any `em` elements that exist outside a `p` element, or within a different element, the pattern is broken and the text color change is not applied. The descendant selector allows you to pattern-match your XHTML documents and to be very specific as to how you affect elements with any given rule.

Play the demo: Creating a Descendant Selector

Here are the steps reviewed in the demo:

1. Click the New CSS style button in the CSS Styles panel. The New CSS Style dialog box opens.
2. For Selector Type, click the Advanced option.
3. In the Selector text box, enter **p em**. Notice the space between the two elements. This is important; this space is known as a descendant combinator.

4. Ensure that the external.css style sheet is selected.
5. Click OK.
6. Select the Type category in the CSS Style Definition dialog box that opens.
7. Enter **#990000** in the Color text box.
8. Click OK. Your newly created descendant selector appears in the external.css file.
9. Select File > Save to save the external.css file.

The descendant selector has been added to the external.css tree in the CSS Styles panel.

Redefining a Selector

Review the Defining the `h1` Type Selector demo and then create a new rule in your external.css file. Redefine the `p` element and give one property and value pair. Set the property to **font-size** and set the value to **80%** (see Listing 6).

```
p {

}
```

Listing 6. Redefined `p` rule

Conclusion

If you were totally new to CSS at the beginning of this tutorial, you have come far. You have seen how you can use Dreamweaver MX 2004 to create an external style sheet and link that style sheet to documents within your website. You have learned about selectors and how to create them.

Specifically, you have looked at:

- Redefining elements with the Type selector
- Creating ID selectors
- Creating class selectors
- Creating descendant selectors
- Zeroing off your body element

In Part 2 you will explore CSS further and look at how your document interacts with the relevant CSS Styles panel to make editing CSS a breeze in the Design view. You will look at avoiding a common pitfall or two and lay the foundations further to create your first CSS positioning layout, which I cover in Part 3.

The CSS rules you have created in this tutorial will be put to good use as you work your way through this series. I hope that you now have a good understanding of how to create CSS rules using the Dreamweaver design panels.

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Dreamweaver Article

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Adrian Senior

www.webade.co.uk
www.communitymx.com
www.ukcsstraining.co.uk

Table of Contents

FEEDBACK

Created:
01 November 2004
Modified:
12 September 2005
User Level:
Beginner

Note: This article has been updated for Dreamweaver 8. If you are still using Dreamweaver MX 2004, please read the version of this article series for Dreamweaver MX 2004. The CSS features in Dreamweaver have been vastly improved in Dreamweaver 8. You can learn about those changes in Julie Hallstrom's article, An Overview of CSS in Dreamweaver 8.

Welcome to the second part of this article series on CSS design concepts. If you missed Part 1, you can get to it below:

Designing with CSS – Part 1: Understanding CSS Design Concepts

This series reviews how you can use Dreamweaver 8 to move towards CSS as a positioning technique when developing web pages. Part 2 builds on what you learned in Part 1. I discuss floating elements and investigate the `height` property. In addition, I cover inheritance and specificity. In Part 3 you'll create your first CSS-positioned layout.

You may find it advantageous to read an earlier tutorial I wrote for the Developer Center on relative, absolute, and static positioning, called Introduction to CSS Positioning in Dreamweaver MX 2004. It should give you a decent grounding in what you can achieve with each of the positioning values.

Requirements

To complete this tutorial you will need to install the following software and files:

Dreamweaver 8

- Try
- Buy

Tutorials and sample files:

- css2_samples.zip (ZIP, 2K)

Prerequisite knowledge:

- Lorem Ipsum Generator extension by Captain Cursor Creations (optional)
- Internet Explorer browser

- Firefox browser
- Read Part 1 of this series

About the author

Adrian Senior owns the UK-based web design agency Webade, which has been in business since 1998. Adrian also provides courses for companies and individuals who need to build compliant, accessible websites using CSS and XHTML. He is also an Adobe Community Expert and a partner at Community MX. The year 2004 saw Adrian's first trip to America, where he visited Orlando and delivered two sessions at the TODCon conference.

Before becoming involved with website design and development, Adrian's working life centered around the shipyards of the River Mersey and oil production units in the North Sea. Adrian has been married to his wife, Janette, for 26 years. They have two children, Antony and Eleanor.

Dreamweaver Article

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Table of Contents

Adding Default Properties to the `Body` Rule

Let's begin by building on the `body` rule you created in Part 1. There you created three files that you were working with: external.html, external2.html, and external.css.

1. Open the three files from Part 1 in Dreamweaver 8 and make external.html the active document by clicking its tab.
2. Press Shift+F11 to open the CSS Styles panel.

Figure 1. CSS Styles panel displaying the external.css file

3. Select the `body` tag. You will see the Tag inspector update to show the CSS properties for the `body` element.

Neat! As you can see, by simply selecting the CSS rule in the CSS Styles panel, you can quickly and easily gain access to the properties you have already set—and others you may want to use by clicking the Add Property link.

Zeroing off default margin and padding values

Block-level elements carry default margin and padding values, and these default values often gets designers into a difficult spot when creating designs that need to display accurately across a range of browsers and operating systems. The problem is that the default settings used for these properties are not consistent; in fact they can vary wildly from browser to browser and operating system to operating system.

It is good practice to remove these default values (we did this for the body element in part 1 when we set the margin and padding to zero). Although this technique works fine, it can be somewhat laborious to do this to all elements that require zero values. With this in mind, you'll create a group selector that will handle all the zeroing for you.

Before you do this, you will first take a look at what I mean by grouping selectors and it will be necessary for you to understand inheritance and how inheritance works within our style sheet.

Grouping selectors

If you need to apply the same set of properties and values or a base set of properties and values that are the same for two or more elements then you can *group* your selectors by separating each selector with a comma. Examine listing 1 below.

Note: Fonts that consist of more than one word are set in quotes.

```
h1 {
font-family: "Times New Roman", Times, serif;
color: #003366;
}
```

Listing 1: Setting the font-family and color for an h1 element

Should you decide that you want all your h elements to be that color and of that font then we can simply group them, as shown in listing 2.

```
h1, h2, h3, h4, h5, h6 {
font-family: "Times New Roman", Times, serif;
color: #003366;
}
```

Listing 1: Setting the font-family and color for all the h elements

That is somewhat quicker than writing out an individual rule for each h element. You now have all your h elements grouped; you have declared each h element within the same rule by separating each selector with a comma.

Inheritance

Inheritance allows you to re-use properties and values on specific elements. Let's revisit listing 2 and build on that as shown in listing 3.

```
 h1, h2, h3, h4, h5, h6 {
font-family: "Times New Roman", Times, serif;
color: #003366;
}

h1 {

}
```

Listing 3: The properties and values from your grouped selector will inherit into the h1 type selector;

When rendered in the browser your h1 element will now look like listing 4.

```
h1 {
font-family: "Times New Roman", Times, serif;
color: #003366;

}
```

Listing 4: The inherited h1 rule

All the properties and values are brought together by inheritance to be combined as a single rule. The rule closest to the bottom of the style sheet always wins out against one higher in the style sheet when the selectors are the same, it is deemed to be more specific. Let's revisit listing 3 and make a slight change as shown in listing 4.

```
h1, h2, h3, h4, h5, h6 {
font-family: "Times New Roman", Times, serif;
color: #003366;
}

h1 {

color: #000000;
}
```

Listing 4: Adding a second color value

As you can see your lower and ungrouped h1 selector now has an added color value of #000000 – black. Because this color value is lower in the style sheet – or closer to the h1 element in the XHTML document – it is deemed to be more specific. For this reason the black color value will win out over the color value set in the grouped selector, which is higher up in the style sheet. From this example you can see how it is possible to override settings within CSS rules with alternate properties and values. This is the basis of the method you will use to create your zeroing rule.

Imagine if you had an embedded h2 rule in the head of your page and you placed it below the link to your external style sheet. In this instance, the h1 rule that is embedded in the head of your page would override the h2 rules in your external style sheet. By the same token, an inline rule (like the one shown below) would override the h2 rule in the head and the h1 rule in the external style sheet:

```
<h1 style="">My h1 font size is now 60%</h1>
```

This is all well and good. However, you really do not want to use embedded or inline styles in your XHTML documents. Although it is not incorrect to use CSS in this manner, it does prevent you from having global access to your CSS rules. There is little point in using CSS and then having to open goodness knows how many XHTML pages to change embedded or inline rules. Keep your CSS rules where they give you the most benefit—in an external CSS file.

Play the demo: Creating a Zeroing Selector

The steps in the movie are as follows, press the enter key after each step:

1. Type: html, body, ul, ol, li, p, h1, h2, h3, h4, h5, h6, form, fieldset {
2. Type: margin: 0;
3. Type: padding: 0;
4. Type: border: 0;
5. Type: }

Setting the Font Properties

The `body` element provides you with an excellent opportunity to set some default settings. Take this opportunity to set your font properties and values and allow the rest of your document to inherit the settings you set here.

For instance, if you declare a color on the `body` element, then that color will be the color used for all text, no matter what element—`h1`, `h2`, or `p`—in which the text is contained. The color is set by inheritance from the `body` element. You can change the text from the default properties and values easily if you wish, you have already seen how you can do that a little earlier in this tutorial.

Open external.html in design view and click the body rule in the CSS panel. The CSS panel will now update to show the properties and values of the body selector. But, we have no properties in the body rule as we deleted them when the margin and padding settings were moved to our zeroing selector. With that in mind you will not see any property and value settings under the "Properties for body" section when the body tag is selected.

Play the demo: Setting Defaults on the Body Rule

Complete the following steps to set a default set of values against the body rule.

1. Click the Add Property link
2. Select font-family from the select list
3. From the values select list choose the Arial, Helvetica, sans-serif option
4. Click the Add Property link
5. Select the font-size option
6. From the first value select list type: 100.01
7. From the second select list choose % for the unit of measurement
8. Click the Add Property link
9. Select the background-color option
10. Click the color cube and select the white cube
11. Click the Add Property link
12. Select the color option
13. Click the color cube
14. Select the #666666 color cube
15. Save your work

You have now set a series of default values on the body rule that will be inherited throughout your document unless you specifically override them.

Dreamweaver Article

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Table of Contents

Setting a Default Font Size for the p Element

Your next task is to set a default font size for the p element. You haven't created a rule for the p element as of yet. You can create that using the technique you learned in Part 1 when you created the h1 rule. Remember that you want to set only a font-size property with a value of 80%. There is no need for anything else in your p rule at this time. Once completed, your p rule should look like Figure 2.

```
35 p {
36     font-size: 80%;
37 }
38
```


Figure 2. The finished p rule

With your h1 and p rules successfully completed, following these steps to put them to use:

1. Type **This is an h1 title scaled to 130%** at the top of the external.html page.
2. Apply your h1 rule to the text by selecting the text and selecting Heading 1 from the Format pop-up menu in the Property inspector.
3. Type **This is default paragraph text scaled to 80%** below it. It should look like Figure 3.

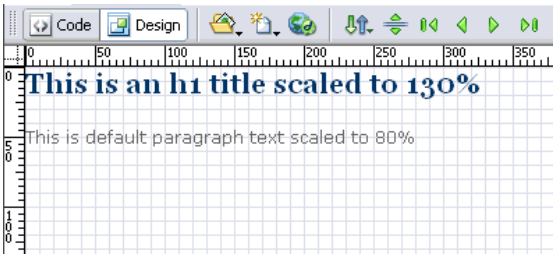

Figure 3. The h1 and the p rules

Let me recap what you have done here and explain the workings behind it all. You have set a default font size, color, and font-family on the body rule. You can see that your p element have inherited two of these settings—the color and the font family. The size is different. You specifically set the size of the p element to 80%. Because the size is set against the p element, it carries a greater specificity than the font size that you set on the body rule. With the exception of your h1 element – where you have also specifically set a default font size, any text you add to your page outside of a p element will be shown at 100.01%.

Your p and h1 elements have specific font-size settings. The font-size values on those two rules are said to be more specific than the font size set on the body element. Because the p and h1 elements have a font size of their own, those values carry a greater specificity and win out against the body rule. In effect, you have overridden the font size set on the body element. It is the same laws of specificity that allowed you to change the color of the h1 element. The p element is still the default #666666 color. You have not set a color within the p rule, so the color set against the body rule is inherited and displayed.

In Part 1 you set up a p em descendant selector and set a color value against it. In the Design view, select the word *default* in the paragraph text and press Control+I to wrap your selection in tags (see Figure 4). You will see the selected text take on the color you set for that rule in your CSS file. This is another example of specificity winning the day.

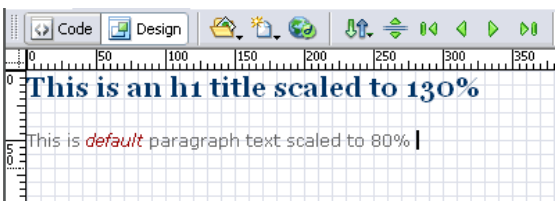

Figure 4. The word *default* with the p em rule applied to it

Dreamweaver Article

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Table of Contents

Investigating the `Height` Property

The `height` property is a common frustration for CSS newcomers, so I'll take you through a little exercise to help you understand it. Use the steps in the demo below to create the following `BlueBox` style, and add the style to your external2.html page. You can also find the completed BlueBox CSS rule in the completed external.css file, which is included in the sample download file linked at the beginning of the article. First attach the style sheet to your external2.html file as you learned earlier in this series.

```
#BlueBox {
background: #6699FF;
height: 250px;
width: 250px;
border: 1px solid #000000;
}
```

FEEDBACK

The following demonstration uses the Lorem Ipsum Generator extension by Captain Cursor Creations.

Note: If you don't want to install Lorem Ipsum Generator extension, you can add text of your choosing in the step where the extension is used. (For instance, you can search Google for "lorem ipsum" or find dummy text at www.lipsum.com.) In any event, to understand the point of this exercise, you will need to add enough text to push the `div` height past 250 pixels. To prevent having to re-preview from Dreamweaver I don't view my pages as temporary files, by disabling this feature I can simply refresh the browser to see changes in my CSS. If you would like to work this way then complete the following steps.

1. Select Edit > Preferences.
2. Select Preview in Browser from the Category list.
3. Deselect the "Preview using temporary file" option (see Figure 5).

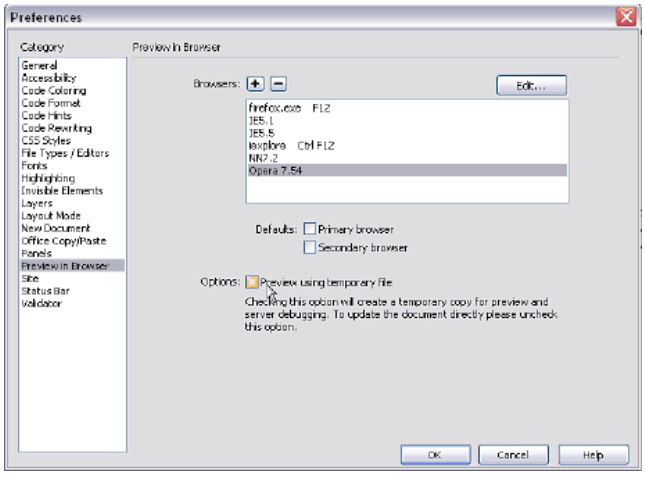

Figure 5. Changing the browser preview settings

If you prefer not to do that, you can simply use the browser preview shortcuts as you would normally.

Play the demo: Investigating the Height Property

Once you have added the #BlueBox selector to your external.css file, open the external2.html file in the Design view. You should see something like Figure 6. From the ruler on the left side of the window you can see that the div is a little over 250 pixels in height, yet you set it to be only 250 pixels high. Something is a little out of sync.

Note: Enable the ruler by selecting View > Rulers > Show. You will also notice that I have the Grid view enabled. You can enable the grid by selecting View > Grid > Show Grid. Adjust the grid settings by selecting View > Grid > Grid Settings.

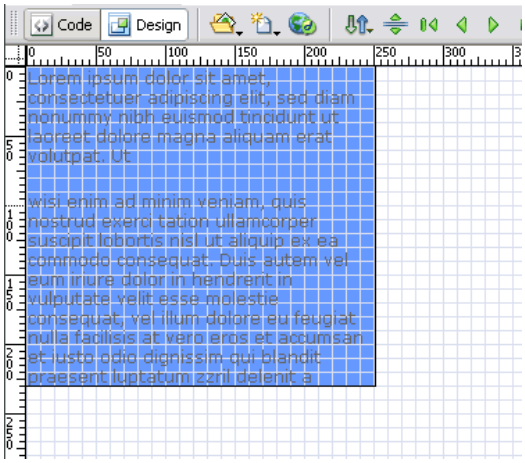

Figure 6. BlueBox div containing lorem ipsum dummy text

To demonstrate the problem, let's test the page in Internet Explorer and Firefox. If you don't have Firefox installed, I recommend installing it. It's a good habit to test web pages in multiple browsers. Firefox is free and you can get it from Mozilla. Although testing pages is imperative, you should also be aware of the Dreamweaver browser compatibility check feature.

Or you can complete the following steps that are shown in the above Captivate demo.

1. Select the new style button
2. Select the Advanced radio button
3. Type #BlueBox into the selector box
4. Click the OK button
5. The CSS rule definition dialog box opens
6. Select the box category
7. Type 250 into the width field
8. Type 250 into the height field
9. Type 10 into the padding field
10. Select the border category
11. Select solid from the style select list
12. Type 1 into the width field
13. Select black from the color cube from the color section
14. Select the Background category
15. Select the color cube from the Background color option
16. Select the #6699FF color cube
17. Save your CSS file
18. Select the external2.html file
19. Switch to code view
20. Insert the code for the BlueBox div
21. Save your work
22. Place your cursor inside the BlueBox div tags
23. Switch to design view
24. Select the Lorem Ipsum extension
25. Insert 1000 characters
26. Place your cursor in the text
27. Create a paragraph by hitting the enter key
28. Repeat the last two steps to create three paragraphs in all
29. Save your work
30. Preview the page in Firefox
31. The text spills outside the div, this is correct behavior
32. Preview the page in Internet Explorer
33. The BlueBox div expands to accommodate the content, this is wrong
34. Open your css file
35. Add overflow: auto; to your BlueBox rule
36. Preview the page again in both browsers - both browsers show scroll bars for the div

Now that you understand how height works and the pitfalls you may encounter, use the CSS panel, or manually edit the CSS code to remove the overflow property you added to your #BlueBox rule. Also remove the height property and value from the rule.

```
#BlueBox {
background: #6699FF;
width: 250px;
border: 1px solid #000000;
}
```

Once you have removed the `overflow` and `height` properties, preview your external.html in both browsers again.

If no value is defined to restrict the div height, it is free to expand to whatever height is required to contain what it holds (see Figure 7). Generally, you shouldn't set a height on your div tags—at least not unless it is absolutely necessary. Leave the `height` property out of your rules when you can and allow the contents to dictate the height of the containing element.

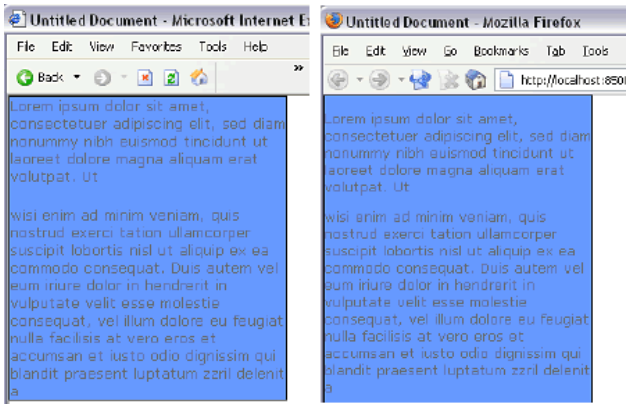

Figure 7. `#BlueBox` div in Internet Explorer (left) and Firefox (right) with the `height` and `overflow` properties removed from the div

Dreamweaver Article

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Table of Contents

Working with Floats

The `float` property allows you to move a floated element either to the right or to the left within its parent container. There are many occasions where the `float` property can be deployed within your work. Let's learn first about floating images within a `p` element.

It is a common practice—at least for me—to set up two generic `float` classes. This is functionality I require on every site I build. One class floats elements to the left; the other floats them to the right.

You made a class earlier called .leftimage and gave it right and bottom margin values. Select this class in your CSS panel to display the prop you've already assigned. Then complete the following steps.

1. Click the Add Property link
2. Select Float from the list
3. From the value list to the right select the left option
4. Save your CSS file

Create a second class called .rightimage and give properties and values of:

1. margin-left: 15px;
2. margin-bottom: 5px;
3. float: right;
4. Save your work

Check your CSS file, your new rule should look like the code below.

```
.rightimage {
margin-left: 15px;
margin-bottom: 5px;
float: right;
}
```

Now apply one of the `float` classes to an image on your page:

1. Open the external2.html file, which you linked to your CSS file earlier.
2. Remove the BlueBox div code.
3. Add a `p` element to your page and insert several lines of text within the opening and closing <p> tags.
4. Switch to the Code view. You need to be able to place the image accurately that you are going to float.
5. Insert your cursor immediately after the closing angle bracket of the opening <p> tag (see Figure 8). You are now in a position to insert your image. It will have one of the float classes applied to it, and it will be floated within the `p` element.

Figure 8. Placing the cursor where you will insert the image

6. Set an image place holder by completing the following steps:
    1. Select the common tab on the insert bar.

2. From the images option, select image place holder.
3. Complete the dialog box.
4. I set the width and height to 50px.

5. Click OK.

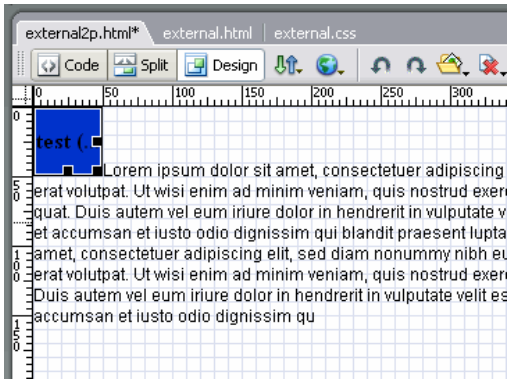

Figure 9. Using an image placeholder

7. Click the image to select it then from the quick tag selector select the image by right clicking and selecting class from the context menu.

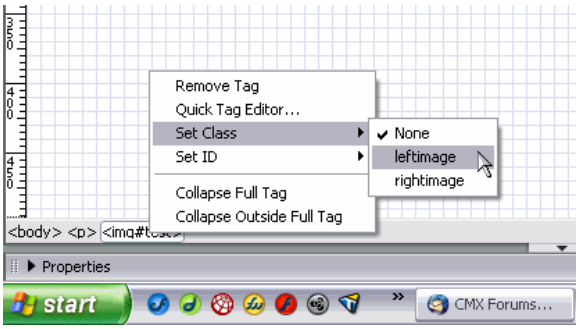

Figure 10. Setting the leftimage class

Once the class has been set, the image will float into position as shown in Figure 11.

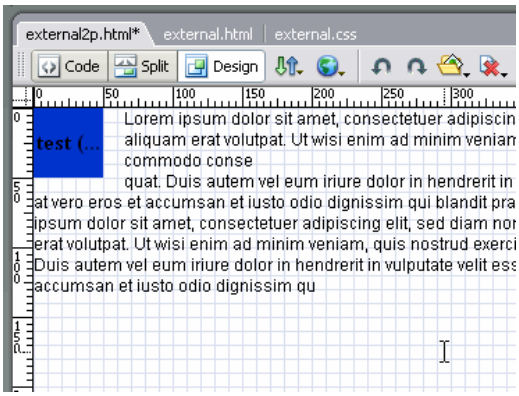

Figure 11. The image floating into position

We have covered quite a bit of ground now. The good news is that with the skills you have learned in Parts 1 and 2, you are now ready to put together your first CSS-positioned layout, which I cover in Part 3. Before you continue, it would be a good idea to read my article, Introduction to CSS Positioning in Dreamweaver MX 2004.

Point of Interest

If you have an interest in the CSS design technique and you also like to make your pages accessible to the Section 508 and WAI specifications, then you might be interested in taking a look at a template that was released on Community MX under the banner name of CMX JumpStarts. It is called Paris and it meets all of these objectives. It has been designed as a learning tool and is something you can use over and over again as a starting point for your website projects. This design is the first in a series that are city-based.

Designing with CSS – Part 3: Creating Your First Design Without Tables

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Adrian Senior

www.webade.co.uk
www.communitymx.com
www.ukcsstraining.co.uk

Table of Contents

FEEDBACK

- Printable version
- Send feedback
- Get an e-mail update of new articles

Created:
23 November 2004
Modified:
12 September 2005
User Level:
Beginner

Note: This article has been updated for Dreamweaver 8. If you are still using Dreamweaver MX 2004, please read the version of this article series for Dreamweaver MX 2004. The CSS features in Dreamweaver have been vastly improved in Dreamweaver 8. You can learn about those changes in Julie Hallstrom's article, An Overview of CSS in Dreamweaver 8.

Welcome to the third part of this article series on CSS design concepts. If you missed Parts 1 or 2, you can get to them below:

Designing with CSS – Part 1: Understanding CSS Design Concepts

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

In this series, we review how you can use Dreamweaver 8 to move towards using CSS as a positioning technique when developing web pages. Now that you have completed Parts 1 and 2, you are ready to move on and create your first CSS-positioned layout. You will design a fixed-width page that allows the contents to flow. You will use an unordered list to create a horizontal navigation system. You will also design a banner image in Fireworks and use it on the page.

You may find it advantageous to read an earlier tutorial I wrote for the Developer Center on relative, absolute, and static positioning, called Introduction to CSS Positioning in Dreamweaver MX 2004. It should give you a decent grounding in what you can achieve with each of the positioning values.

After you complete Part 3 you will have created your first CSS layout. Its structure resembles Figure 1.

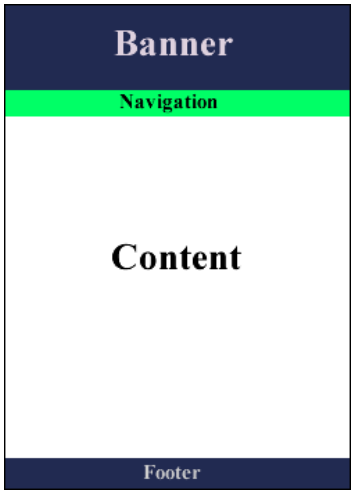

Figure 1. The completed structure

Requirements

To complete this tutorial you will need to install the following software and files:

Dreamweaver 8

- Try
- Buy

Fireworks 8

- Try
- Buy

Sample files:

- css3_samples.zip (ZIP, 237K)

Prerequisite knowledge:

- Lorem Ipsum Generator extension by Captain Cursor Creations (optional)
- Internet Explorer browser
- Firefox browser
- Read Part 1 of this series
- Read Part 2 of this series
- Experience with "How to Define a Site in Dreamweaver" (TechNote 14028)

About the author

Adrian Senior owns the UK-based web design agency Webade, which has been in business since 1998. Adrian also provides courses for companies and individuals who need to build compliant, accessible websites using CSS and XHTML. He is also an Adobe Community Expert and a partner at Community MX. The year 2004 saw Adrian's first trip to America, where he visited Orlando and delivered two sessions at the TODCon conference.

Before becoming involved with website design and development, Adrian's working life centered around the shipyards of the River Mersey and oil production units in the North Sea. Adrian has been married to his wife, Janette, for 26 years. They have two children, Antony and Eleanor.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Using a Correct Doc Type

FEEDBACK

I discussed in Part 1 the importance of using a correct doc type. Using either an incomplete doc type, or no doc type at all, will drop your bro into "quirks mode," which causes your CSS to be improperly rendered and will set you on a road to frustration and confusion. Always set your web pages up correctly. You can do this by making sure you have a valid doc type. (As with the rest of this series, we use the XHTML Transitional doc type.)

Creating a Standards-Compliant Page

To achieve the structure shown previously in Figure 1, lay out your page so that it can be processed in a standards-compliant mode.

1. Select File> New from the main menu in Dreamweaver. This opens the New Document dialog box.
2. Select the General tab.
3. From the Category column select Basic Page.
4. From the Basic Page column select HTML.
5. Check the Make Document XHTML Compliant option.
6. Click the Create button.
7. Save your page as **basiclayout1.html** at the root of your defined site.

Your page should look like the code below. This is what I would call a perfect starting point because Dreamweaver 8 has created an XHTML page for you in its most basic form:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Untitled Document</title>
</head>

<body>
</body>
</html>
```

Add a new folder to your local site and call it **CssFiles**. This is the location that you use to save your CSS documents. Do that now so you can create your CSS file. I covered creating and attaching a CSS file extensively in Part 1.

Save your CSS file as **basiclayout.css**. Once you have completed the exercise of attaching the CSS file to basiclayout.html, the HTML source code will look like the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Untitled Document</title>
<link href="CssFiles/basiclayout.css" rel="stylesheet" type="text/css" />
</head>

<body>
</body>
</html>
```

As you can see, the link to basiclayout.css has been inserted in the head of basiclayout.html. I discussed the merits of using an external CSS file earlier in this series, particularly the ability to control the appearance of your website from a single location.

Note: Avoid using inline and embedded styles whenever possible.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Writing the Body Selector

In Part 1, you learned about defining the margins and padding on the body selector and you saw how CSS allows you to set up margin and pa values. In Part 2 you saw how it is possible to set up a series of default settings, and how other selectors can inherit them. Using this method reduces the size of your CSS file and decreases any style maintenance.

You can now write your zeroing and body selectors into your basiclayout.css file. Set up the zeroing selector as you were shown in Part 2.

```
html, body, ul, ol, li, p, h1, h2, h3, h4, h5, h6, form, fieldset {
margin: 0;
padding: 0;
border: 0;
}
```

- Background color of **#666666**
- Font family of **Arial, Helvetica, sans-serif**
- Default font color of **#666666**
- All four margins set to **0** (zero)
- Four sides set to **0** (zero) **padding**
- Default size of **100%**

If you set up your body selector correctly, it should resemble the code below:

```
body {
font-family: Arial, Helvetica, sans-serif;

color: #666666;
background-color: #666666;
}
```

Note: If you use Dreamweaver CSS dialog boxes, you may see `0px;` as the value of the margin and padding properties instead of `0;`. This is OK, although it is not necessary to provide a measurement unit for zero.

In this series of tutorials I have covered the following methods of adding selectors to your CSS file:

- Using Dreamweaver dialog boxes
- Hand-coding the selectors directly into the CSS file

It makes sense to hand-code selectors if you have the confidence to do it right. If you prefer to use Dreamweaver dialog boxes, that is fine too. However, be sure to examine the CSS file as you work on it because it will help you understand CSS syntax in greater detail. I will be using hand-coding and code hints for writing CSS selectors but use whichever method suits you best.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Writing the Wrapper Selector

As you can deduce from the title of this section, I refer to this div as a *wrapper*. The function of a wrapper div is to act as container element f elements you use within a page. The wrapper also determines where the design is positioned in the body element.

In this example the positioning of the wrapper—and therefore your design—will be positioned in the center of the browser window with a 10-pixel margin at the top and bottom of the window. (If you recall from the earlier parts in this series, you used the auto value on the left and right margins to center the wrapper horizontally in the user's browser.) Revisit the Creating an ID Selector movie from Part 1 if you are unsure how to set up the margins for the wrapper div.

When you are ready to create your wrapper ID selector, give it the properties and values shown below:

- Background color of **#FFFFFF**
- All four sides set as a **black border** with a **solid line** that is **1 pixel wide**
- Width of **770 pixels**—so it fits comfortably at 800 pixels wide without scroll bars
- Margin settings of **10 pixels to the top and bottom margins**
- Margin settings of **auto for the left and right margins**

Your rule for the wrapper should look like the code below (remember that the # in `#wrapper` signifies an ID selector).

```
#wrapper{
width: 770px;
background-color:#FFFFFF;
margin:10px auto;
border: 1px solid #000000;
}
```

I show you how to add the wrapper div into your basiclayout.html page in the following movie.

Play the demo: Adding the Wrapper Div

Preview your page in the Firefox browser (see Figure 2).

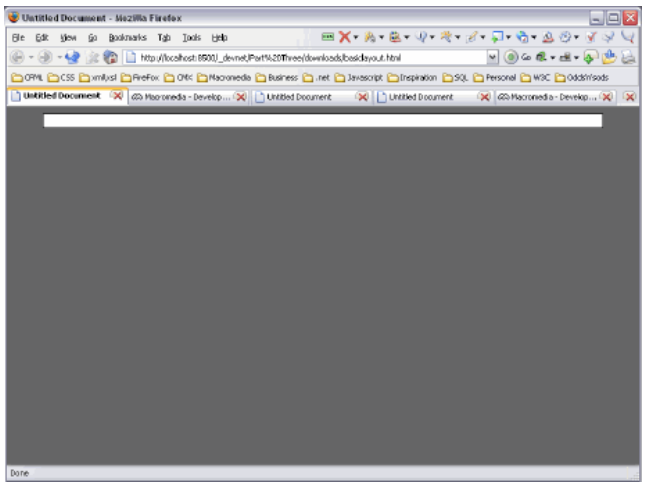

Figure 2. The wrapper div centered in the Firefox browser window

Notice that the margin is somewhat greater than 10 pixels below the wrapper div. This is because the wrapper collapses around the content. You

could use the `min-height` property if you wish, although support for it is fairly poor at this time. I like the div to collapse around the content.

If you have access to Internet Explorer 5.x for Windows, you may want to test your page in one of those versions (see Figure 3).

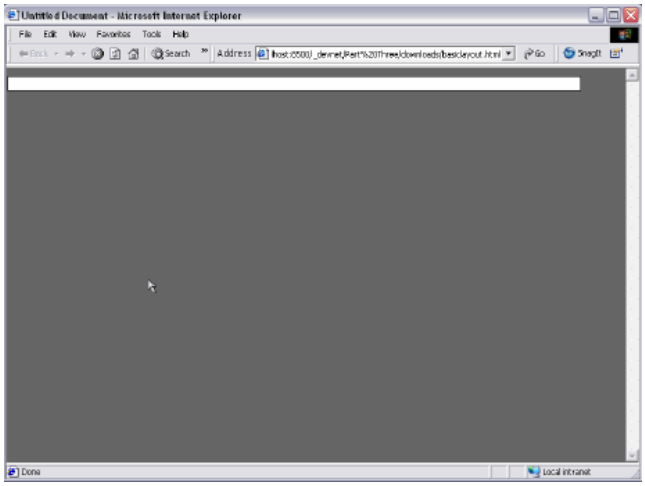

Figure 3. The wrapper div in IE 5.01—not exactly what we were hoping for!

As you can see, IE 5.x for Windows has a problem: It does not respect the auto values for the left and right margin. Not to worry, though; this is a simple fix. You can use another bug in IE 5.x to center the wrapper div.

Add the following property and value pair to your `body` selector:

```
text-align: center;
```

Preview the page again (see Figure 4).

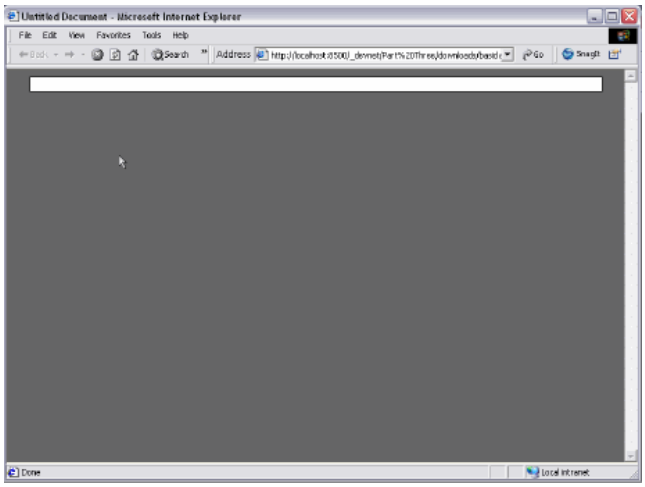

Figure 4. The wrapper div in IE 5.01—that is much better!

IE 5.x also applies the `text-align: center;` property and value incorrectly. It centers the wrapper div, which is wrong but works just fine for your designs needs. Now go back to your wrapper divs rule and add the following property and value pair to realign your text to the left.

```
text-align: left;
```

You have completed the set up of your wrapper div. Your CSS file should look like the following:

```
html, body, ul, ol, li, p, h1, h2, h3, h4, h5, h6, form, fieldset {
margin: 0;
padding: 0;
border: 0;
}
body{
background-color:#666666;
color:#666666;
font-family:Verdana, Arial, Helvetica, sans-serif;
margin: 0;
padding: 0;
text-align: center;
}

#wrapper{
width: 770px;
background-color:#FFFFFF;
margin:10px auto;
border: 1px solid #000000;
text-align:left;
```

```
}
```

Time to move on.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Creating the Banner Graphic

First create the image for your banner:

1. Open Fireworks 8.
2. Create a blank canvas **770** pixels wide by **100** pixels tall.
3. Select the Rectangle tool from the Tools panel
4. Draw a rectangle that completely covers your canvas
5. Give it a color value of **#212A51**.
6. Open the car image in Fireworks 8 and select it.
7. Copy the car image.
8. Paste it onto your newly created canvas and position it on the right side (see Figure 5).

Figure 5. The banner image

9. Click the car image to select it.
10. Choose Commands > Creative > Fade Image to open the Fade Image dialog box (see Figure 6).

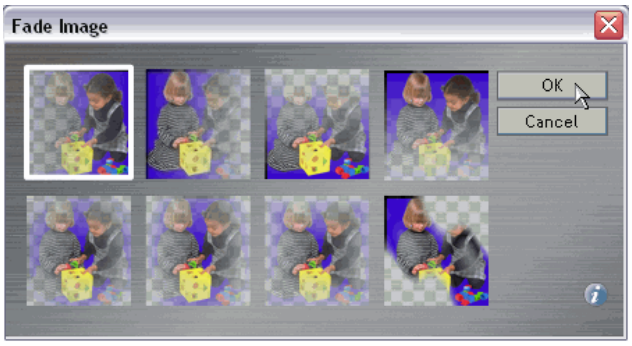

Figure 6. The fade image options

11. Select the top-left option, as shown, and click OK.

Your banner should now show the fade effect applied to the car image (see Figure 7).

Figure 7. The banner image with the fade command applied

You are now ready to add some text to the banner. Let's do this in Fireworks:

1. Select the Text tool from the Tools panel and click on the canvas—anywhere will do.

2. Double-click the Text tool in the Tools panel to open the Text Editor.
3. Enter the text shown in Figure 8 and click OK.

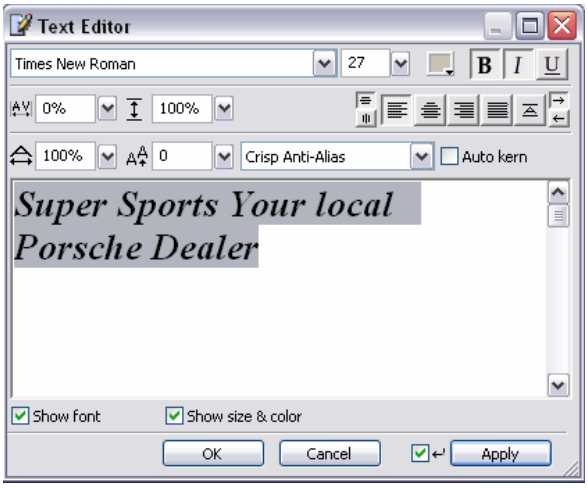

Figure 8. Adding the text to your banner

4. To add a motion effect to the text, click the text area and select Filters > Eye Candy 4000 LE > Motion Trail. (Click OK in the warning dialog box that says this effect converts vectors to bitmaps.)
5. In the Motion Trail dialog box, complete the settings as shown in Figure 9.

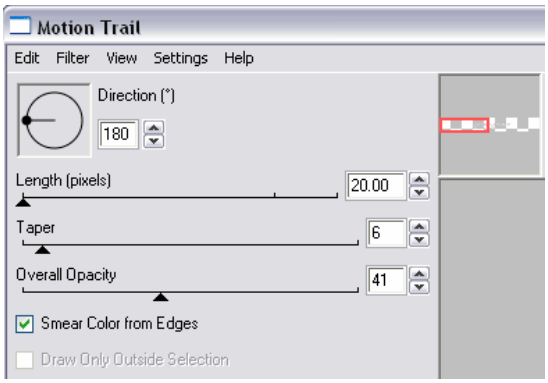

Figure 9. Setting the properties for the motion trail

6. Click OK to complete the text on the banner (see Figure 10).

Figure 10. The motion effect has been added to the text

7. Select File > Export Preview.
8. In the Export Preview dialog box set the image type to **JPEG** and set the compression to **68** (see Figure 11).

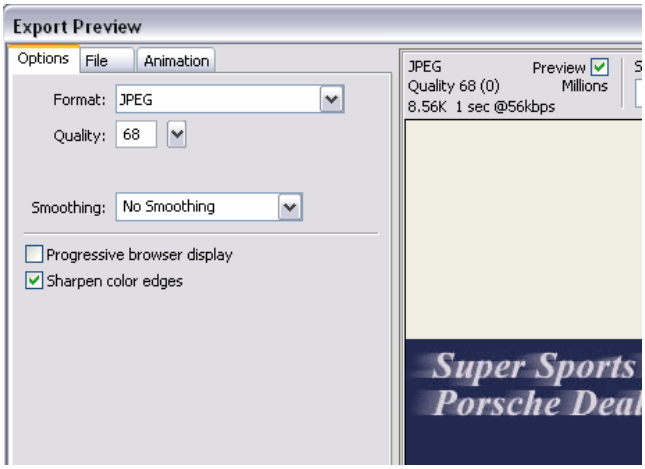

Figure 11. Exporting the banner image

9. Once you are happy with your settings, click the Export button to export the image to your local site.

Now you are ready to create the banner div in the CSS file and then add the code for it in the XHTML document.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Creating the Banner Selector

The next stage of the process is to create the banner div. Because the content within the wrapper flows from top to bottom, you first create th div, then the content div, and finally the footer div. The page will just flow naturally; there will be no defined positioning within this page. So you need to place the elements within the wrapper in the order that they will appear in the document. The first element within the wrapper is the banner, which you created in the previous section. Now create the selector for the banner.

The banner div is an ID type selector and it will be the first div you place within the wrapper div. Because we are simply letting the divs flow down the page, the banner div must take its place right after the opening wrapper div tag. The only property and value pair you need to add to the banner selector is the `height` property and its value. There is no content within the banner div that will be cut off in case the user resizes the browser text, so the `height` property in this instance is safe to use. Make the banner div 110 pixels high.

The banner selector should look like the following:

```
#banner{
height: 110px;
}
```

Now that you have created the banner selector, I will show you how to set up a background image for this particular div. This time let's use Dreamweaver code hints so that you can see just how easy Dreamweaver makes it for you to set up a background image right from within the CSS file.

With your basiclayout.html file open in design view you will use the Add Property function in the CSS panel to add a background image to our banner rule.

Play the demo - Adding a Background Image

1. Open basiclayout.html
2. Open the CSS panel
3. Click the Add Property option
4. Select background-image
5. Select the folder in the value side
6. Browse to your banner_bg.jpg file
7. Double click the file in the browse window
8. Select the Add Property option
9. Select the background-repeat option from the list
10. From the value list select no-repeat
11. Save your CSS file

Your banner selector should now look like the following:

```
#banner{
height: 110px;
background-image: url(../images/banner_bg.jpg);
background-repeat: no-repeat;
}
```

Now add the banner div to the XHTML document, switch to the Code view, and use the same technique that I demonstrated when inserting the wrapper div:

```
<body>
<div id="wrapper">
<div id="banner"></div>
 </div>
</body>
```

This code shows the banner div in position within the page. Notice how it is positioned immediately after the opening wrapper div tag and that the banner is completely encased within the wrapper div. Preview the page in the browser of your choice (see Figure 12).

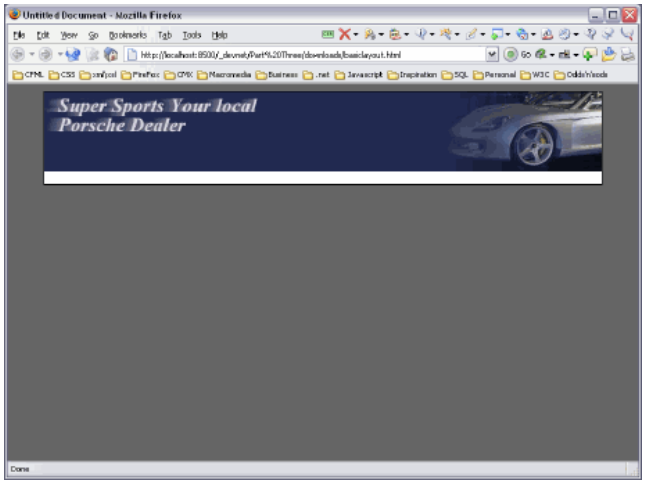

Figure 12. The banner positioned on the page

Next we look at inserting navigation into the design.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Creating the Navigation Selector

The next element to design within the flow of the page is the container for the navigation. Make this an id selector and give it a name of **nav**. Depending on your preferences, you can either hand-code the selector and use Dreamweaver's code hints or you can work through the CSS Styles panel (press Shift+F11) and Dreamweaver's dialog boxes.

Define your nav ID selector to have a black border, 1 pixel wide, and on only the bottom edge.

Play the demo: Creating the Nav ID Selector

1. Place your cursor below the #banner selector.
2. Type **#nav{** and press Enter. The code hints appear.
3. Select `border-bottom` from the list and press Enter.
4. Type the short-hand for the value: **1px solid #000000;**
5. Type **}** to close the selector.

Your nav selector should look like the following:

```
#nav{
border-bottom: 1px solid #000000;
}
```

Adding the `ul` Element

You will use an inline list for the navigation menu. The first thing you need to define is the `ul` element that holds the list items. Earlier I mentioned the benefits of using zero for default margins to provide a level playing field to start from. This is an important part of working with lists because browsers render them so differently from one another. Some use padding, while others use margins. So let's use zero for the padding and margins of the `ul` element:

1. Open basiclayout.css in Dreamweaver.
2. Place the cursor after the closing "}" of the nav div.
3. Press Enter twice and type **#nav ul{** on the line.

   This opening line of the selector declares that the rules you apply here will apply only to `ul` elements that are a descendant of the #nav ID. If you had simply declared this line as **ul{**, then any rules you determine here would have been applied to *all* `ul` elements wherever they are located on the page. This is also an example of specificity. In short, you are pattern-matching the layout of this document. The selector says, find the nav div and, if any `ul` elements exist within the nav div, style them like this.

   Note: I discussed descendant selectors and pattern matching in Part 1 of this series.

4. Press Enter and type **padding: 0;** to set the padding.
5. Press Enter and type **margin: 0;** to set the margin.
6. Press Enter and type **background-color:#00FF99;** to set the background color.
7. Press Enter and type **}** to close the `ul` element definition.

You do not actually need to type each property out in its entirety. Watch the code hints and select the property and value you need. Once you are used to working with code hints, you will be able to create CSS rules quickly and easily. It is worth persevering with this method of hand coding. Just remember to add a semicolon after each value.

The `#nav ul` selector should look like the following:

```
#nav ul{
padding: 0;
margin: 0;
```

```
background-color:#00FF99;
}
```

Defining How to Display the `li` Element

The next step in the process of creating the navigation is to define how the `li` element (list item element) appears within the `ul` element (list element). CSS makes it quite easy to create a horizontal navigation system. By default, list items are stacked on top of one another. But you can change this quite easily by using the `display` property and the `inline` value. When you apply the `display: inline;` rule, lists are moved to a horizontal plane and bullets are dropped.

You should be pretty much up to speed at coding selectors by now, so I'll show you the next selector and discuss what it does:

```
#nav ul li{
display: inline;
padding: 0;
margin: 0;
}
```

Do you see the use of descendant selectors to pattern-match the document? The opening line finds the #nav ID, sees if it contains a `ul` element, and, if it does, looks within that `ul` element and see if it contains an `li` element; if all these conditions are met, then the line applies the the following rules to the `li` element.

I'm sure you can see now how these selectors are taking shape. You are being very specific about the areas of the XHTML document that you want to affect with your rules. To be specific, you'll use descendant selectors to pattern-match the elements as they appear within the XHTML code.

Besides the `display: inline;` rule, the remaining properties in this selector are zero for the default padding and margin settings, which you have done on many occasions in this article series.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Defining the Link Style

Once again you will use the descendant selector to pattern-match this XHTML document. You confine this link style to any links that reside the `li` element within the nav div.

Play the demo: Defining Your Link's Appearance

1. Place the cursor below the `#nav ul li` selector in your style sheet.
2. Use the code hints to add the `#nav ul li a` selector.
3. Type **#nav ul li a{**
4. Type
5. Type **color: #FFFFFF;**
6. Type **background-color: #3333CC**
7. Type **text-decoration: none;**
8. Type **padding: 0 25px 0 25px;**
9. Type **border-right: 1px solid #000000;**
10. Type **text-align: center;**
11. Type **width: 9em;**
12. Type **}**
13. Hit the Enter key twice
14. Type **#nav ul li a:hover, #nav ul li a:focus {**
15. Type **background-color: #990000;**
16. Type }

Step 4 scales the font size on the `body` selector to 80%. I find this is a good default size for body text. I would set this for `font-size` on the `p` element as well, which you will see shortly. Step 5 removes the underline from links and Step 6 applies padding to either side to give the links a button appearance and make them clickable along their width. Step 7 adds a border to the right of each link, which provides an end to each button and a start to the next one.

IE 5.x for Windows has a problem with the padding when it is set on the link. To combat this, I center the text alignment so the link text is in the center of the button. I also provide a width so that IE 5.x for Windows gives a button-like appearance for each link. If IE 5.x for Windows applied the padding on the link, then the width would not be necessary.

Your completed selector should now look like the following:

```
#nav ul li a{

color: #FFFFFF;
background-color: #3333CC;
text-decoration: none;
padding: 0 25px 0 25px;
border-right: 1px solid #000000;
text-align: center;
width: 9em;
}
```

You're almost done. The final selector in the navigation system allows a change of background color when the user hovers the mouse over the buttons:

```
#nav ul li a:hover, #nav ul li a:focus{
background-color: #990000;
}
```

Earlier in the series I described grouping selectors and separating each one with a comma. This is exactly what you are practicing here. You set a selector that changes the background color of the buttons on hover, or on focus if a user uses Tab to move around the page with a keyboard or some

other pointing device.

Note: Internet Explorer does not support the focus section of this selector. To see it in action, you need to preview it in Firefox or one of the other browsers that provide true standards support.

Add the navigation div into the XHTML code. I've added three list elements and made each of them a link: Home, About Us, and Search:

```
<body>
<div id="wrapper">
<div id="banner"></div>
<div id="nav">
<ul>
<li><a href="#">Home</a></li
><li><a href="#">About Us</a></li
><li><a href="#">Search</a></li
></ul>
</div>

</div>
</body>
```

Lists can be rendered oddly in browsers. Therefore, it is necessary to make them "ugly" to prevent erroneous spaces from being added, which can throw your navigation out of line. This can be achieved in two ways, first by using the method shown above which keeps everything vertical and at hand. Alternatively you can butt the `li` items together on a single line ensuring that no spaces exist in the code between each `li` element.

Notice the non-breaking space in the wrapper div to prevent it from collapsing when you preview your work. You will remove this as soon as you enter the content div into the page:

```
<div id="nav">
<ul>
<li><a href="#">Home</a></li
><li><a href="#">About Us</a></li
><li><a href="#">Search</a></li
></ul>
</div>
```

See how I placed the closing **>** for each `li` item on the *next* line of code? Make sure you do the same because this negates the erroneous spaces that can appear in list menus. Once you have added the code above to the basiclayout.html document, preview it in your browser of choice (see Figure 13).

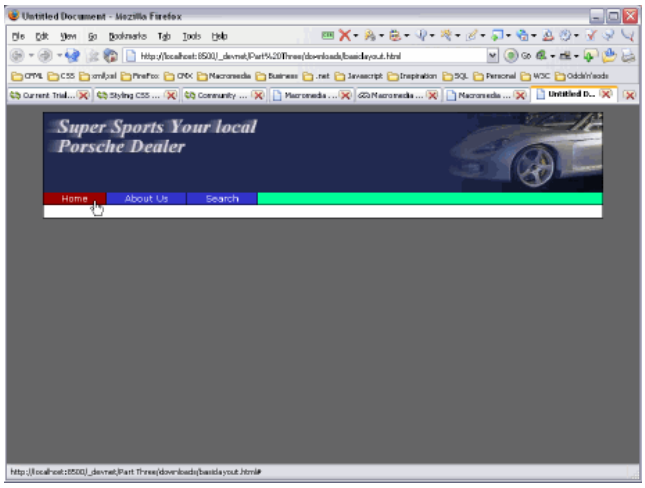

Figure 13. The completed list navigation

In Figure 13 you can see the green background of your `ul` element. It spans the width of its containing element—the wrapper—without having a width of its own declared. This is the default behavior and is worth bearing in mind when laying out your pages in the future.

On the next page you define the content div.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Setting Up the Content Div

The next element to design in the XHTML document is the container for the main content, appropriately named **content**. You can let the div the natural order of the XHTML page:

1. Open basiclayout.css in Dreamweaver.
2. Type **#content p{**
3. Type
4. Type **margin: 20px;**
5. Type **}**

Your content selector should now look like the following:

```
#content p{

margin: 20px;
}
```

Now switch basiclayout.html to the Code view, locate your closing nav div tag, and add the following code immediately below it:

```
<div id="content"></div>
```

Dreamweaver's code hints will help you as you type the div. It really is necessary to switch to the Code view to complete this type of work because the placement of the div within the code is critical.

Add two `p` elements to the content div in the XHTML document. You can use the Lorem Ipsum extension to insert some filler text or you can use any other method to add text. Once you have added your two `p` elements and the filler text, your code should look like the example shown below. In your case add plenty of filler text into the two `p` elements—enough to make them both wrap over several lines:

```
<div id="content">
<p>Some filler text in here</p>
<p>Some filler text in here</p>
</div>
```

Now that you have some real content in the page, you can remove the ` ` element you placed earlier in the wrapper div. The filler text will hold the page open (see Figure 14).

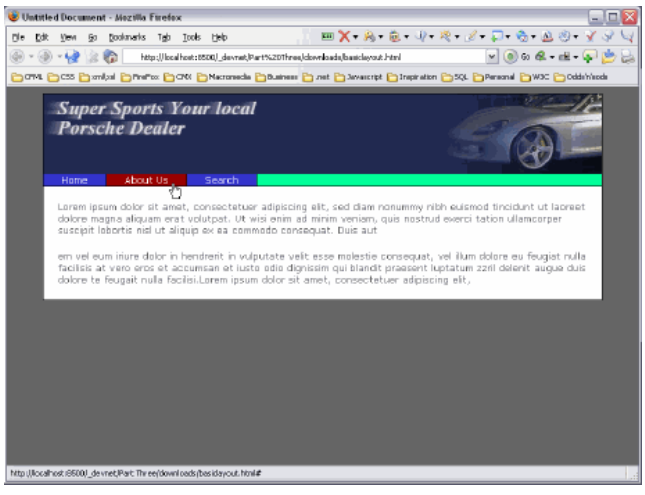

Figure 14. Margin settings around the `p` element

Every page needs a header or two. Create two headers: `h1` for the top of the page and `h2` to go above the second paragraph.

If you need help doing this, you can refresh your memory by reading the discussion in Part 1 (scroll down to the "Defining the h1 Type Selector" section). I suggest you give `h1` a font size of 130% and `h2` a font size of 110%. Don't forget to enter zero for padding. Also, add a margin value of 20 pixels to all four sides to match the `p` element you defined earlier.

Once completed, your two selectors should look like those shown below:

```
#content h1{

color: #003366;
padding: 0;
margin: 20px;
}

#content h2{

color: #003366;
padding: 0;
margin: 20px;
}
```

Insert an `h1` above the first paragraph and an `h2` above the second in basiclayout.html and preview it in your chosen browser (see Figure 15).

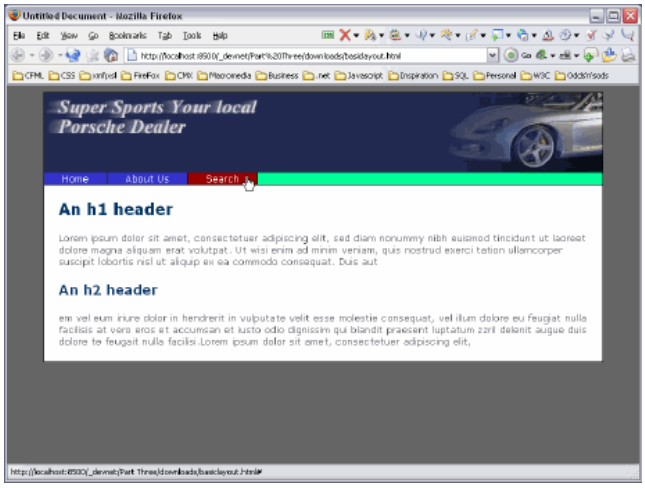

Figure 15. Headers added to the sample text

Adding Floating Images to the Design

To complete the layout of the content area, add a couple of images to accompany the information on the page. In Part 2 of this series I discussed how you can float an image within its containing element. In this instance, that containing element is the `p` element. Look at the amount of text in each `p` element in Figure 15. Make sure you have a similar amount of text (it doesn't need to be absolutely the same, just close enough). You don't want the text to wrap around the image, so keep it fairly short.

Use image1.png and image2.png from the downloaded sample files and create two float classes, one for a left float and another for a right float. If you need a recap on how to do this, review the section, "Working with Floats" from Part 2. When you insert your images in the latest version of Dreamweaver, you will be prompted for accessibility feature information. Go ahead and add alternate text; you can skip the long description if you wish.

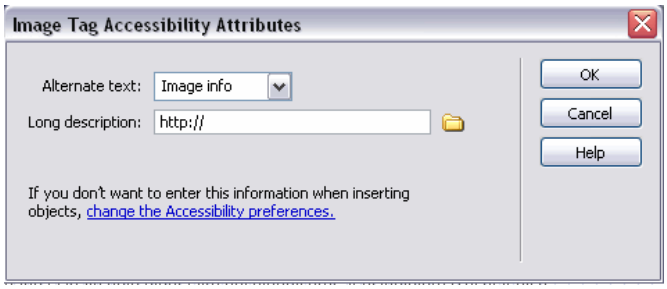

Figure 16. Adding the accessibility information

Once you have created your float classes, they should look like the code below:

```
.leftimage{
float: left;
margin: 0 10px 20px 0;
border: 1px solid #000000;
}

.rightimage{
```

```
float: right;
margin: 0 0 20px 15px;
border: 1px solid #000000;
width: 150px;
}
```

Preview the page in Firefox (see Figure 17). Internet Explorer gets this wrong, so be sure to preview it in Firefox.

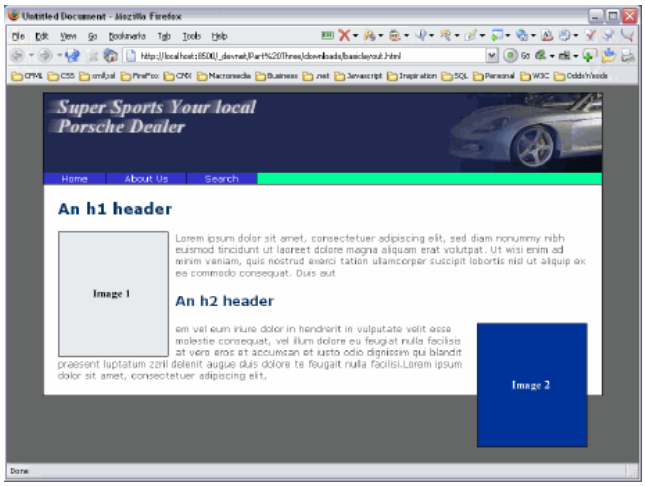

Figure 17. Float element problem rendered correctly in Firefox

When you float an element, it is removed from the document flow, as I discussed in Part 2. Because the two images are floated, the content around them has no bearing on where to be placed. Consequently, the `h2` element moves up alongside Image 1 and Image 2 flows out of the bottom of the content div.

On first glance, this would seem to be a minor disaster, if not a major one. In this instance, you really want to keep each section of text and its accompanying image as a single entity. You don't want the bottom text content flowing up to meet the first set of content, and you certainly don't want the image from the second content section flowing off the page. Let's see how to fix this.

On the next page you will learn how to clear elements to prevent this problem occurring.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Clearing Elements

The following property (and value) makes an element appear below all the content that is above it within the flow of the div:

```
clear: both;
```

Follow the steps below to place a horizontal rule immediately after each closing tag:

1. Open basiclayout.html in Dreamweaver and switch to the Code view.
2. Find the closing tags.
3. Immediately after the closing tags, add the following code: ** **
4. Preview the page in your browser.

The structure of the page has not been altered at all, despite the addition of the new div. The new div will do what you require of it soon enough; you just need to write a new selector for it so it behaves differently.

Before you do this, however, let's examine the `clear` value, which you can set to the following:

- **None** is the default
- **Left** clears all elements to the left
- **Right** clears all elements to the right
- **Both** clears all elements to the left and right

If you look at the top set of content, it becomes immediately apparent that you must clear Image 1. That is, you want to move `h2` below Image 1. Because Image 1 appears to the left of the horizontal rule, applying the property `clear: left;` brings the horizontal rule below Image 1.

This is fine for the top section but what about the bottom section? Image 2 appears to the right in this instance, so you need to set the property as `clear: right;` instead. To avoid creating two classes to meet two different scenarios, use the `both` value on the `clear` property.

To add this selector to the basiclayout.css file, complete the following steps:

1. Open basiclayout.css in Dreamweaver.
2. Press Shift+F11 to open the CSS Styles panel.
3. Click the New CSS Style button at the bottom of the panel.
4. Click the Class option in the Selector Type area of the New CSS Style dialog box.
5. Type **.clearit** in the Name text box.
6. Click OK. This opens the CSS Style Definition dialog box.
7. Select the Box option in the Category list.
8. From the Clear pop-up menu select Both.
9. Select the Block option in the Category list.
10. From the Display pop-up menu select Block.
11. Click OK to close the dialog box.

The newly added selector for `clearit class` looks like the code below:

```
.clearit{
clear: both;
}
```

Once you add the `clearit` class to the basiclayout.css file, you can apply it to the two divs you added immediately after the closing tags. To do this accurately, switch to the Code view, locate your opening div tag, and place your cursor between the `v` and the closing `>` of the opening div tag. Press the Spacebar and start typing **class**. The code hints will kick in, from which you can select a class.

Internet Explorer 6 has some float bugs that are beyond the scope of this article. Although these bugs will not break your page, they will prevent it from being pixel-perfect.

Note: John Gallant and Holly Bergevin have written an excellent series of tutorials for Community MX that are dedicated completely to float theory and float bugs: Float: The Theory, Float: The Bugs (Part 1), and Float: The Bugs (Part 2). You can sign up for a free 10-day trial to access these tutorials if you want a complete overview of float bugs and how to fix them.

Now preview basiclayout.html in your browser (see Figure 18).

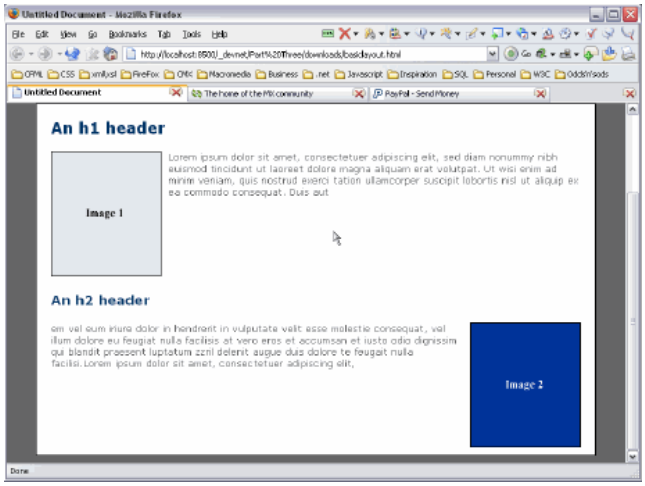

Figure 18. A clearly defined page structure

On the next page you will finish this design by adding a footer below the content div.

Dreamweaver Article

Designing with CSS – Part 3: Creating Your First Design Without Tables

Table of Contents

Writing the Footer Selector

Make the footer selector the same way you created the other selectors. The footer will also be an ID selector, so it will begin with the pound

Earlier in this series you created the nav div and gave it a bottom border. Provide a border for the footer div, except this time, make it a top border rather than a bottom border. You can keep the border black if you want. Add two other property and value pairs to this selector: Give the footer a background color of #003366 and a text color of white.

The completed selector for your footer should like the following:

```
#footer{
border-top: 1px solid #000000;
background-color: #003366;
color: #FFFFFF;
}
```

The next step is to add the footer div to the basiclayout.html page. Locate the closing wrapper div tag in the Code view and add the footer div immediately above it, as shown in the code below:

```
</div>
<div id="footer"></div>
</div>
</body>
</html>
```

Adding Text to the Footer Div

This is the final step. You will undoubtedly want to add text to your footer, so define how that will look with regard to its size. Most importantly, set specific margins and padding values to ensure you don't use the default settings, which can vary greatly from browser to browser, as I discussed earlier.

Because the text in your footer resides in a `p` element, you need to reflect this in your selector. You can do this by using the descendant selector. Set the default margins to zero, provide a little room around the text by giving it a padding value of 10 pixels, and make the text 70% of the size you set against the body.

Open basiclayout.css in Dreamweaver and set the following styles:

- **#footer p{** is the descendant selector, which affects only the `p` elements that reside within the footer div
- **font-size: 70%** scales the font size to 70% of the default value set in the body selector
- **padding: 3px;** sets a value of 10 pixels for the padding on the `p` element

Locate the footer div in basiclayout.html and add the `p` element as shown below:

```
<div id="footer"><p>My company info</p></div>
```

Preview the page in your chosen browser (see Figure 19).

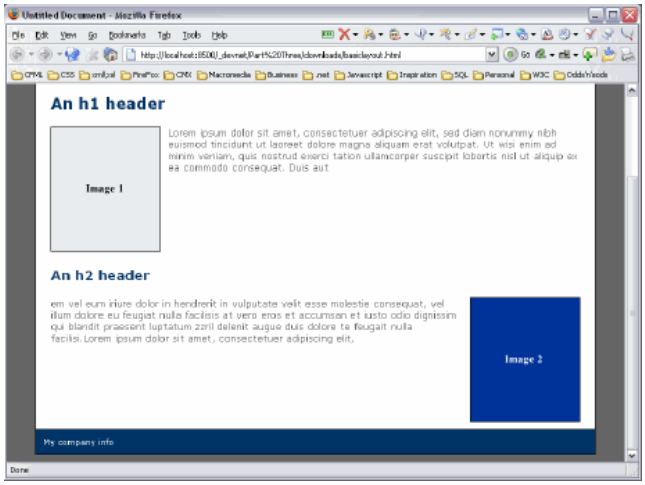

Figure 19. The completed design without tables

Conclusion

You have just completed your first design without using tables. While this is a fairly simple layout, you have learned some new techniques to add to your design armory. Creating layouts with CSS positioning is not harder than working with tables, it is just different. Sometimes that means learning some new skills to complete common tasks differently.

Once you become more familiar with CSS positioning and many of these tasks become second-nature, you will find that designing with CSS is much nicer than designing with tables. Designing with CSS has a lot of advantages with regard to layout control and the appearance of your site.

Designing with CSS – Part 4: Creating a Two-Column Layout

Dreamweaver Article

Designing with CSS – Part 4: Creating a Two-Column Layout

Adrian Senior

www.webade.co.uk
www.communitymx.com
www.ukcsstraining.co.uk

Table of Contents

FEEDBACK

Created:
6 December 2004
Modified:
12 September 2005
User Level:
Beginner

Note: This article has been updated for Dreamweaver 8. If you are still using Dreamweaver MX 2004, please read the version of this article series for Dreamweaver MX 2004. The CSS features in Dreamweaver have been vastly improved in Dreamweaver 8. You can learn about those changes in Julie Hallstrom's article, An Overview of CSS in Dreamweaver 8.

Welcome to Part 4 of this article series on CSS design concepts. If you missed Parts 1, 2, or 3, you can get to them below:

Designing with CSS – Part 1: Understanding CSS Design Concepts

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Designing with CSS – Part 3: Creating Your First Design Without Tables

This series explains how you can use Dreamweaver 8 to move towards using CSS as a positioning technique when developing web pages. In Parts 1 and 2 you investigated how to use some of the techniques common to most designs that use the CSS positioning technique. In Part 3 you implemented those skills to create your first CSS design. You have seen how to use the Dreamweaver panels to create ID, class, and tag selectors; and you know how to use code hints to write these same selectors directly into the style sheet without using any panels at all. The Dreamweaver panels are fine but there is no substitute for becoming intimately familiar with the syntax of CSS.

In Part 4 you pick up from where you left off in Part 3. It's time to modify the layout of your design by easily making dramatic structural changes to your basiclayout.html page. If you wish, download the files from the link below or simply open your existing files in Dreamweaver and pick up from where you left off (at the end of Part 3).

You may find it advantageous to read an earlier tutorial I wrote for the Developer Center on relative, absolute, and static positioning, called Introduction to CSS Positioning in Dreamweaver MX 2004. It should give you a decent grounding in what you can achieve with each of the positioning values.

On with the show!

Requirements

To complete this tutorial you will need to install the following software and files:

Dreamweaver 8

- Try
- Buy

Sample files:

- css4_samples.zip (ZIP, 237K)

Prerequisite knowledge:

- Lorem Ipsum Generator extension by Captain Cursor Creations (optional)
- Internet Explorer browser
- Firefox browser
- Read Part 1 of this series
- Read Part 2 of this series
- Read Part 3 of this series

About the author

Adrian Senior owns the UK-based web design agency Webade, which has been in business since 1998. Adrian also provides courses for companies and individuals who need to build compliant, accessible websites using CSS and XHTML. He is also an Adobe Community Expert and a partner at Community MX. The year 2004 saw Adrian's first trip to America, where he visited Orlando and delivered two sessions at the TODCon conference.

Before becoming involved with website design and development, Adrian's working life centered around the shipyards of the River Mersey and oil production units in the North Sea. Adrian has been married to his wife, Janette, for 26 years. They have two children, Antony and Eleanor.

Dreamweaver Article

Designing with CSS – Part 4: Creating a Two-Column Layout

Table of Contents

Preparing Your Page for Debugging

We investigated the float technique earlier in this series and now it's time to use it to create a column in your design. Floating an image within a paragraph allows the text to move up alongside the image. There are two examples of this in the basiclayout.html file: one immediately below the `h1` title and another immediately below the `h2` title. Creating the second column works the same way: When you float the div that becomes the left column, the main content div can sit alongside it.

I discussed floats in depth in the Working with Floats section of Part 2 of this article. For the right-hand column, you will use the content div you created in basiclayout.html. The wrapper div acts as the parent container rather than the `p` element in the example shown in Part 2.

It is good practice to give your divs distinct background colors when laying out your design. This gives you a precise view of exactly what is going on in your page at even the quickest of glances. I find that using "debugging colors" in divs is an excellent way to monitor exactly what is going on in your page.

Dreamweaver 8 has a fantastic new feature that can do this for you at the click of a button—no more manually adding debugging colors for Dreamweaver 8 users! By accessing the “Visual Aids” icon on the document toolbar, you can simply switch on background colors and Dreamweaver 8 handles the rendering for you.

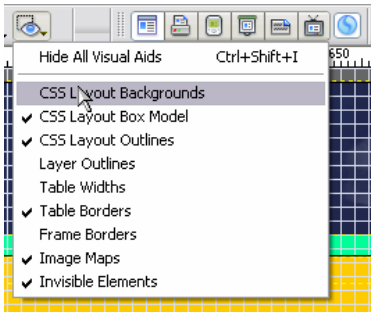

Figure 1. Switching on the CSS Layout Backgrounds

The CSS Layout Background colors are applied at random, in this instance you can see that our content div has been given a green background color.

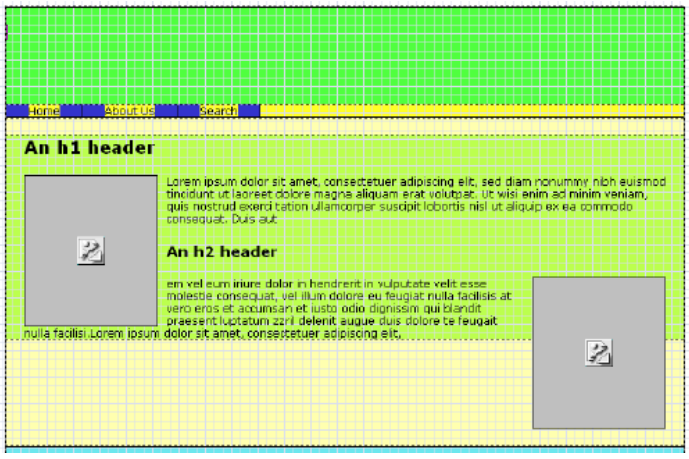

Figure 2. Dreamweaver 8 in CSS Layout Backgrounds mode

The screen shot in Figure 2 is taken in the Dreamweaver 8 Design view. Notice how Image 2 extends outside the content div, and how easy it is to see the boundaries of the content div now that it is highlighted by the CSS Layout Backgrounds.

If you give the content div a background color of red—or any color of your choice—and preview this page in Microsoft Internet Explorer, you will see that the content div expands to include the height of Image 2. This is an example of IE getting the CSS specification for floats absolutely wrong.

The cause of the white space above the green content div background you see in Figure 2 is from the margin setting on the `#content` h1 selector. Change the value of that property to see how it affects your page layout. The new CSS features in Dreamweaver 8 are great, though you should always test in as many browsers as you can. One of my favorite new CSS features in Dreamweaver 8 is the "CSS Layout Outlines." I have this feature switched on permanently so I can see exactly what is going on in my page as I design.

On the next page you will return to the basiclayout.css file and add a div to create a left column.

Dreamweaver Article

Designing with CSS – Part 4: Creating a Two-Column Layout

Table of Contents

Adding the Left Column

Open the basiclayout.css file and create space at the bottom to add a new selector. Once you do this, complete the following steps:

1. Type **#leftcol{** and press Enter.
2. Type **width: 170px;** and press Enter.
3. Type **height: 150px;** and press Enter.
4. Type **background-color:** and select yellow from the color palette.
5. Type **;** and press Enter.
6. Type **}** and press Enter.

Your newly created selector should look like the code below:

```
#leftcol{
  width: 170px;
  height: 150px;
  background-color: #FFFF00;
  }
```

Note: As I stated in Part 2 , use the `height` property with caution. I am using it here only to open up the leftcol div so that you can see where it is on the page and how much space it occupies.

Near line 24, in the basiclayout.html file, you will see the opening content div tag, and just above this you will see the closing nav div tag. Make some space between the two and add the leftcol div (see Figure 2). The leftcol div should be in between the closing nav div tag and the opening content div tag, as you can see commented in Figure 2.

```
19    <!-- end nav -->
20  </div>
21  <div id="leftcol"></div>
22  <!-- start content -->
23  <div id="content">
24    <!-- Begin the top content containe
25    <div class="container">
26      <h1>An h1 header</h1>
```

Figure 3. Adding the leftcol div

Once you have done this, save your work and preview it in your browser (see Figure 3).

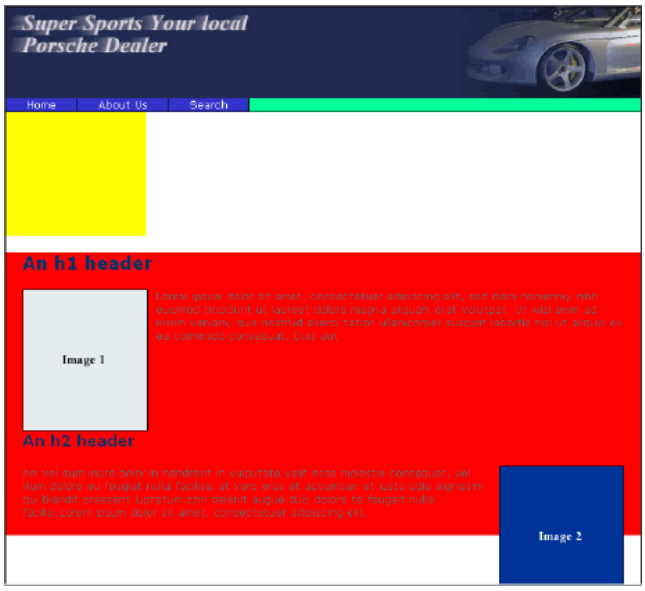

Figure 4. The yellow leftcol div added above the content div

This is not exactly what you were after! Figure 3 shows you the leftcol div in its natural position within the flow of the document. Once you add the float, the leftcol div will be removed from the document flow and the content div will move up as if the leftcol div doesn't exist.

Adding the Float

Return to the basiclayout.css file and locate the `#leftcol` selector. By now you should be able to quite easily add a `float` property to this selector and give it a value of **left**. The selector should now look like the following:

```
#leftcol{
  float: left;
  width: 170px;
  height: 150px;
  background-color:#FFFF00;
  }
```

Save your CSS file and preview your page in the browser (see Figure 4).

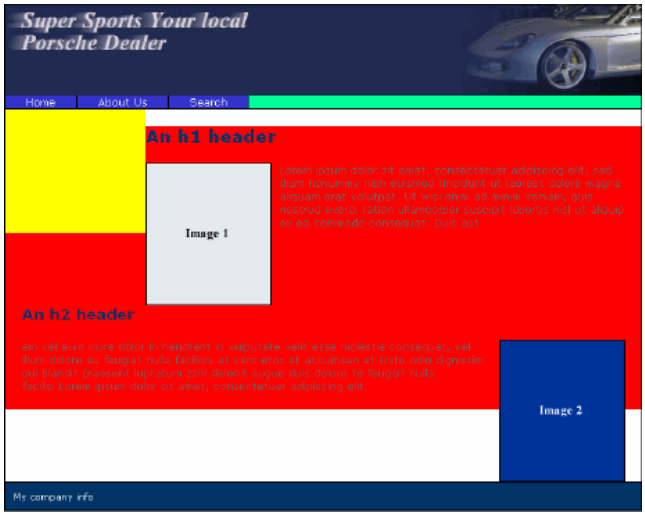

Figure 5. The yellow leftcol div within the content div

In Figure 4 the content div has now moved up, despite the fact that the leftcol div sits above it in the XHTML code. The leftcol div has been moved out of the document flow and has no effect on the content div.

Of course, the banner precedes the leftcol div in the XHTML code, so the fact that the leftcol div is floated has no bearing on that element. If you placed the leftcol div above the banner in the XHTML code, you would see an entirely different behavior because then the leftcol div would also be floated around the banner. Move the leftcol div above the banner div in the XHTML code and see what happens to the page layout in the browser.

Dreamweaver Article

Designing with CSS – Part 4: Creating a Two-Column Layout

Table of Contents

Organizing the Columns

At the moment you have the two columns roughly in position, although you still have some work to do to give them a proper look and feel. Aligning the two columns can be a little tricky but with your colored backgrounds in place you can see clearly where each of the two divs—leftcol and content—sit on the page.

Let's summarize the problems we have at the moment:

- **Point 1: The leftcol div sits higher than the content div.** The margin setting on the `h1` element within the content div controls this. Y to match that margin and apply it to the top of the leftcol div.
- **Point 2: The leftcol div needs to move away from the edge of the wrapper div.** A div contains no default margins, so if you want to use margins to move a div away from an adjacent element, you must declare them explicitly.
- **Point 3: The content div runs underneath the leftcol div.** Because there is no width or margins set on this div, it assumes by default the width of its container, the wrapper div. You need to control the position of the content div.
- **Point 4: There aren't two distinct columns.** This is because the lower section of the content div wraps below the leftcol div. This is to be expected. The div is not constrained by margins or width settings so it flows to suit its container. When you correct Point 3, you will fix this issue.

Let's fix Point 1 first. To correct the top alignment of the leftcol div, give the selector a top margin of 20px to match the top margin of the `h1` selector in the content div:

1. Open the basiclayout.css file and locate the `#leftcol` selector.
2. Place the cursor after the opening `{` and press Enter.
3. Type **margin-top: 20px;** (remember to watch the code hints as you type).
4. Save your work.

5. Preview your page in the browser (see Figure 5).

Figure 6. The yellow leftcol div taking up its position vertically

Next, let's fix Point 2. To move the leftcol div away from the edge of the wrapper div, follow these steps:

1. Locate the `#leftcol` selector.
2. Place the cursor after the opening `{` and press Enter.
3. Type **margin-left: 10px;** (remember to watch the code hints as you type).
4. Save your work.

5. Preview your page in the browser (see Figure 6).

Figure 7. A space of 10 pixels added to the left of the yellow leftcol div

Now you are left with Points 3 and 4 to correct, both of which involve modifying the `#content` selector. However, fixing Point 3 also fixes Point 4. Your first task requires moving the content div out from underneath the leftcol div. To do this you need to set a left margin within the `#content` selector, but by how much? A little addition quickly tells you how much margin you need:

- From the edge of the wrapper to the leftcol div you have 10 pixels
- The leftcol div is 170 pixels wide
- You want 20 pixels of space on the right of the leftcol div before you see your content in the content div

From this you can easily deduce that you need a margin of 10 + 170 + 20 (or 200) pixels. This gives you enough distance to clear the leftcol div and allows enough room to keep your content a nice distance away from the leftcol div's right edge. These margins are, of course, personal preferences. Change them to suit your requirements.

To set the left margin on the content div, complete the following steps:

1. Locate the `#content` selector.
2. Place the cursor after the opening `{` and press Enter.
3. Type **margin-left: 200px;** (remember to watch the code hints as you type).
4. Save your work.

5. Preview the page in your browser (see Figure 7).

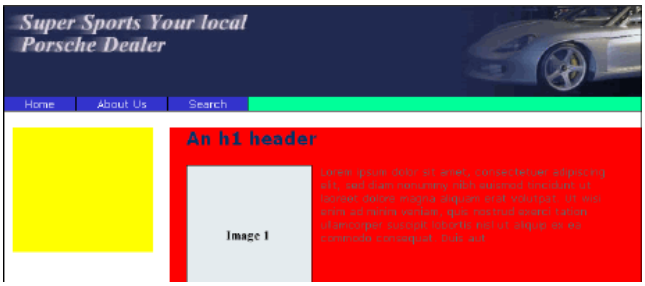

Figure 8. The content div's left margin properly set—things are now taking shape!

Everything looks great. As you can see, you now have two distinct columns—which was the aim of this tutorial. However, there's a pitfall waiting just around the corner to trip you up. Locate the leftcol div in your basiclayout.html code and place the cursor between the opening and closing div tags (see Figure 8).

```
19    <!-- end nav -->
20  </div>
21  <div id="leftcol"></div>
22  <!-- start content -->
23  <div id="content">
```

Figure 9. Placing your cursor between the div tags

Insert some dummy content into the leftcol div. Hold down the Shift key and press Enter. This inserts a line of
 tags into the leftcol div. The idea here is to make the contents of the leftcol div longer (or taller) on the page than the contents of the content div. Add enough
 tags to the leftcol div to make that happen. Before you preview your page, go to the basiclayout.css file and complete the following steps:

1. Locate the `#leftcol` selector.
2. Within this selector locate the `height` property and value pair. It should look like this: `height: 150px;`
3. Delete it.
4. Save your CSS file.

Note: Just to make things quite clear, I added a background color to the `#wrapper` selector. As you can see in the following figures, I colored the selector orange. Follow the steps you already completed for giving the `#leftcol` selector a background color and apply them to the `#wrapper` selector.

Now preview the basiclayout.html page in your browser (see Figure 9). I added a non-breaking space into the clearing divs set in Part 3 of this tutorial and colored them green so you could see them easily.

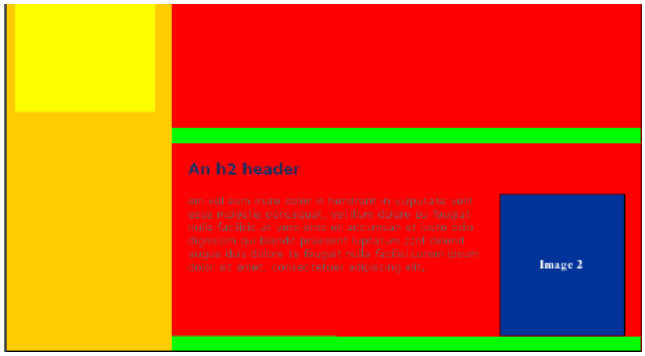

Figure 10. The top clearing div clearing the leftcol div—but dropping out content

This is not what we wanted at all. What is going on here?

The problem occurs because I didn't clear the divs: Although they may look like they are inside the content div, they are not. They force the content div down until they can occupy a line in the browser window on their own. The position of the first clearing div is above the `h2` header; below the left column, the second clearing div drops into the layout immediately below Image 2. Clearing the content in this manner will no longer work in this scenario.When you clear a float, you clear all floats—consequently all content below the first clearing div moves down beyond the floated left column.

On the next page you will see how to correct this problem.

Dreamweaver Article

Designing with CSS – Part 4: Creating a Two-Column Layout

Table of Contents

Floating the Content

Your first job here is to locate the the two clearing divs in the basiclayout.html page and delete them.

To correct this problem, you will have to encase your content sections in a containing div. Because you will be using this selector more than once on your page, make it a class:

1. Open the basiclayout.css file and create some empty space at the bottom.
2. Type **.container{** and press Enter.
3. Type **width: 99%;** and press Enter.
4. Type **float: right;** and press Enter.
5. Type **background-color: #6633CC;** and press Enter.
6. Type }.
7. Save your work.

FEEDBACK

In Step 4 you float to the right the containers to which you apply this class. By doing this you avoid the problems caused by the clearing divs moving your content down below the left column. In Step 3 the width of 99% opens up the containers to almost the complete width of the content div. By not using a full width of 100% you avoid any possibility of the main content dropping down to find room to open to the given width. This scenario can occur if the browser introduces rounding errors into the page width.

Note: John Gallant and Holly Bergevin have written an excellent series of tutorials for Community MX that are dedicated completely to float theory and float bugs: Float: The Theory, Float: The Bugs (Part 1), and Float: The Bugs (Part 2). I really can't implore you enough to read these tutorials. If you don't want to join Community MX, take advantage at least of their free 10-day trial to access the site. John and Holly are at the top of the CSS game, so read anything they've written about it. It will open your eyes to what you can achieve.

Because you now have elements floated both left and right, make an adjustment to the footer div:

1. Open the basiclayout.css file.
2. Locate the `#footer` selector.
3. Place the cursor immediately after the opening `{` and press Enter.
4. Type **clear: both;**
5. Save your work.

The `clear: both` property/value ensure that your footer is moved down below the content and leftcol divs, no matter which of these two elements is longer. (After you complete this tutorial, try changing the `clear` value to **left** and preview your page. Then, change it to **right** and preview your page again. This will give you a great visual example of how the `clear` value works.)

Now that you have created your class for the containing div, open basiclayout.html in the Code view. Take time to study the code shown in Figure 10. I entered a new div with the container class around each section of content (you will need to hand-code these two divs).

```
23    <!-- start content -->
24    <div id="content">
25      <!-- Begin the top content container class div -
26      <div class="container">
27        <h1>An h1 header</h1>
28        <p><img src="tutorial_images/image1.gif" width
ipsum dolor sit amet, consectetuer adipiscing elit,
laoreet dolore magna aliquam erat volutpat. Ut wisi
 ullamcorper suscipit lobortis nisl ut aliquip ex ea
29      <!-- End the top content container div -->
30      </div>
31      <!-- Begin the bottom content container class di
32      <div class="container">
33        <h2>An h2 header</h2>
34        <p><img src="tutorial_images/image2.gif" width
eum iriure dolor in hendrerit in vulputate velit ess
feugiat nulla facilisis at vero eros et accumsan et
luptatum zzril delenit augue duis dolore te feugait
consectetuer adipiscing elit,</p>
35        <!-- End the bottom content container div -->
36      </div>
37      <!-- end content -->
38    </div>
```

Figure 11. One div highlighted green, the other blue

Once you make these changes to the basiclayout.html file, preview it in your browser (see Figure 11).

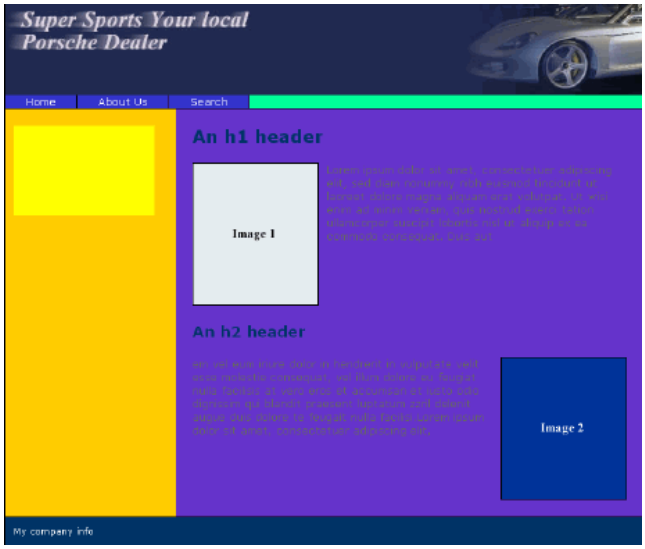

Figure 12. The footer div moved below the content div—you're all done!

You can add a string of
 tags to force the height of the left column (see Figure 12).

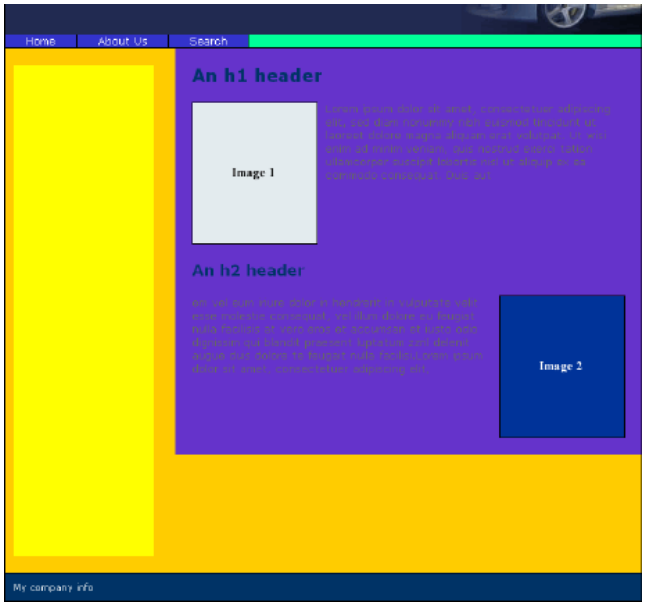

Figure 13. The footer div moved below the leftcol div

Exercise

Remove all the
 tags from the leftcol div and create content that is specific to that div by using a descendant selector. Create `h1` and `p` selectors for that div and make sure your new selectors do not affect the content in the main area of your layout.

In Part 5, I show you how to remove the horizontal navigation and move it to the leftcol div. Best of luck implementing your `h1` and `p` elements in the leftcol div. I'm sure you will do just fine.

Designing with CSS – Part 5: Defining Columns and Vertical List Navigation

Dreamweaver Article

Designing with CSS – Part 5: Defining Columns and Vertical List Navigation

Adrian Senior

www.webade.co.uk
www.communitymx.com
www.ukcsstraining.co.uk

Table of Contents

FEEDBACK

Created:
17 January 2005
Modified:
12 September 2005
User Level:
Beginner

Note: This article has been updated for Dreamweaver 8. If you are still using Dreamweaver MX 2004, please read the version of this article series for Dreamweaver MX 2004. The CSS features in Dreamweaver have been vastly improved in Dreamweaver 8. You can learn about those changes in Julie Hallstrom's article, An Overview of CSS in Dreamweaver 8.

Welcome to Part 5 of this article series on CSS design concepts. If you missed the earlier tutorials in this series you can access them from the links below:

Designing with CSS – Part 1: Understanding CSS Design Concepts

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Designing with CSS – Part 3: Creating Your First Design Without Tables

Designing with CSS – Part 4: Creating a Two-Column Layout

This series reviews how you can use Dreamweaver 8 to move toward using CSS as a positioning technique when developing web pages. Parts 1 and 2 showed you how to use some of the techniques common to most designs that use the CSS positioning technique. Part 3 described creating your first CSS design and Part 4 introduced you to using the float technique to create a two-column layout.

In Part 5, you will use the design you created in Part 4. However, you will make some navigation changes. You will remove the horizontal list and replace it with a vertical navigation so that the new navigation system resides in the left column you added to the design in Part 4.

Make backups of your current layout if you intend to use it in any future work. By the end of Part 5 you will have two very different layout structures that you can customize to suit your own needs.

We have covered the various options that Dreamweaver affords when laying out a design with CSS. You saw how to use panels to create ID, class, and tag selectors, and you also saw how to use code hints to write the same selectors directly into your style sheet without using Dreamweaver panels at all. These are two very different techniques. The latter should be your goal. Using Dreamweaver panels is fine but there is no substitute for becoming familiar with the syntax of CSS.

If you wish, download the files from the link below or simply open your existing files in Dreamweaver and pick up from where you left off, at the end of Part 4.

You may find it advantageous to read an earlier tutorial I wrote for the Developer Center on relative, absolute, and static positioning, called Introduction to CSS Positioning in Dreamweaver MX 2004. It should give you a decent grounding in what you can achieve with each positioning

value.

Note: You may find a series of CSS layouts, known as JumpStarts, that I wrote for Community MX, of interest. It's called the North Pole JumpStart.

On with the show!

Requirements

To complete this tutorial you will need to install the following software and files:

Dreamweaver 8

- Try
- Buy

Fireworks 8

- Try
- Buy

Sample files:

- css5_samples.zip (ZIP, 238K)

Prerequisite knowledge:

- Lorem Ipsum Generator extension by Captain Cursor Creations (optional)
- Internet Explorer browser
- Firefox browser
- Read Part 1 of this series
- Read Part 2 of this series
- Read Part 3 of this series
- Read Part 4 of this series

About the author

Adrian Senior owns the UK-based web design agency Webade, which has been in business since 1998. Adrian also provides courses for companies and individuals who need to build compliant, accessible websites using CSS and XHTML. He is also an Adobe Community Expert and a partner at Community MX. The year 2004 saw Adrian's first trip to America, where he visited Orlando and delivered two sessions at the TODCon conference.

Before becoming involved with website design and development, Adrian's working life centered around the shipyards of the River Mersey and oil production units in the North Sea. Adrian has been married to his wife, Janette, for 26 years. They have two children, Antony and Eleanor.

Dreamweaver Article

Designing with CSS – Part 5: Defining Columns and Vertical List Navigation

Table of Contents

Moving the Navigation

At the end of Part 4, I set a little exercise for you to add an `h1` title and `p` elements into the leftcol div. How did it go, OK? Hopefully you completed it just fine by implementing descendant selectors in much the same way as you have already covered in the series.

You can delete all that information from your page for the time being. If you never bothered with the exercise and you left the temporary
 tags in the leftcol div, then please delete them now.

Making Structural Changes in the HTML Document

Open basiclayout.html in the Code view of Dreamweaver. Because you need to make some precise selections within your code, the Design view really won't do. Code view shows you exactly what you are doing.

You need to perform the following tasks, which I have listed below. You can also view them in the presentation linked below the list.

1. Locate the Start and End comments for the nav div.
2. Highlight them and all the code in between.
3. Select Edit > Cut (Control+X) to cut the code to the Clipboard.
4. Locate the leftcol div.
5. Place your cursor between the opening and closing div tag.
6. Select Edit > Paste (Control+V) to paste the code into the leftcol div.
7. Open the basiclayout.css file.
8. Select Edit > Find and Replace (Control+F) to open the Find and Replace dialog box.
9. Type **#nav** into the Find box.
10. Type **#leftcol #nav** into the Replace box.
11. Click the Replace All button.
12. Save your work.

Play the demo: Changing the Page Structure

Once you complete the changes above, you can preview the page in your browser of choice (see Figure 1).

Figure 1. Navigation now inside the leftcol div

As you can see from Figure 1, the navigation div now resides in the leftcol div. You still have some styling to do but I'm sure you would agree that making this big change was relatively painless in CSS. If you had been using tables to lay out your design, this alteration would have required a major redesign of the table structure. Notice how the other elements on the page simply moved up to fill the void left by by moving the nav div.

Let's embark on a little tidying up of the debugging colors:

1. Open basiclayout.css in Dreamweaver.
2. Locate the `#content` selector.
3. Delete the `background-color` property and its value.
4. Locate the `.container` class selector.
5. Delete the `background-color` property and its value.
6. Locate the `#wrapper` selector.
7. Change its `background-color` value to **#ffffff** (white).
8. Locate the `#leftcol` selector.
9. Delete the `background-color` property and its value.

Preview the page again in your browser of choice (see Figure 2).

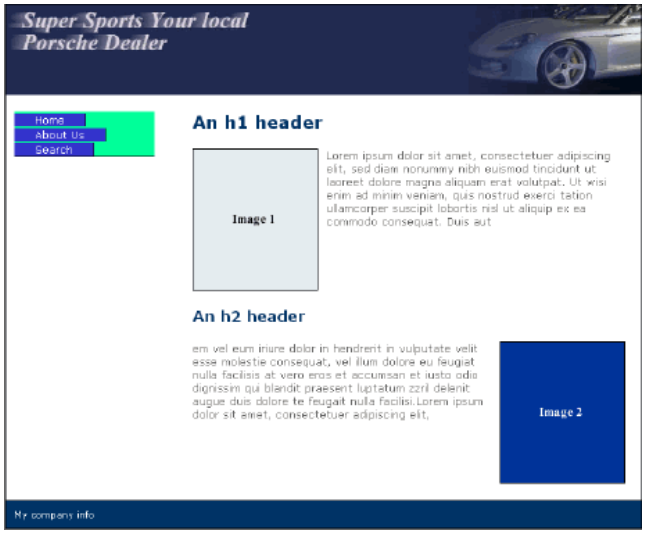

Figure 2. Colors tidied up on the page

Leave the background color you set on the `ul` element for now. Next we will begin to restyle the unordered list to give it a completely new look.

Dreamweaver Article

Designing with CSS – Part 5: Defining Columns and Vertical List Navigation

Table of Contents

Restyling the Unordered Navigation List

Begin by making changes to your `#leftcol #nav ul` selector. It should currently look similar to Listing 1.

```
#leftcol #nav ul{
padding: 0;
margin: 0;
background-color:#00FF99;
}
```


Listing 1. `#leftcol #nav ul` selector code

Change the code in Listing 1 so it resembles the version in Listing 2.

```
#leftcol #nav ul {
margin: 0;
padding: 0;
background-color: transparent;
list-style-type: none;

border: 1px solid #000000;
}
```

Listing 2. Revised `#leftcol #nav ul` selector code

Open basicalyout.css in Dreamweaver, locate the `#leftcol #nav ul` selector, and complete the following steps. Remember that while you type in the style sheet, your CSS code hints are active. Take advantage of them and monitor what they are offering as you type. Code hints speed your workflow dramatically:

1. Locate the `background-color` value delete it and type **transparent**.
2. Place your cursor after `;` and press Enter.
3. Type **list-style-type: none;** and press Enter. When you set the value of this property to `none` it removes the bullets from the list items.
4. Type and press Enter. The percent value you set here scales the font size based on the default percentage you set in the `body` selector.
5. Type **border: 1px solid #000000;**. This shorthand code sets the border for all four sides in accordance with the given values. In this case the borders will be one pixel wide, solid, and black.
6. Save your work.

Making Changes to the Link Styles

To change the default, or inactive, state of the links in your new navigation system, you first need to locate the `#leftcol #nav ul li a` selector (see Listing 3). Quite a selector name, isn't it? Earlier I discussed the advantages of being very specific with your selectors; this is an ideal time to review that discussion and refresh your memory. By using descendant selectors, you can pattern-match your selectors right down through the markup of your XHTML document. This gives you much flexibility. If you had set this selector as simply `#leftcol #nav`, then every link in your nav div would have taken on the same appearance. Surely you wouldn't have wanted that.

```
#leftcol #nav ul li a{

color: #FFFFFF;
background-color: #3333CC;
text-decoration: none;
padding: 0 25px 0 25px;
border-right: 1px solid #000000;
text-align: center;
width: 9em;
}
```

Listing 3. `#leftcol #nav ul li a` selector code

The list navigation is going to be styled to make it react in the same manner as a button graphic might. Any links styled like that, which were not a part of the main navigation, would have looked very odd indeed. By being specific, however, you can create a totally different link style that sits in the nav div but outside the `ul` element. Change this selector to reflect the code in Listing 4.

```
#leftcol #nav ul li a{
```

```
background-color: #869BCC;
border-bottom: 1px solid #000000;
color: #000000;
display: block;
padding: 4px 0 6px 4px;
text-decoration: none;
height: 1%;
}
```

Listing 4. Revised `#leftcol #nav ul li a` selector code

To make the necessary changes, complete the following steps:

1. Locate the `font-size` property and value and delete them.
2. Locate the `background-color` property and change its value to **#869BCC**.
3. Locate the `border-right` property and change it to **border-bottom**.
4. Locate the `padding` property and change the values from `0 25px 0 25px` to **4px 0 6px 4px**.
5. Locate the `text-align` property and delete it and its value of `center`.
6. Locate the `width` property and delete it and its value of `9em`.

Add two new property and value pairs to this selector:

- `display: block;`
- `height: 1%;`

The `display: block;` property makes your links act like buttons. The link will be clickable along its entire width—except in Microsoft Internet Explorer for Windows. To work around this problem, I deployed the "Holly Hack" and gave the link a height of 1%, which makes IE behave in the proper manner.

Note: Read more about the Holly Hack on CommunityMX, where none other than Holly Bergevin herself explains how it works.

Save your work and preview basiclayout.html in your favorite browser (see Figure 3).

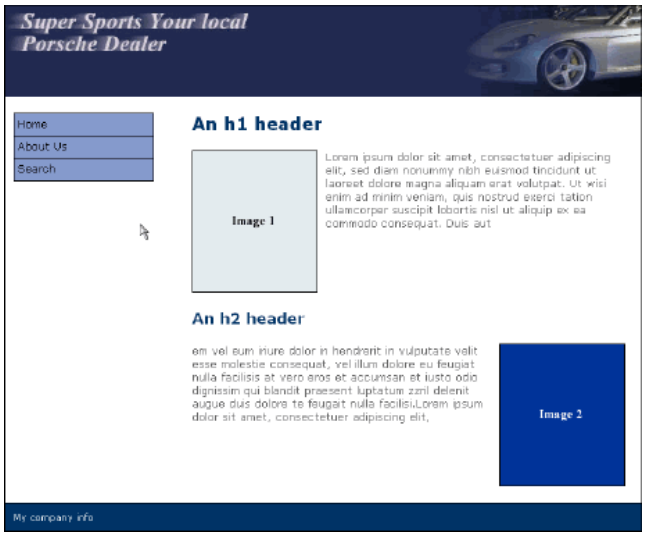

Figure 3. Navigation where links act like buttons

Next you will add the `hover` and `focus` styles.

Dreamweaver Article

Designing with CSS – Part 5: Defining Columns and Vertical List Navigation

Table of Contents

Adding Hover and Focus Styling

With the hover and focus styles you are going to use a little border trickery to make the links act more like buttons. You can change the border color on `hover` and `focus` to make the links appear as if they really are buttons being depressed (see Listing 5).

```
#leftcol #nav a:hover, #leftcol #nav a:focus {
background-color: #003366;
border-right: 1px solid #ffffff;
border-bottom: 1px solid #ffffff;
color: #ffffff;
}
```


Listing 5. Using hover and focus styles

Before you implement the changes shown above, take a quick look at the selector, which is set for both the hover and focus states. Each selector has been stated immediately before the opening `{` and each selector is separated by a comma. You saw this earlier when I discussed grouping selectors; that is all that's going on here. You are likely familiar with the hover state; this is the state the link changes to when the mouse or pointing device is positioned over the link. The focus state occurs when an alternative to a pointing device is used to navigate your pages, such as a keyboard.

First complete the selector; then you can see how the focus state works:

1. Locate the selector and change the `background-color` value to **#003366;**.
2. Add a property and value for the right border by typing **border-right: 1px solid #ffffff;**
3. Add a property and value for the bottom border by typing **border-bottom: 1px solid #ffffff;**
4. Make the `color` value **#ffffff;**
5. Save your work.

Preview the page in your browser of choice (see Figure 4).

Figure 4. Hover section of the selector in action

Below, Figure 5 shows how the light-colored—in this case, white—borders to the right and bottom enhance the depressed effect of the button when it is in the hover state.

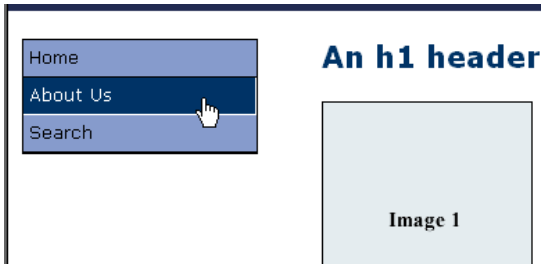

Figure 5. Hover effect in detail

To see the focus side of the selector in action, press the Tab key on your keyboard to hop from link to link. You can determine the order of the links visited by using `tabindex` in each link. This provides an ordered navigation sequence.

Note: You know, this might make a good tutorial in the future. If this is something you would like to learn more about, click the Send Feedback link on this page and let Macromedia know so they can pass it on to me.

Now that you have the basis of your navigation system in place, you can use a background image to make the two columns look the same length at all times. This will help make the page look more structured.

Dreamweaver Article

Designing with CSS – Part 5: Defining Columns and Vertical List Navigation

Table of Contents

Adding a Background Image

Making two columns appear to be the same length is very easy to accomplish using background images. You will use Fireworks 8 to accomplish this. Complete the following steps:

1. Launch Fireworks and click the New Image icon.
2. Make the image **200** pixels wide and **20** pixels high.
3. Select the Rectangle tool and drag it out over the canvas.
4. Give it the color value **#D4DAD6**.
5. Select the Line tool and hold down the Shift key to ensure your line remains straight.
6. Draw a line down the right side. It should be one pixel wide.
7. Give the line the color value **#666666**.
8. Select Modify > Canvas > Canvas Size.
9. In the Canvas Size dialog box type **1** in the height box and click OK.
10. Select File > Image Preview.
11. In the Image Preview dialog box select 4 for the color maximum and click the Export button.
12. Save the file in your images folder.

Play the demo: Creating a Background Image Movie

You now need to reference the image from within the basiclayout.css file. Open this file in Dreamweaver and locate your `#wrapper` selector (see Listing 6).

```
#wrapper{
width: 770px;
background-color: #ffffff;
margin: 10px auto;
border: 1px solid #000000;
text-align: left;
}
```

Listing 6. `#wrapper` selector code

Now complete the following steps:

1. Place your cursor after the `text-align: left;` property and press Enter.
2. Type **background-image:**
3. When the Browse button appears, click it to open the Select File dialog box.
4. Navigate to your image, select it, and click OK to insert the code.
5. Type **;** after the code and press Enter.
6. Type **background-repeat:**
7. From the code hints, select **repeat-y** and then type **;**
8. Save your work.

The `#wrapper` selector should now resemble Listing 7.

```
#wrapper{
width: 770px;
background-color: #ffffff;
margin: 10px auto;
border: 1px solid #000000;
text-align: left;
background-image:url(../../tutorial_images/wrapper_bg.gif); /*the path may vary according to your local
site set up*/
background-repeat: repeat-y;
}
```

Listing 7. Revised `#wrapper` selector code

You can now preview the page in your browser (see Figure 6).

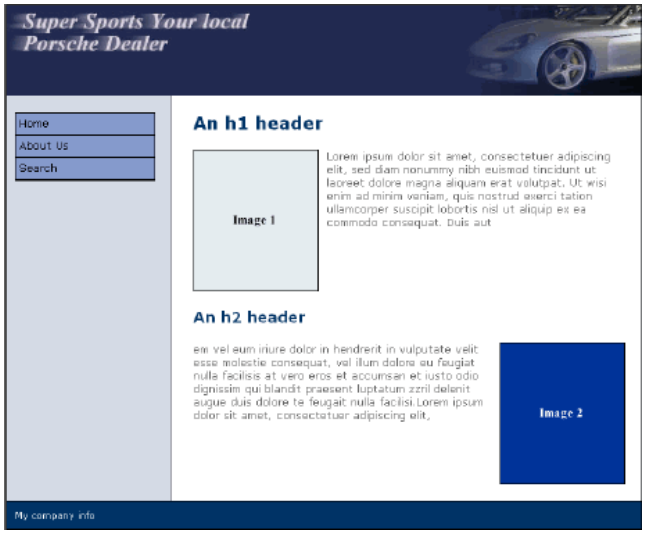

Figure 6. Two distinct columns in the layout

I hope you have enjoyed this tutorial so far. Next, got through the last article in this series.

Designing with CSS – Part 6: Deciding Whether to Float or Position Columns

Dreamweaver Article

Designing with CSS – Part 6: Deciding Whether to Float or Position Columns

Adrian Senior

www.webade.co.uk
www.communitymx.com
www.ukcsstraining.co.uk

Table of Contents

FEEDBACK

Created:
22 February 2005
Modified:
12 September 2005
User Level:
Beginner

Note: This article has been updated for Dreamweaver 8. If you are still using Dreamweaver MX 2004, please read the version of this article series for Dreamweaver MX 2004. The CSS features in Dreamweaver have been vastly improved in Dreamweaver 8. You can learn about those changes in Julie Hallstrom's article, An Overview of CSS in Dreamweaver 8.

Welcome to Part 6 of this article series on CSS design concepts. If you missed the earlier tutorials in this series you can find them below.

Designing with CSS – Part 1: Understanding CSS Design Concepts

Designing with CSS – Part 2: Defining Style Properties and Working with Floats

Designing with CSS – Part 3: Creating Your First Design Without Tables

Designing with CSS – Part 4: Creating a Two-Column Layout

Designing with CSS – Part 5: Defining Columns and Vertical List Navigation

In this series, I am reviewing how you can use Dreamweaver 8 to move towards using the CSS positioning technique of developing web pages.

In this, the final part in this series, I will review alternative methods to create a two-column layout. So far, I have used the float method (my preferred option), but I could also *position* the navigation div to achieve the same results. I will review the positioning method in this tutorial, and I'll explain why I prefer the float method over the positioning method. In this tutorial, you won't use the CSS and page layout you created in earlier parts of this tutorial series; you will be concentrating solely on the structure of the page. I'm sure you have a good grasp on styling your pages now, as I have covered that topic in depth earlier in the series.

You may find it advantageous to read an earlier tutorial I wrote for the Developer Center on relative, absolute, and static positioning, called Introduction to CSS Positioning in Dreamweaver MX 2004. It should give you a decent grounding in what you can achieve with each positioning value.

Requirements

To complete this tutorial you will need to install the following software and files:

Dreamweaver 8

- Try
- Buy

Sample files:

- css6_samples.zip (ZIP, 2K)

Prerequisite knowledge:

- Lorem Ipsum Generator extension by Captain Cursor Creations (optional)
- Internet Explorer browser
- Firefox browser
- Read Part 1 of this series
- Read Part 2 of this series
- Read Part 3 of this series
- Read Part 4 of this series
- Read Part 5 of this series

Further reading:

I have completed a two-column CSS layout—called North Pole JumpStart—that you can download from CommunityMX, complete with a set of tutorials on how the layout was achieved. I have also written a study of the structural side of the CSS of this JumpStart. I recommend that you read through it:

- Download the North Pole JumpStart
- Read the structural study article on North Pole

About the author

Adrian Senior owns the UK-based web design agency Webade, which has been in business since 1998. Adrian also provides courses for companies and individuals who need to build compliant, accessible websites using CSS and XHTML. He is also an Adobe Community Expert and a partner at Community MX. The year 2004 saw Adrian's first trip to America, where he visited Orlando and delivered two sessions at the TODCon conference.

Before becoming involved with website design and development, Adrian's working life centered around the shipyards of the River Mersey and oil production units in the North Sea. Adrian has been married to his wife, Janette, for 26 years. They have two children, Antony and Eleanor.

Dreamweaver Article

Designing with CSS – Part 6: Deciding Whether to Float or Position Columns

Table of Contents

Should I Float or Should I Position?

That is the question! You have already learned that to have two block-level elements, like divs, sitting side by side within your pages, you must remove one of them from the flow of the document. In this series, you have achieved that by floating the navigation div to the left. As I discussed in Part 2, floated content is removed from the document flow. You can refresh your memory if you need to by reviewing the float section of that tutorial.

So if you don't float the navigation div out of the document flow, how else can you achieve this? The answer is by positioning the div with an absolute value. Any element that is positioned absolutely is taken out of the document flow—and, as I've discussed, once you take your navigation div out of the document flow, you can have two divs sitting side by side in your document.

In the sample files (also linked to at the beginning of this article), you will find a page called positioned.html. The CSS for this page is embedded in the <head> section of the page's code. I have discussed the merits of using an external style sheet in this series and it still holds true. I embedded the CSS in the head of the document for the following reasons:

- The CSS you are using is minimal
- It's easier this way for the tutorial

When you use an absolutely positioned element, you need to be aware that the element containing it needs to be positioned. If the element that contains the absolutely positioned element is not positioned, you will get some rather unexpected results.

Let's begin by taking a look at the CSS in the positioned.html page:

```
<style type="text/css">
/* HEIGHT COMMENT: The height in the following selectors is only to hold the divs open, as you are only
looking at structure in this tutorial and will not be adding content. In a real page, you wouldn't use
the height property in this manner. Instead, you would allow the div to collapse around the content*/
body{
  margin: 0;
  padding: 0;
  background-color: #fff;
}
#wrapper{
  margin: 0 auto;
  width: 770px;
  height: 600px;/*See the HEIGHT COMMENT */
  background-color: #ccc;
  border-left: 1px solid #000;
  border-right: 1px solid #000;
}
#content{
  width: 520px;
  margin-left: 230px;
  background-color:#ffc;
  height: 600px;/*See the HEIGHT COMMENT */
}
#nav{
  position: absolute;
  width: 200px;
  left: 10px;
  height: 150px; /*See the HEIGHT COMMENT */
  border: 1px solid #000;
  background-color: #66f;
}
</style>
```

Note: Regarding the use of the `height` property, I have discussed this property (along with its pitfalls) in Part 2 of this series. You can also view the movie below to learn about the `height` property. In short, Internet Explorer implements the `height` property incorrectly and expands the div. Firefox implements the `height` property correctly, but allows the contents of the div to spill outside the div when the contents exceed the given height.

Play the demo: Investigating the Height Property

The only property and value pairs that I have not covered yet in this series can be found in the #nav selector. These include the `position: absolute;` property and the value pairs that are used to position the div—namely, `left: 10px` and `top: 0;`. Notice in the previous code listing that I did not declare zero for the top value. Not declaring a position value is not the same as zero. If you do not specifically declare zero, your div is not truly out of the document flow. Taking a div out of the document flow opens up possibilities for search engine optimization, as you will see on the

next page. For now, I have omitted the `top` property and value in order to show you that declaring zero is not the same as declaring nothing.

Open positioned.html in Dreamweaver and preview it in Firefox (see Figure 1).

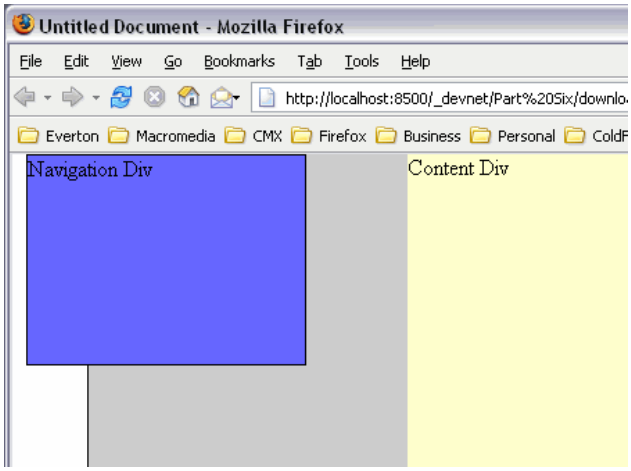

Figure 1. Absolutely positioned nav div outside the wrapper div

See how the navigation div has moved outside the wrapper div, shown in gray. This is not what you want. The idea is that the `left: 10px;` property and value pair should position the navigation div 10 pixels inward from the left of the wrapper div. The fact that the wrapper div is not a positioned element allows the navigation div to move outside its defined boundaries. It will do this until it finds a positioned element or it meets the view port—meaning it flushes up against the left edge of the browser window.

I haven't declared a top position for the #nav selector because I want it at the top of the wrapper. Because zero is the default, I have no need to declare a value. You can see from Figure 1 that the navigation div has taken its position from the view port. It has the default top value of zero, which sets it tightly against the top of the view port, but the `left: 10px;` assignment is being taken from the left of the view port rather than the wrapper div. Don't worry; this is easily corrected.

Play the demo: Positioning the Wrapper Div

Follows these steps to complete the next process:

1. In Dreamweaver, locate the #wrapper selector in positioned.html.
2. Place your cursor after `#wrapper{` and press Enter.
3. Type **Position: relative;**
4. Save your work and refresh the browser or preview the positioned.html page again.

You should now see the nav div respect the position setting on the wrapper and move correctly into place (see Figure 2).

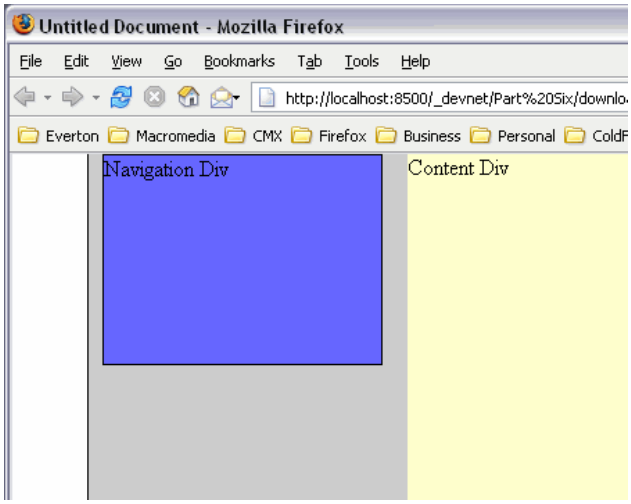

Figure 2. Making the nav div respect the position property on the wrapper

Dreamweaver Article

Designing with CSS – Part 6: Deciding Whether to Float or Position Columns

Table of Contents

Major Problem with Absolute Positioning

As I see it, the major problem with absolute positioning is this: You cannot clear an absolutely positioned element. This could cause problems and it's something you need to take into account when you consider the structure and layout of your designs. If you want a footer that expands the width of your wrapper, and you are unsure whether the navigation div might end up taller than the content div, then you have a problem.

Recall how you were able to ensure that your footer always appeared below your content when you floated your navigation column by using the `clear` property. Absolutely positioned elements do not respect this. They just push straight through the footer regardless of any attempts you might make to force the footer below the absolutely positioned element. This can make your page very ugly, as your navigation div pushes right through your footer and out through the bottom of your design.

Open withfooter.html in the Code view and preview the page in Firefox (see Figure 3).

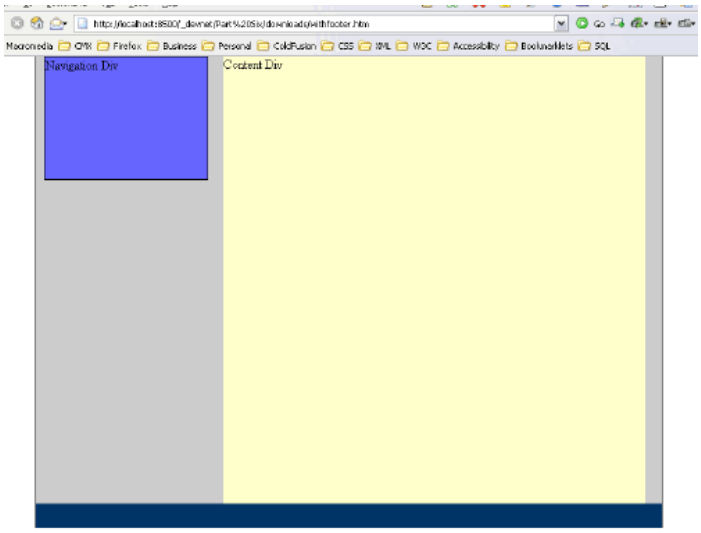

Figure 3. Footer in place with the design still intact

I demonstrate the problem in the Breaking Through the Footer Captivate demo.

Play the movie: Breaking Through the Footer

Follow these steps to complete the next process:

1. In the Code view, locate the #nav selector.
2. Change the height value to **700px**.
3. Preview your page. Note how the nav div pushes through the footer.
4. Locate the #footer selector.
5. Place your cursor after `background-color: #036;` and press Enter.
6. Type: **clear: left;**
7. Save your work and preview the page in Firefox.
8. Notice that the nav div still pushes through the footer.

While there is nothing wrong in principle with using positioned divs to create columns, you do need to be 100% sure that any absolutely positioned element will not cause this type of problem. As you have seen in earlier tutorials, a floated layout does not suffer from this type of scenario because you can clear floats.

Surely Absolutely Positioned Elements Have Their Good Points Too?

The answer to that question is yes. When you define an element as being absolutely positioned, it is taken out of the document flow. Unlike a floated element, an absolutely positioned div can be placed just about anywhere in your code. This can present some good search engine optimization opportunities. For example, you could move the nav div out of its current position in the source code and place it elsewhere within the wrapper div.

Play the demo: Source Order and Absolutely Positioned Divs

Follow these steps to complete the next process:

1. Open withfooter.html in the Code view.
2. Locate the #wrapper and #nav divs in the body of your page.
3. Switch the positions around in the code. Your content div should now be before your nav div in the source order.
4. Preview the page in Firefox.
5. You will see the nav div is unaffected by its change of location in the source code.

You now have a position where your content is above your navigation in the source order. This can be a good method for pushing your content at a search engine's bots. The content that matters now appears higher in the code than it would be with a floated column scenario.

You have now seen two methods of creating a two-column layout. A bit of planning before you dive in will probably help you decide which method to use. Both have their strengths and weaknesses. The choice is yours.

I hope you enjoyed working through this series. I know I enjoyed writing it. I plan to write another series that looks at other aspects of CSS. Stay tuned.