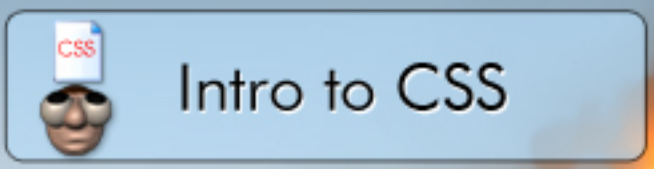
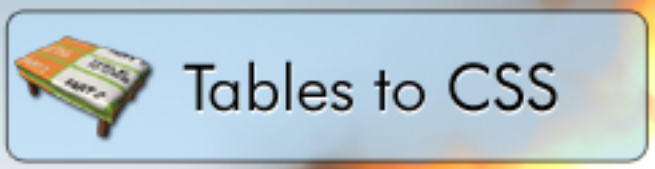
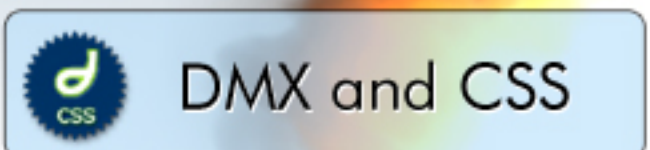
DMX zone

CSS Web Sites with Dreamweaver MX

Everything you ever wanted to know about Web Standards, CSS and Dreamweaver but were afraid to ask







Rachel Andrew  Molly E. Holzschlag 

CSS Web Sites with Dreamweaver MX

(Everything you ever wanted to know about web standards, CSS and Dreamweaver .. but were afraid to ask)

Rachel Andrew

Molly E. Holzschlag

Published by DMXzone.com
Dynamic Zones International
Hengelosestraat 705
7521 PA Enschede
The Netherlands

No font tags were harmed during the making of this book.

Trademark Acknowledgements

DMXzone has endeavoured to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, DMXzone cannot guarantee the accuracy of this information.

Dreamweaver, Dreamweaver MX and Dreamweaver MX 2004 are trademarks of Macromedia.

Table of Contents

Introduction

CSS is news and becoming mainstream.

Cascading Style Sheets have been around for a long-time, and are finally ready for prime time. For many web professionals, that's a relief – finally, browsers have caught up with the CSS standard so we can actually use CSS!

But to many other web professionals it's beginning to seem like a curse: so much to learn, and – equally importantly – so much to **unlearn.** Web pages made with font tags and nested tables still look fine to the majority of people out there, often rendered more reliably than cutting-edge CSS designs.

But like it or not, CSS is here to stay. The CSS wave started with a few personal sites - www.zeldman.com, Eric Meyer's CSS site www.meyerweb.com and a number of others. Then, big name sites started to perceive CSS's advantages: Wired news converted to CSS, as did www.quark.com and www.espn.com with 10 million visitors per day.

Many sites built with CSS still suffered from the perception that it restricts creativity and is always "boxy". That was true, but was more a limitation of the design sensibility of those early adopters, who were mostly developers rather than artists. Dave Shea's CSS design showcase site, the CSS Zen Garden, put the stake through the heart of that myth.

What this book does, and who it's for.

The purpose of this book is **not** to persuade you to use CSS. The fact that you've bought it shows that you want to get to grips with Style Sheets. It might be that you want to see what all the fuss is about, or that you're a convert to the idea and can't wait to get designing. Maybe you don't want to be bothered, but a client or boss demands a CSS site. We're not here to preach, but we do believe that, once you've started designing sites this way, you'll appreciate the future simplicity of CSS design – once you've scaled the learning curve.

Because CSS was largely adopted by the developer community, who are generally hand-coders, most of the information out there on CSS web development is focused on those who are entirely comfortable with code, and mostly aimed at those who hand-roll sites using a text editor.

This book is for Dreamweaver developers. It assumes you have a working knowledge of Dreamweaver MX or MX 2004 (it doesn't matter which, because although MX 2004's CSS support has a better user interface than MX has, the techniques and tutorials work the same. We point out the important differences in the text as we go through). This book **doesn't teach you Dreamweaver**. It shows you how to make CSS sites using Dreamweaver as your development tool.

Each chapter is a project, so you'll learn by doing. We look at the basics of cutting down presentational HTML, then removing tables, move on to CSS for positioning, and then the techniques used to make some common layouts (two columns, three columns, and so on), and then how to use CSS to style boxes, borders, margins, lists and so on. Both authors are established Dreamweaver experts, and both are members of the Web Standards project. Let's meet them.

Authors

Rachel Andrew

Rachel Andrew is a trained dancer and singer, whose CV lists jobs as diverse as company choreographer for a physical theatre company to chargehand carpenter for "The Mousetrap" at St. Martin's Theatre in London's West End. After leaving the theatre when pregnant with her daughter, Rachel started to design sites mainly out of curiosity into how it worked. It didn't take too long for her to figure out that her skills lay in development as opposed to design and these days she tends to leave the design to designers so she can concentrate on writing code, dismantling computers and installing Linux on anything that stays still long enough.

Rachel has worked in the industry as a webmaster, technical project manager and senior web developer but in September 2001 set up her own company 'edgeofmyseat.com', which provides complete web solutions and outsourced development services for design agencies and Internet start-ups who do not have in-house web developers.

Rachel is also a member of the Web Standards Project serving on The Dreamweaver Task Force.

Molly E. Holzschlag

Coined "one of the greatest digerati" and deemed one of the Top 25 Most Influential Women on the Web, there is little doubt that in the world of Web design and development, Molly E. Holzschlag is one of the most vibrant and influential people around. With over 25 Web development book titles to her credit, Molly currently serves as Communications Director for the World Organization of Webmasters.

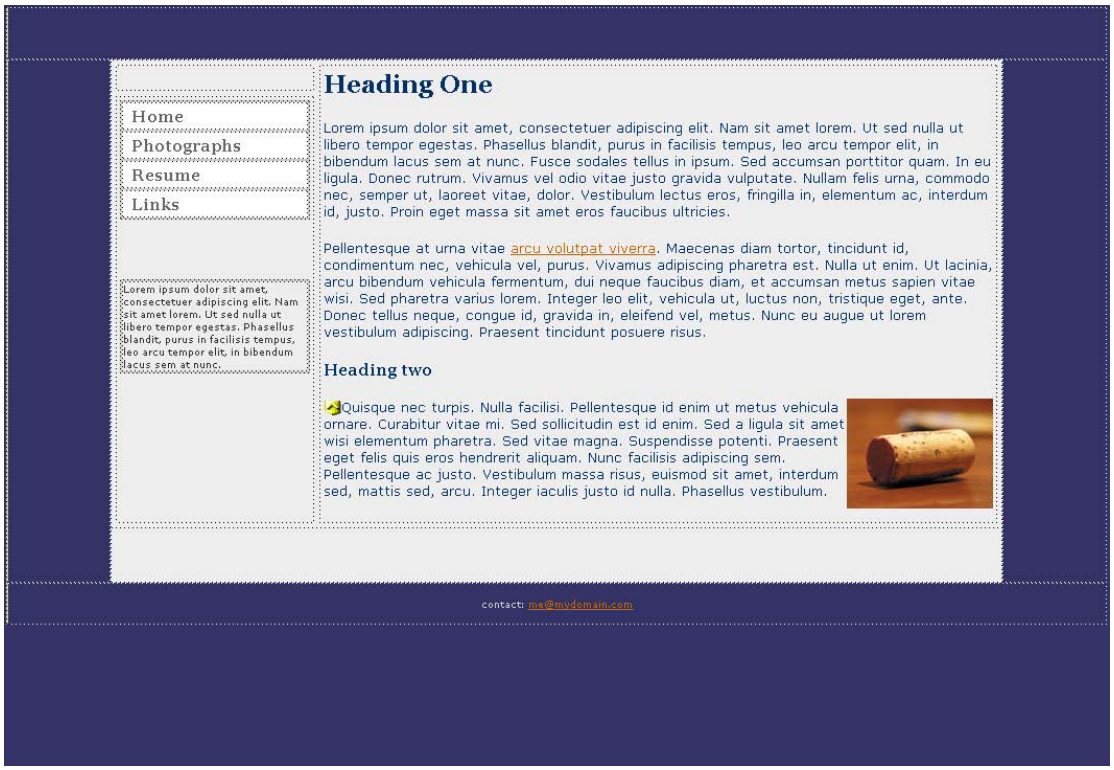
As a steering committee member for the Web Standards Project (WaSP), Molly works along with a group of other dedicated Web developers and designers to promote W3C recommendations. She also teaches Webmaster courses for the University of Arizona, University of Phoenix, and Pima Community College. She wrote the very popular column, Integrated Design, for Web Techniques Magazine for the last three years of its life, and spent a year as Executive Editor of WebReview.com.

1.Tables to CSS: A hybrid layout.

This chapter is for those people who are comfortable using Dreamweaver to lay out sites using tables, but would like to discover more about how to use CSS for more than just basic text styling. We will take a tables-based layout that includes several of the design tricks common when creating sites using tables for layout, and look at how we can use CSS to make a tables-based layout cleaner, leaner and more accessible but without making that huge leap into total CSS layout. This is often known as a hybrid layout as it uses tables for layout, but other styling is achieved through CSS. **It is a perfect solution for those legacy sites that you don't have time to fully redesign but that need to be improved in terms of their accessibility, or as an easy introduction to CSS in Dreamweaver.**

The layout

Here is our layout, fairly uninspiring, but it has been built in Dreamweaver, in the old fashioned way with font tags, cell background colors and nested tables.



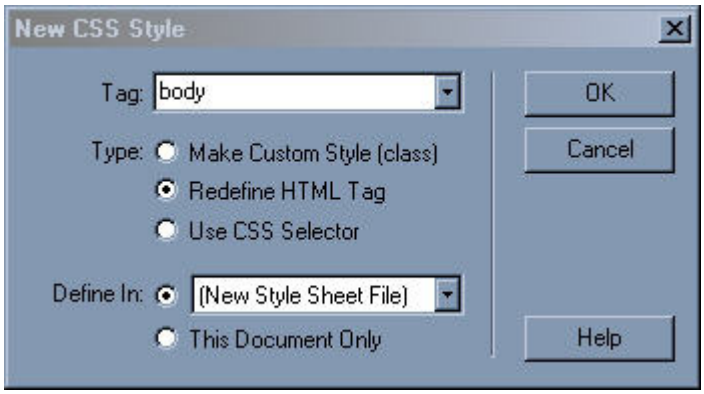
Cleaner table-based layouts with CSS

We are going to keep our existing tables-based layout but use CSS for as much of the styling of those tables as possible. This will result in a site that is easier to maintain and loads more quickly.

Font tags

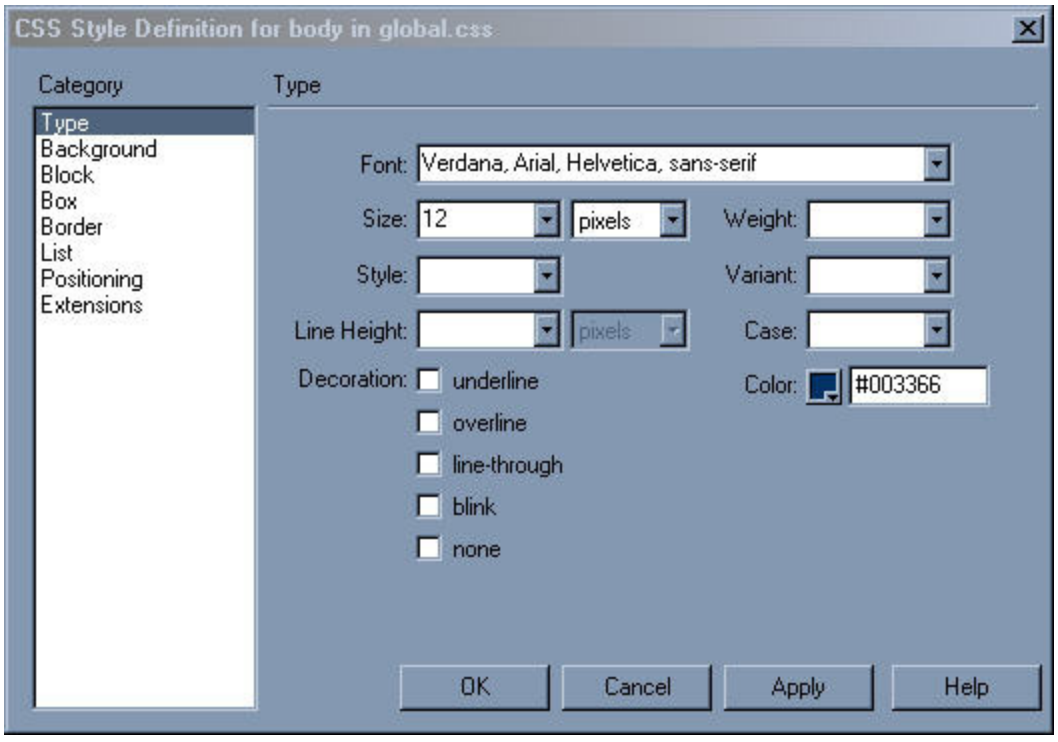
Firstly, let's create a new style sheet in Dreamweaver and put all the font information into it.

In the CSS Styles Panel of the Design Panel Group click '**new CSS style**'.
In the dialogue that opens select **Tag** – '**body'**; select - '**Redefine HTML Tag'**; Define In 'New Style Sheet File'.



Click OK and you will be prompted to save the style sheet. Save it into your site directory.

Once you have saved your style sheet, you will be presented with the CSS Style Definition Dialogue. Here you can reproduce your font tags with CSS declarations as in the image below.

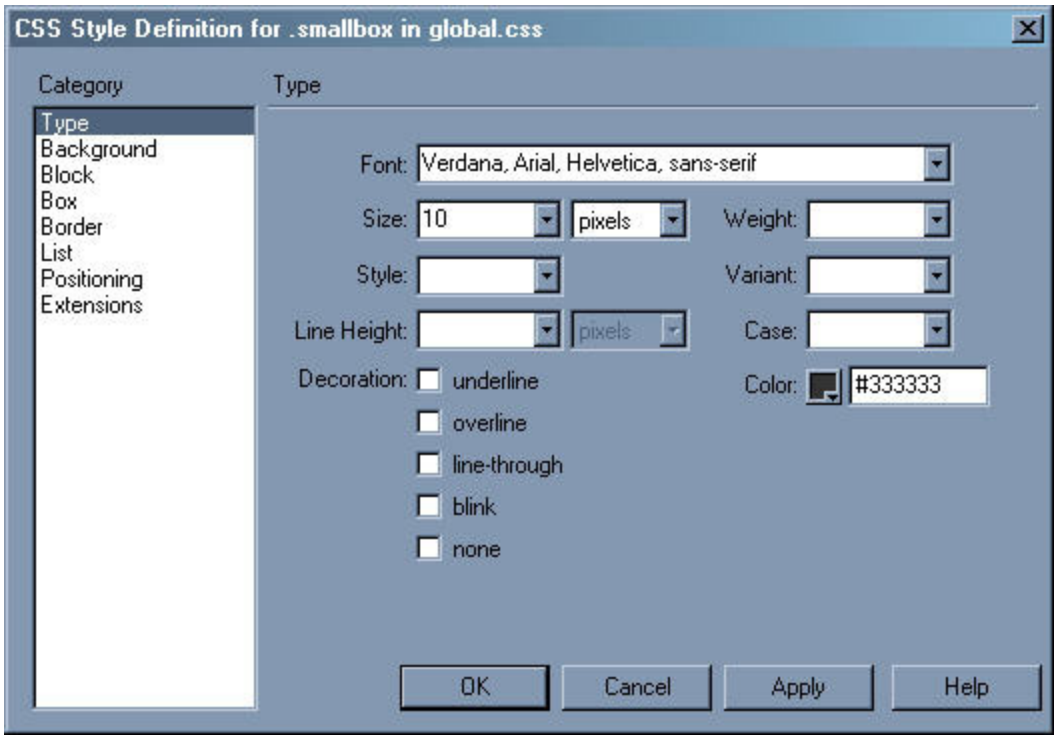


After creating the rules for body you will need to return to the CSS Panel and create rules (in exactly the same way) for p, td, li – this time however, in the initial dialog choose to define your styles in the style sheet that you created when defining the body tag and **not** in a New Style Sheet.
Once you have defined CSS replacements for your main text, you can remove the font tags from within the mark-up.

Custom Classes for Text

In our layout we have a boxed out area below the nav box, on the lower left of the screen, in which the text is smaller than the text on the rest of the page. Using font tags we would just set that text to a smaller size than we did for the main text. However, the CSS styles are applied across all text so if you have removed the font tags from that area it will have changed to the size of the rest of the body text. To set this area to use smaller text we need to use a custom class.

Launch the New Style Dialogue again and this time select 'Make Custom Class', again defining in our style sheet. In the first box type the name of your class, '.smallbox'. Click OK and set the rules for the fonts in this class.

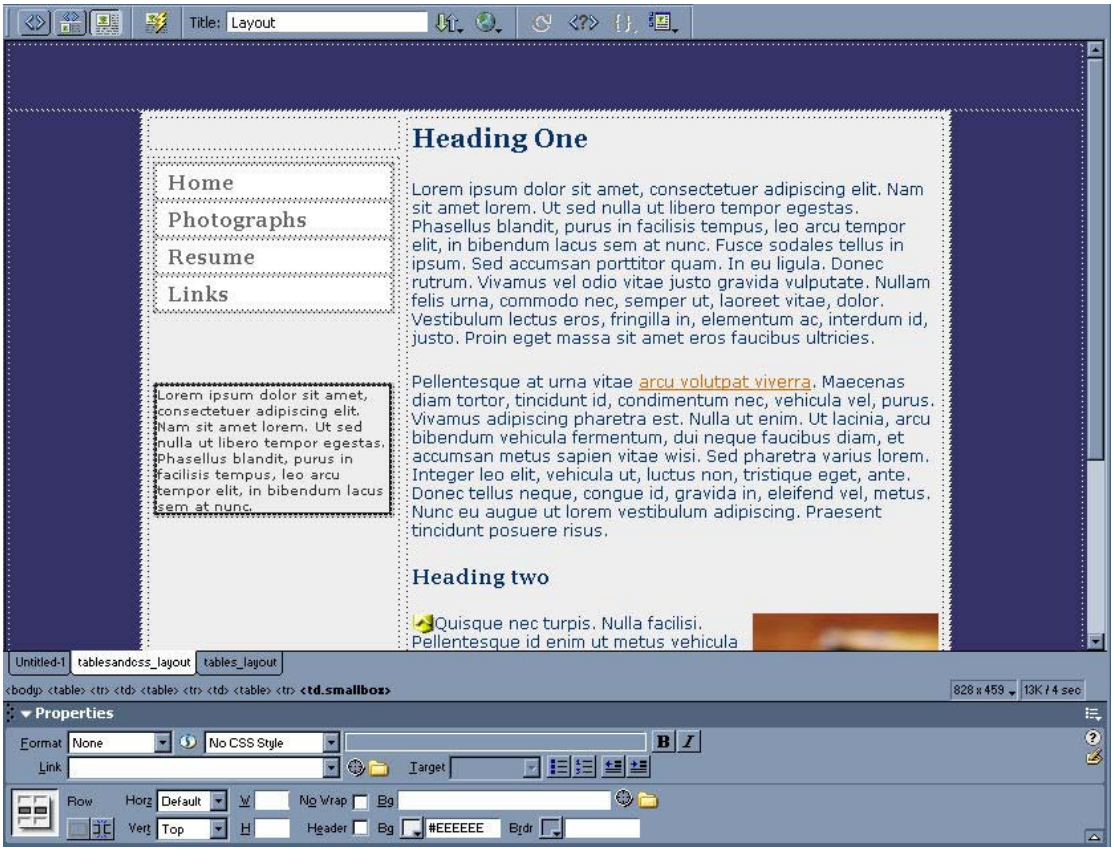


Once you have set your rules and clicked OK you will see your class appear in the CSS Styles Panel. It won't format your box just yet because we need to apply that class to the box.

Before we do that, have a quick look at your Property Inspector. If it looks like this:

Click on the little yellow 'A' and your Property Inspector will transform into the CSS Property Inspector - you can always change it back if you need to by clicking the funny 's' thing – but when working in CSS you will find this mode very useful, and we will use it now to apply our class.

Select the td that contains the boxout text. With this selected, go to the Property Inspector and you will see a drop-down list that currently says 'No CSS Style'. Click on this and you should see the custom class that you created listed here. Select it and the text in the box will automagically transform to take on the styles you set for it.



If the end result doesn't look quite right simply double-click on the name of the class in the CSS Styles Panel to edit the rules.

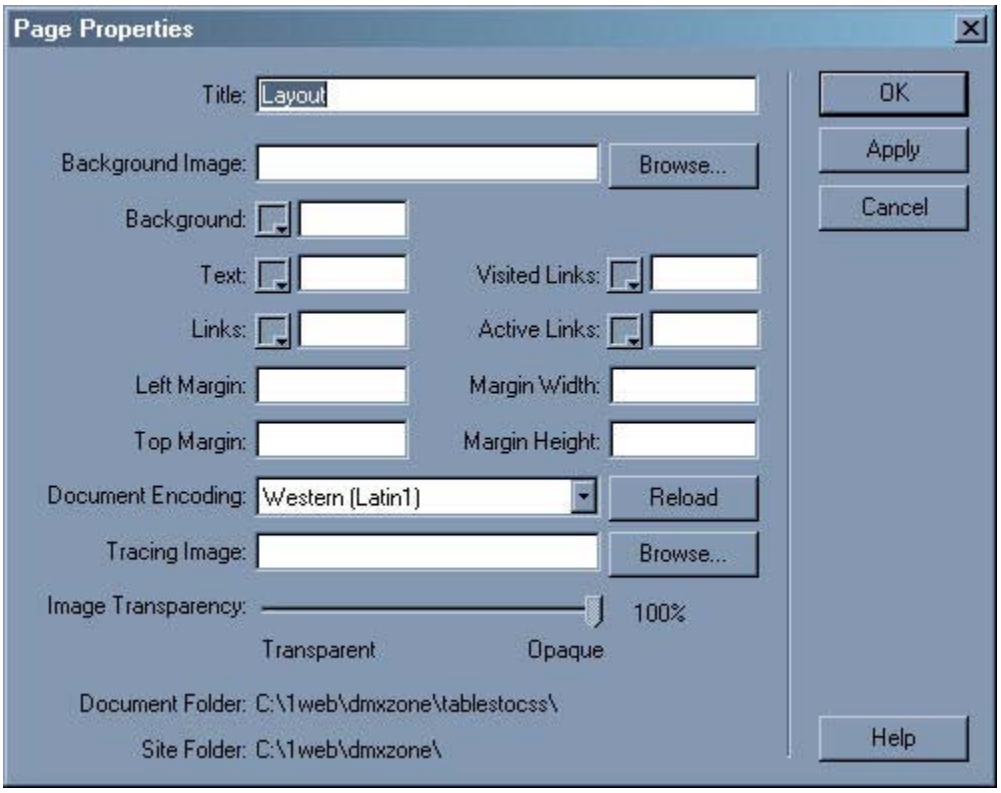
Using these two techniques you should be able to easily replace all the font tags in your site. Redefine your headings – or create custom classes if you need different styles of heading – create a class for your footer text, and so on.

Cleaning up the body

Having removed all those font tags from the page, let's turn our attention to the body tag because there are some deprecated elements in there that can, and should, be replaced with CSS.

When an HTML tag is described as deprecated it has been flagged up to be removed from future versions of (X)HTML. You can still use deprecated tags in valid pages with transitional HTML and XHTML DOCTYPEs (the defaults used by Dreamweaver), but their days are numbered. For maximum 'future-proofing' your pages, it's a good idea to stop using deprecated tags now - especially because the deprecated elements at this point tend to be those that can easily be replaced with CSS.

Go to Modify > Page Properties and delete everything from the boxes there (apart from Title and encoding):

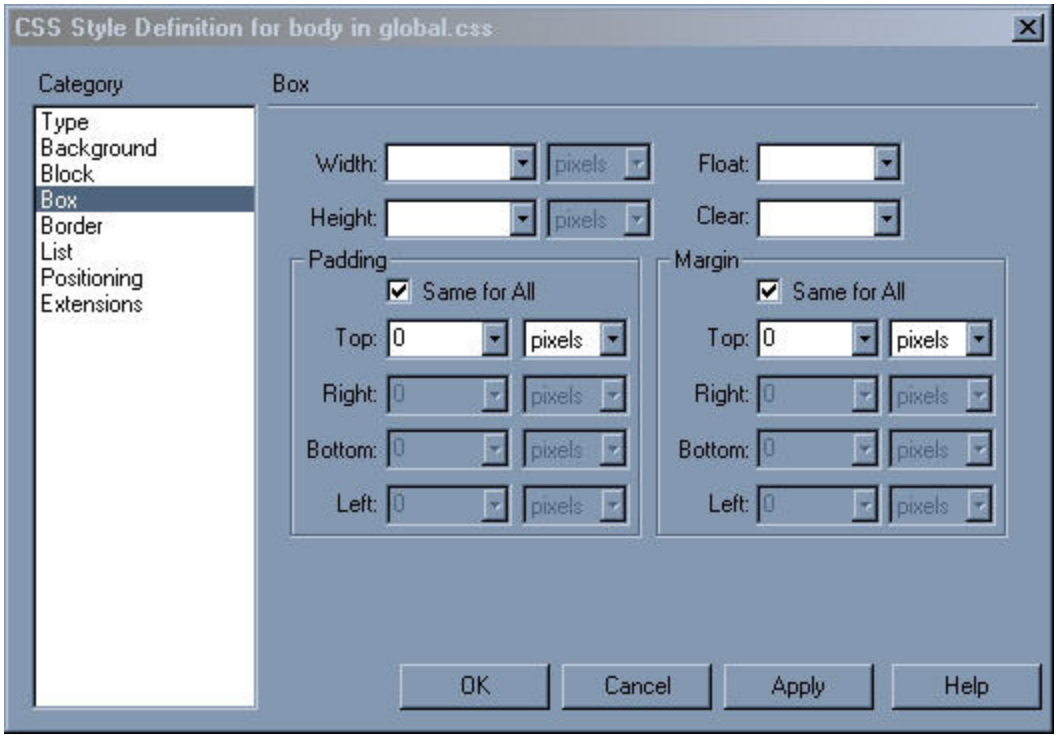


If you are working through with my example layout you will notice that, once you click OK or Apply, page margins appear, the background goes white (which is Dreamweaver's default background color; in reality the background would be the color the user had set as their default page background color) and links turn to default link colors. We can now set these properties using CSS.

In the CSS Styles Panel double-click on 'body' which opens up the dialog containing the rules that we set earlier for the body tag. We can now add to this to set the background color and page margins – just as we used to do with attributes of the body tag itself.

To set the background color, select 'Background' in the Category list on the left, then browse for or type in your background color in the first field on that dialog.

Now select the category 'Box', here you can set the margins and padding for the page, set Padding – Top to 0 pixels and check 'Same for All' and do the same for Margin – Top.

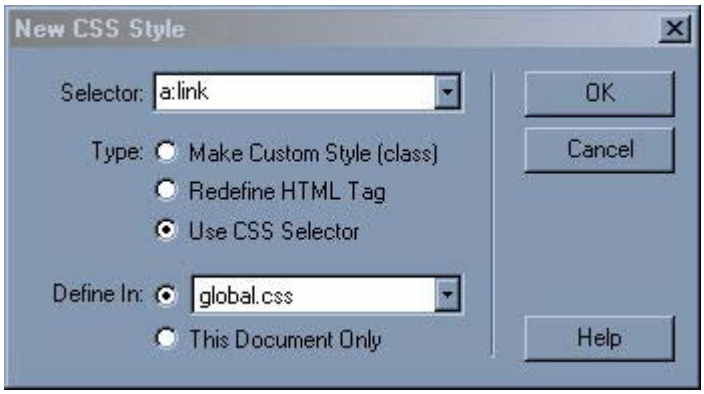


Click OK to get rid of your page margins.

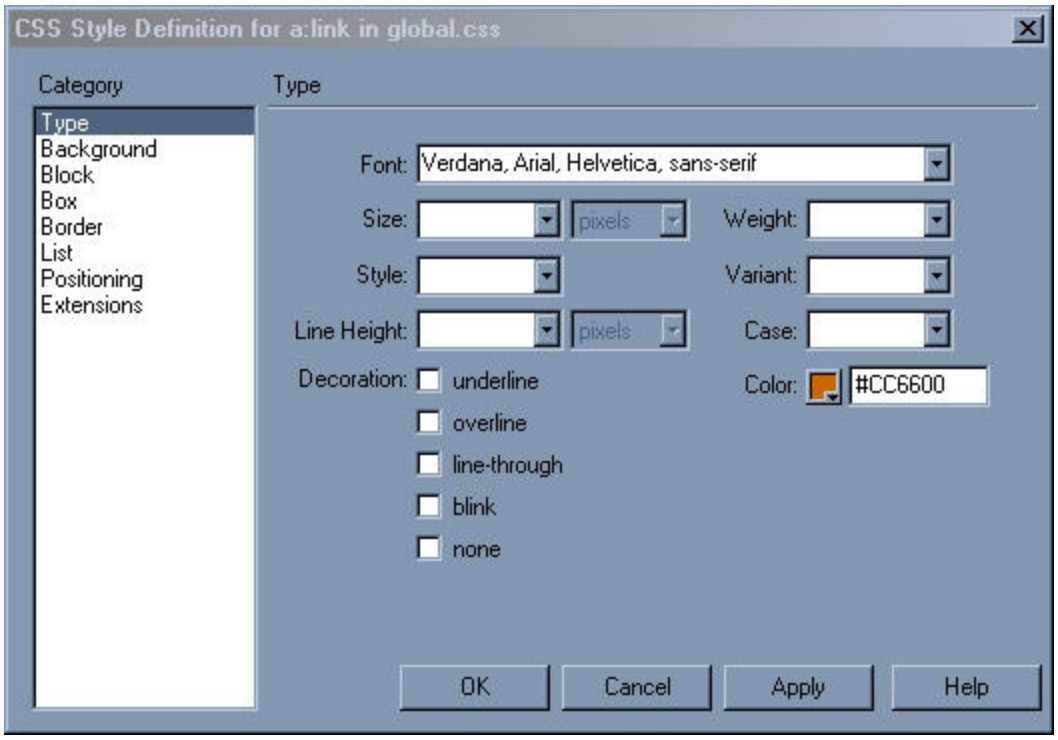
Links

We now can change our links from something other than the default colors set by the browser, and there is a way to do this using CSS in Dreamweaver too. Links, or more correctly anchor tags, can be styled by styling the various states of the tag – link, visited, hover, active.

Create a new CSS Style, this time in the dialog choose the radio button 'Use CSS Selector' and then in the drop down at the top choose 'a:link'.



Click OK and in the dialog set the text properties to how you wish your default link colors to appear.



Click OK and then repeat the process for a:visited, a:hover, a:active.

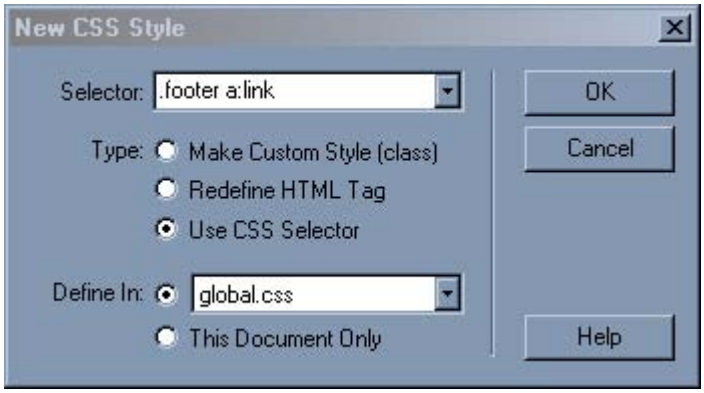
It is important that you create the states in this order LINK, VISITED, HOVER, ACTIVE.

One page – many link styles

With CSS it is possible to set up different sets of link styles for different places on a page – for instance in my demo page I have an email link in the footer which is a little bit dark in the default orange. To create a different scheme for this:

Create a new style and select 'Use CSS Selector' as before and choose 'a:link' in the menu. This time, before clicking OK, type .footer in front of a:link in the first box. '.footer' is the

name we used when creating a class to set the font size and color for this part of the page.



What we have done is create something called a pseudo-class selector – we will meet these again later in this chapter.
Click OK and use the dialog you dictate how you would like this link to display, once again you will need to create this for each of the states – LINK, VISITED, HOVER, ACTIVE. Because we have already applied the class '.footer' to the table cell surrounding this text, it will pick up the styling. With the syntax **.footer a:link** we are simply saying any a:link within the class .footer should look like this.

Navigation

Our original page has a navigation menu, which is a table containing rollover images, using JavaScript. This is a fairly standard way to achieve this effect and is used on many sites, however for many navigation bars such as this, the only reason that the designer has used an image is to get the rollover effect when someone holds their mouse pointer over the image. With CSS we can get rid of the images and JavaScript, making the page faster to load, easier to edit (you won't need to create a new image to add a new menu item or to change the colors) and also more accessible as to a device that doesn't support images or even CSS they will just get nice text links and can navigate your site with ease.

The Table

The navigation sits within a table, lets clean up this table first. Select the table and delete the background color using the Property Inspector.

Create a new CSS Style, choose to create a custom class and call it '.nav'.

In the CSS Style Definition dialog set the background color to the same color we just removed from that table (#cccccc). Click OK.

Now apply the class to the navigation table.

You will find that you can't apply this class using the Property Inspector in Dreamweaver MX). However, there are two other ways to apply classes. You can right click on the tag in the tag selector, choose '**Set Class**' and then select your class name, or you can select the element in the document window, then select the Radio button '**Apply Styles**' in the CSS Styles Panel, and then click on the class name in the panel.

The Buttons

Be ruthless and delete those images out of the cells of our navigation table. Then select each cell of the table in turn and delete the background color. What you will end up with is a grey box with 4 rows. Dreamweaver may or may not tidy up after itself. Finally if you switch into Code View and notice any JavaScript in the head of my example document, feel free to delete it – there's no need for it any more!

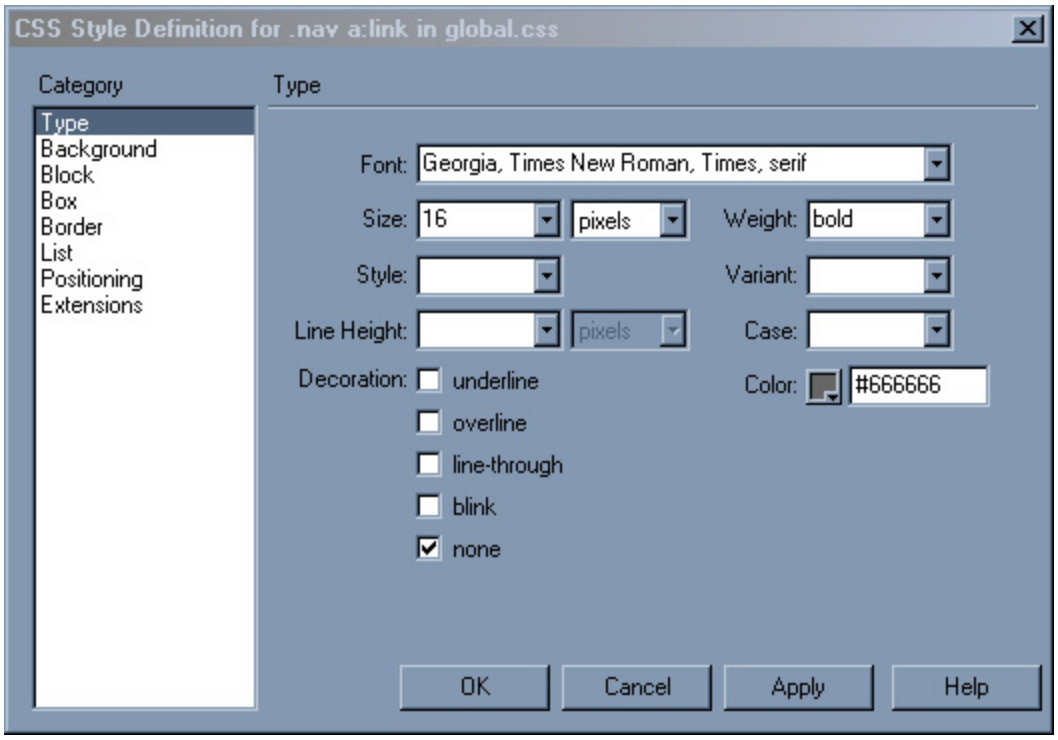
Create a new style, this time select 'Use CSS Selector' as we did for the links, but type '.nav td' into the box. Click OK and define a background color for this class as #ffffff. In the 'Box' category, give the cell a height of 24 pixels.

Click OK and watch all the cells of your table turn white again.

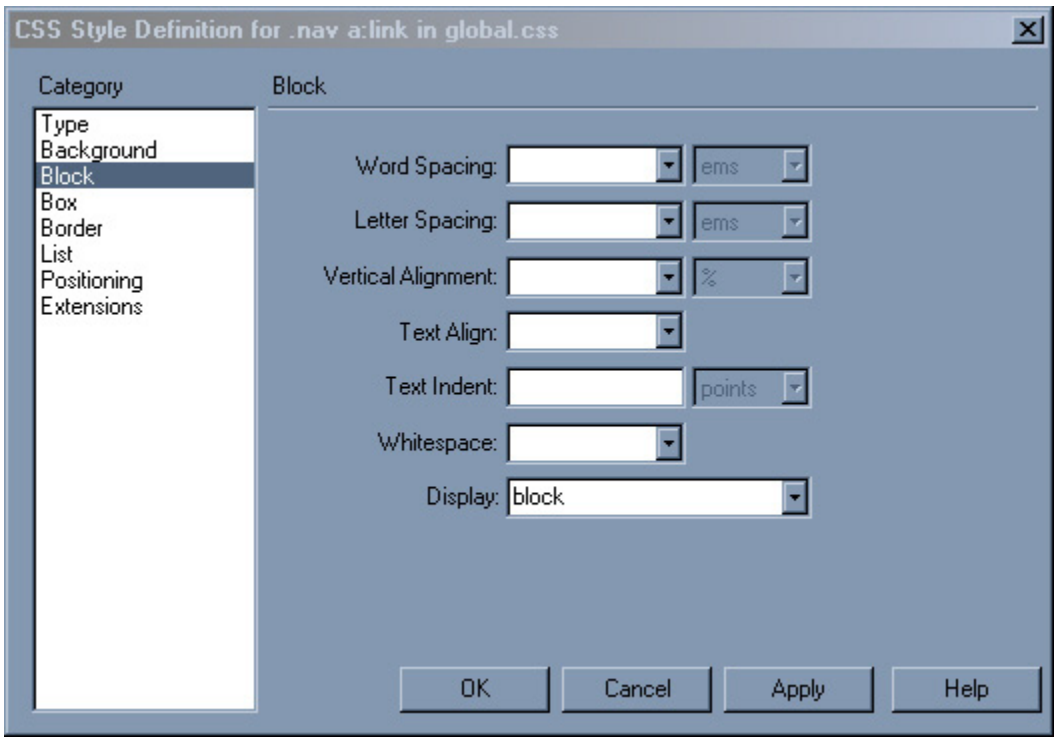
Type into each cell the navigation menu items – Home, Photographs, Resume, Links. Select each one and type '#' into the link field of the Property Inspector to create a dummy link, the links should take on your default link color.

Remember back to when we created the footer link? We can apply the same principles to making navigation links – with just a few extra touches.

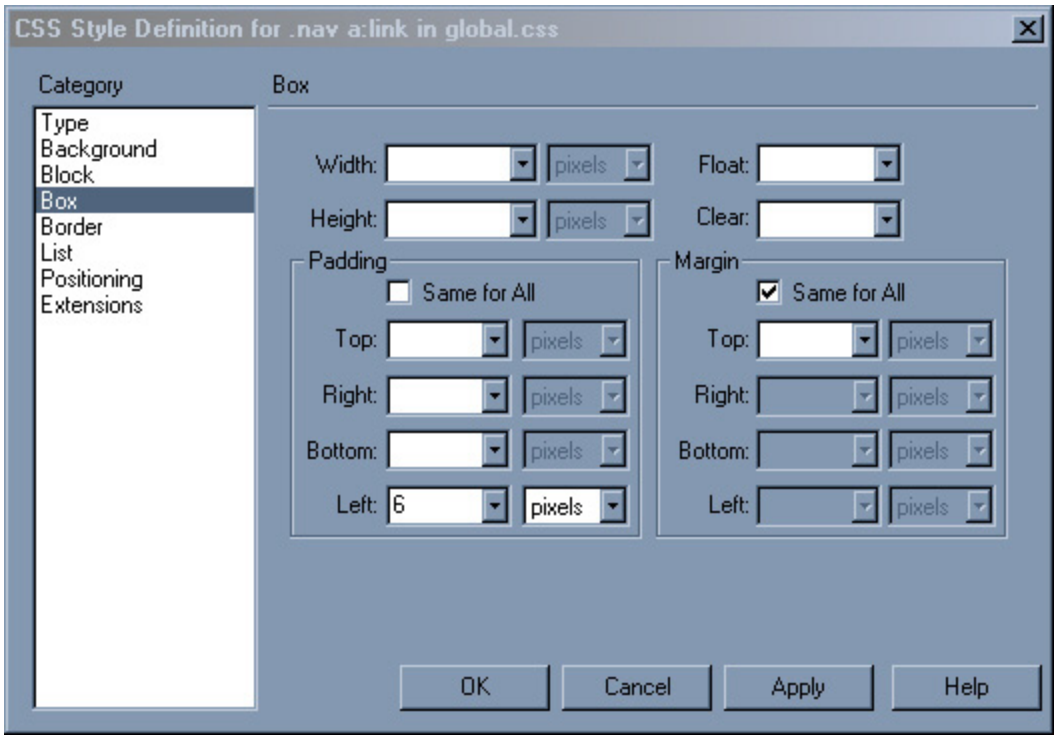
Create a new CSS Style, and as with the footer link select 'Use CSS Selector' and in the box at the top type '.nav a:link'. Click OK and then in the dialog set how you wish these navigation links to appear. Below are the settings I have used in the type category to reproduce the image buttons.



Before clicking Ok, select the 'block' category and in the last item on the list of options, select Display 'block'.



Finally, in the box category, you can set the padding and margins. Setting padding-left to 6 pixels will mean that the text does not bump up right against the side of the cell. You can change the other padding settings until the buttons look as you want them to.



Now you need to create rules for each of the other states as before – to create the hover effect, when you do .nav a:hover set the color to a different shade (in my case #999999) and the text will change to this color when someone holds their mouse over the button.

Taking it further

Even if your clean-up operation stops at the above, you have made your site far easier to update as you can create new pages from this style sheet and any changes made to the style sheet will affect every page linked to it. If you need to change the background color or navigation links color, it is simply a case of editing those once in the CSS. Your page will also be faster-loading and the switching of image links to text may well assist those using text-only devices in navigating your site, and we've done all this in the Dreamweaver Design View without needing to handcode.

You could continue to apply these techniques to further streamline this site, for instance the table containing the small text under the navigation could be treated in the same way as the navigation table and you could replace cellpadding and cellspacing on tables with CSS padding and margins. By experimenting with CSS, even without leaving the Dreamweaver Design View you can start to discover the power of CSS and the difference that it can make to your designs.

In the next chapter we will be taking this CSS redesign further – as here we have concentrated mainly on how to replace html styling tags with CSS and we have remained within the constraints of what we can do in the Design View of Dreamweaver.

2.Tables to CSS: Cleaning nested tables and using styled lists for Navigation.

In the last chapter I looked at how we could take a standard tables based layout and trim it down – using CSS for text styling and to create navigation button effects. In this chapter we will look more deeply into CSS. We will discover how to take our tables based layout down to a very basic one-table structure – removing all the nesting that makes the pages harder to maintain and in complicated pages can cause accessibility problems, and we will look at some of things that CSS can do over and above simply replacing HTML attributes. We will also look at a way of creating a navigation menu without using a table – by styling an html list with CSS.

The layout

This layout is a neat tables-based layout. It doesn't have a huge amount of nesting or html styling and the attached stylesheet already controls the appearance in the browser of most of the elements on the page.

Cleaning up the tables

If you have decided that you want to stay with using tables for layout for the time being but are still concerned about accessibility then you should try to make your tables-based designs as clean as possible, avoiding the temptation to nest tables one inside the other to get the effect that you want. Much of the time, the effects that you get from this nesting can be reproduced with CSS, giving you the added advantage that it is far easier to make changes as all of this information is held in one stylesheet.

Our layout contains tables nested 3 deep. There is a table used for the navigation, the main page content (the light grey area) is a table and then the whole page is a table. There is also a table used for the small boxout underneath the navigation, the nested tables are highlighted in the image below – we'll start by replacing that boxout.

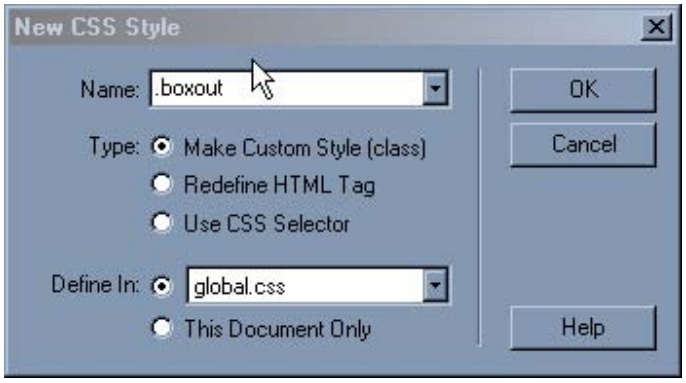
Copy and paste the text for the boxout so it sits just underneath the current boxout table in the left-hand column. You will see that it now takes on the look of regular body text, as the 'smallbox' style was applied to the table cell that it was in.

We could simply apply a style to that paragraph, but it is highly possible that your boxout might contain more than one paragraph so we will enclose this area in tags which mark it up as a separate block on the page, and then we can style this block and anything inside it.

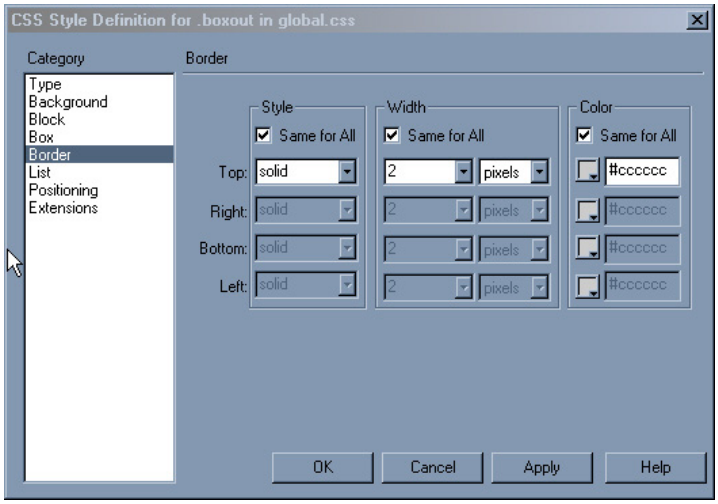
To wrap the text in a div, click your mouse cursor at the start of the text and switch into Code View. Before the first of that boxout, type and after the closing, final type . You should end up with a section that looks like this:

```
<div class="boxout">
            <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Nam sit
              amet lorem. Ut sed nulla ut libero tempor egestas.
Phasellus blandit,
              purus in facilisis tempus, leo arcu tempor elit, in
bibendum lacus
              sem at nunc.</p>
</div>
```

Now switch back into Design View, you won't see any change to the display as yet – we now need to create a class to apply to this box. Create a New CSS Class. In the New CSS Style dialog, name the class '.boxout' and Define in 'global.css'.

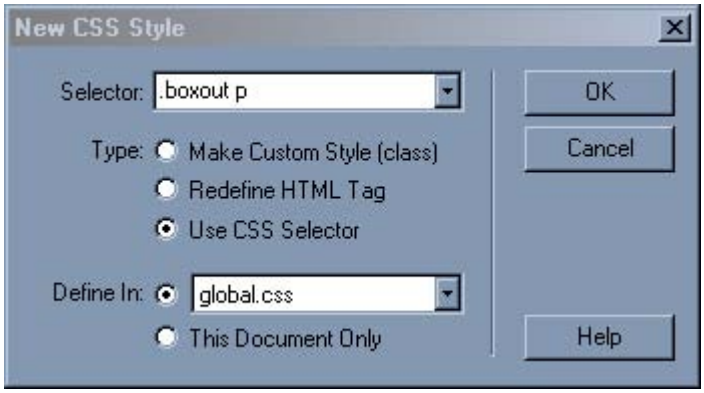


Click **OK**, in the next dialog box select the Category 'Border' and with **Same for All** checked set the values to Solid, 2 pixels and color #cccccc as in the image below.

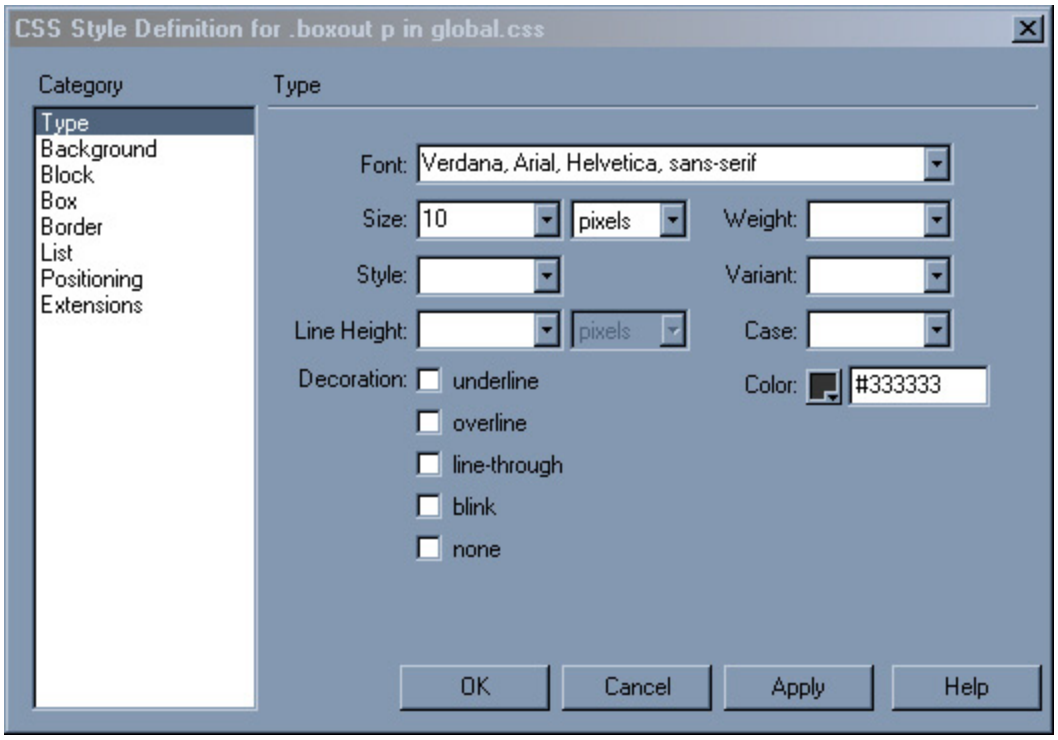


Click **OK** and because we set the name of the div when we created it you should see a border appear around the text.

To style the text within the box we can use a selector. We want to say “style all the <p> tags within the div 'boxout' with these rules. Create a new CSS style within Dreamweaver, choose to 'Use CSS Selector' and type **.boxout p** into the box.

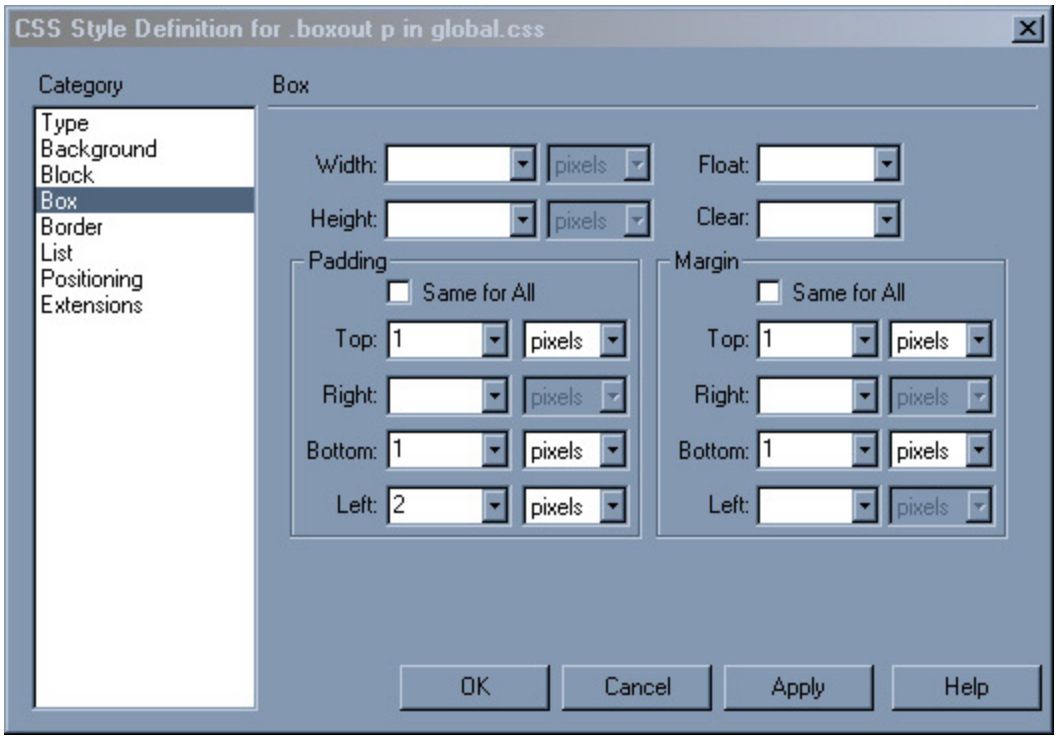


Click OK and then set the rules for this text in the next dialog. Clicking OK or Apply should show you the results.



When you preview in a web browser you will find some unwanted space at the top and bottom of the text, before the border, caused by the default margin and padding around the element.

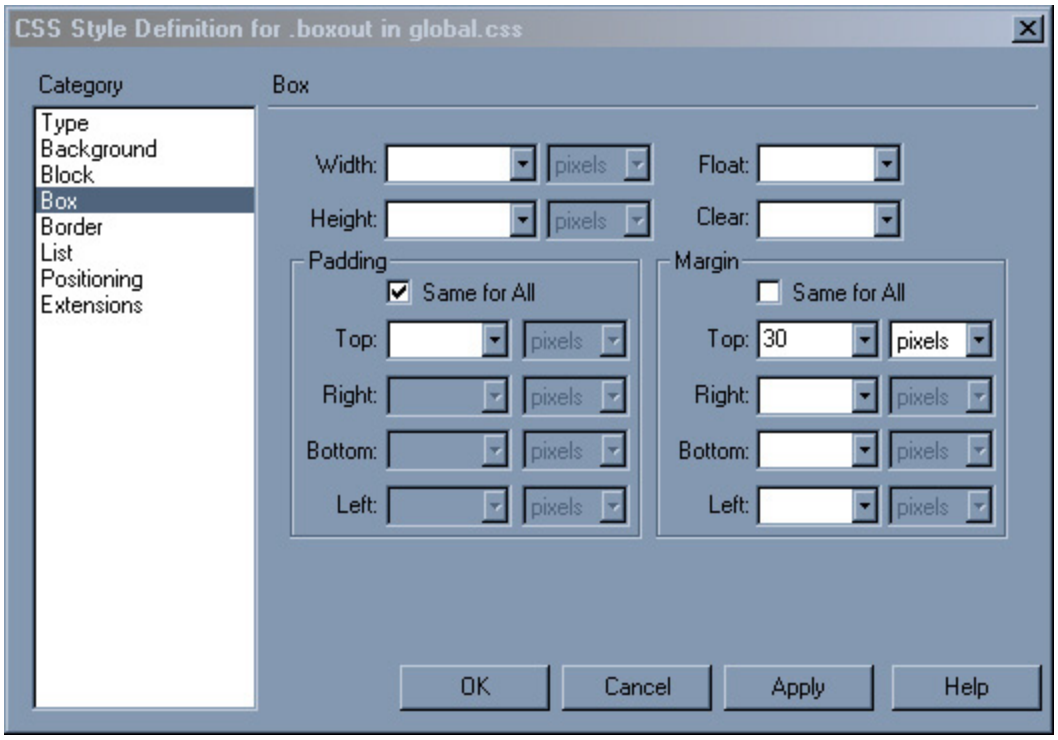
To remove this space open up the CSS style that we created for .boxout p and select the category Box. Here you can change the margin and padding for the element. By setting the top and bottom margin and padding to 1px you will find that the additional space disappears. You might want to add a couple of pixels padding to the left of the text in order to move it slightly off the border. The changes you make don't always show up very accurately in the design view in Dreamweaver MX, as the entire CSS spec has not yet been implemented, so keep checking your work in a browser to see the effect.



Margins and Padding

As you use CSS more, you will often need to change the margin and padding properties on elements in order to get the layout effects that you want. Browsers give most HTML elements a default margin and padding and with straight HTML there is often no way to change it or only a limited set of options – for instance you cannot change the spacing under an <h1> heading using HTML, and setting the spacing and padding on table cells is limited to setting cellspacing and cellpadding when you create the table, at which point it applies to all cells. CSS gives us far more flexibility to change this.

We have used an empty paragraph to separate our navigation and boxout, this is ideally replaced by adding a margin to the boxout with CSS. Click your cursor in between the navigation and the box out and switch into Code View, delete the   when you return to Design View you will find that the boxout div now rests right under the navigation. To get better spacing between these elements we can add a top margin to the class .boxout. Edit this class and select the category Box. Deselect the Same for All Checkbox and add 30pixels in the Margin Top section.



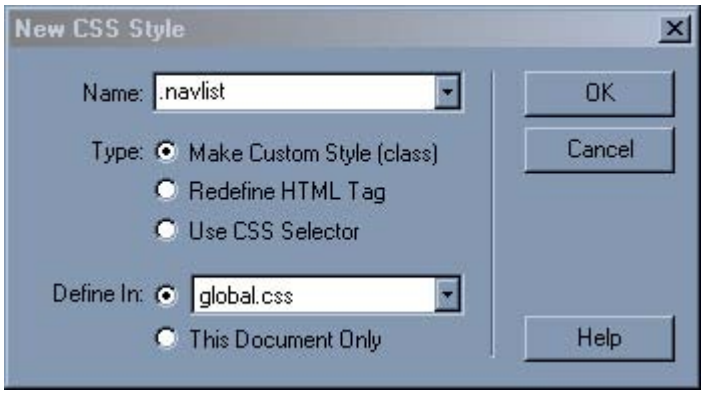
To see the difference between applying margin and padding to an element, try doing the same with padding – you will see that while margin adds white space above the element, padding adds the space between the top of the text and the border, inside the element. It's worth experimenting with these values on various page elements until you feel confident about what they do and how you can use them.

Using a list for navigation

Much navigation on a web site is really a list – it's a list of places that you can visit on the site and so the structural html list tag is a sensible way in which to mark it up. Bearing this and our aim to remove nested tables from our site in mind, let's turn our navigation into a list. We will do as we did with the boxout and create the new navigation underneath the existing in order that we can easily compare them.

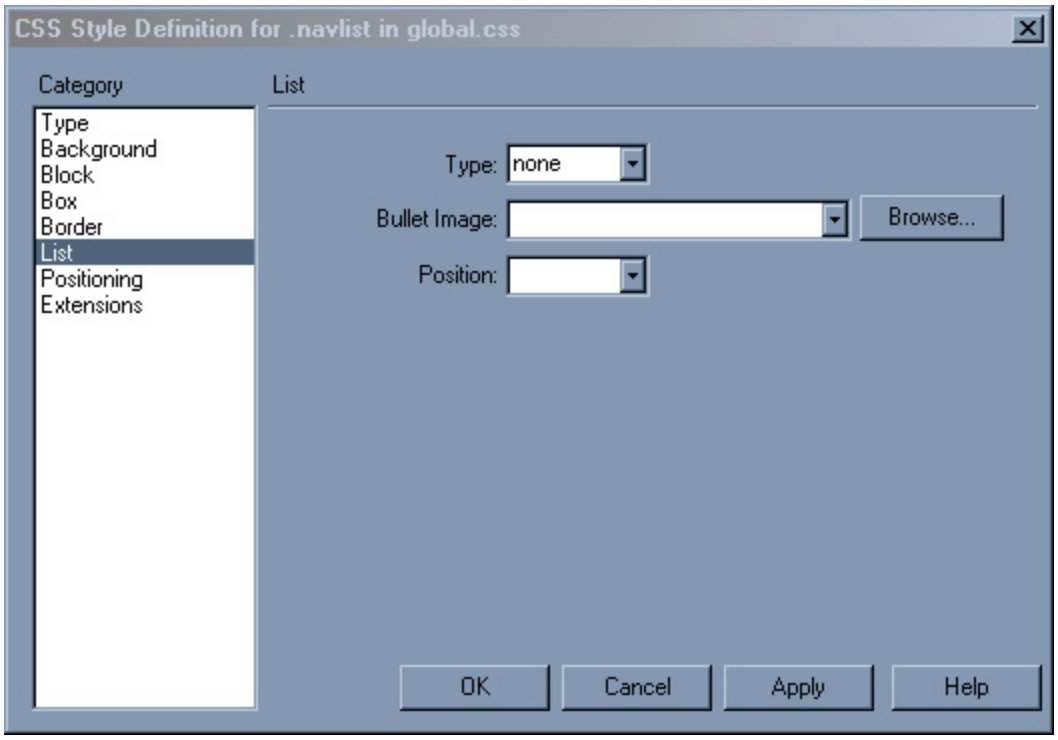
Create an HTML list below the navigation table which contains your navigation items and make them into dummy links by adding # in the Link field of the Property Inspector.

First we need to create a class to apply to the list itself – the <ul> tag. We don't want to style ALL lists in this way so create a new style, select 'Make Custom Style (class)' and call it .navlist.

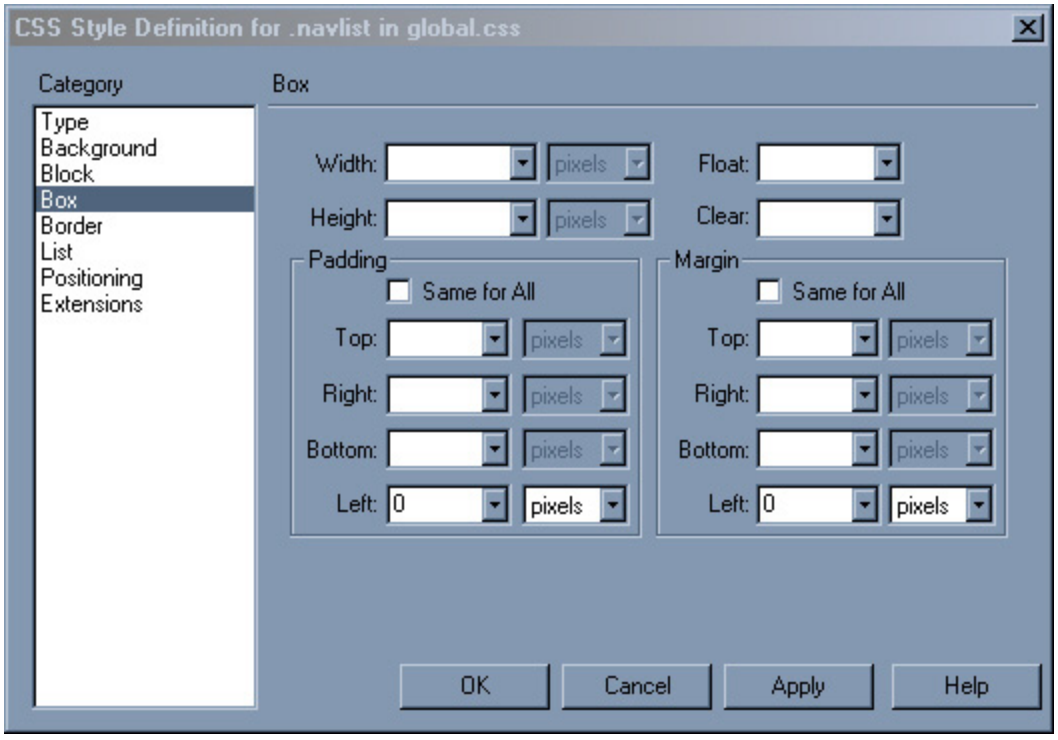


In the next dialog we want to give this list a border, so follow the same steps as those to create a border for .boxout after selecting the 'Border' category. However here set the top and bottom border to be only 1 pixel width, and the left and right borders at 2pixels.

Now select the category 'List'. Under Type select 'none' - this is to remove the default bullet point displayed for each list item.



Most browsers indent the left margin and/or padding of a list - we want to remove this default, select the category 'Box' and set the left margin to 0 pixels. Do the same for left padding.



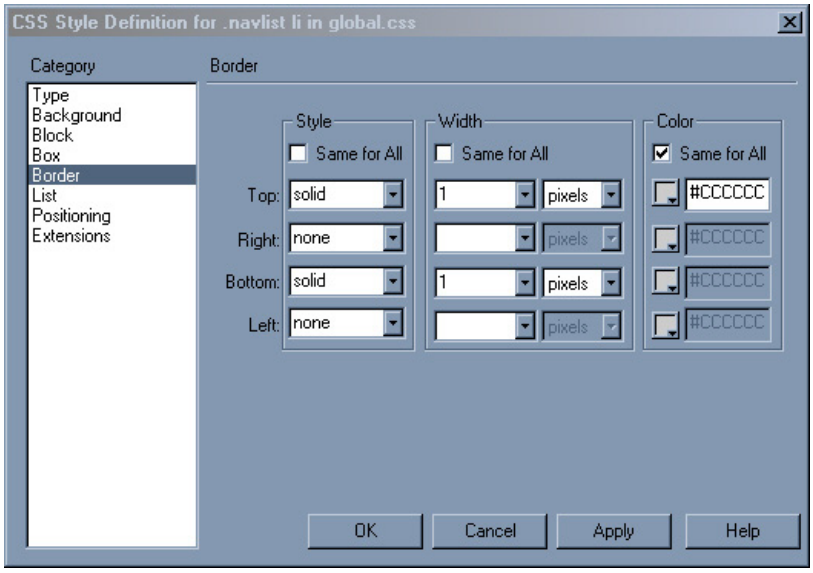
Click OK and then select the <ul> tag and apply the class navlist to it. Most of the changes will display in the Dreamweaver Design View, however the change to the left margin does not display so Dreamweaver MX will continue to render that indenting – check your work in a browser to see it as it will really display.

Now we need to style the individual list items. Create a new CSS style, choose 'Use CSS Selector' and name this .navlist li as we only want to style the <li> tags that are within the list .navlist.

Click Ok and in the next dialog select the category background and set the background color to #ffffff.

In the category Block, set the drop down for Display to 'block'.

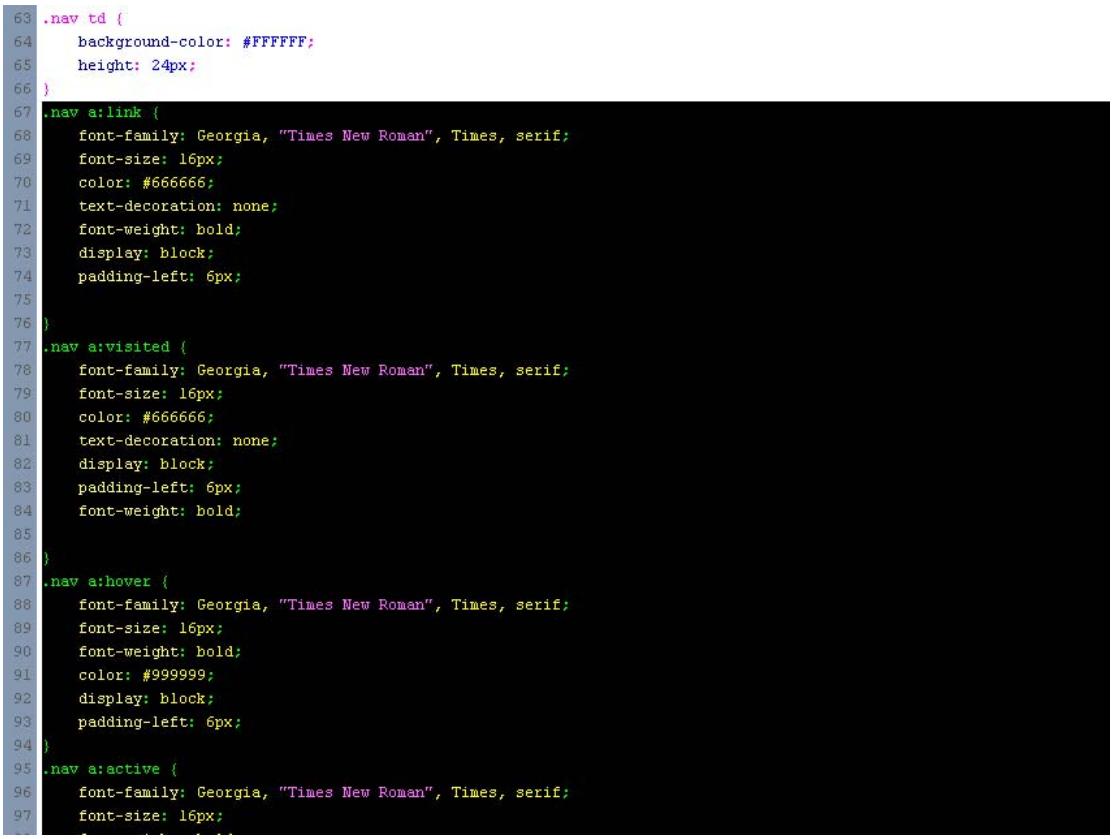
In the category Border set the top and bottom border to 1pixel solid #cccccc as in the screenshot below.



Click OK and you should see most of the changes appear in the Dreamweaver Design View, preview in a browser for the full effect.

Now we just need to apply a style to the link itself. This style is actually no different to the way that the link style was applied to the links within the table cells. The quickest way to recreate that is to look at the stylesheet itself.

In your site files, double click on global.css to open it in Dreamweaver. You will be able to see all of the different classes that we have created. CSS isn't difficult to understand and if you read through the stylesheet you should be able to identify the different things that we did within the Dreamweaver interface. Scroll down till you find the section that starts with the class .nav. You want to select the 4 declarations for .nav a:link, .nav a:visited, .nav a:hover, .nav a:active, which define how the link is styled when unvisited, when already visited, when the cursor is over the link and when the link is clicked.



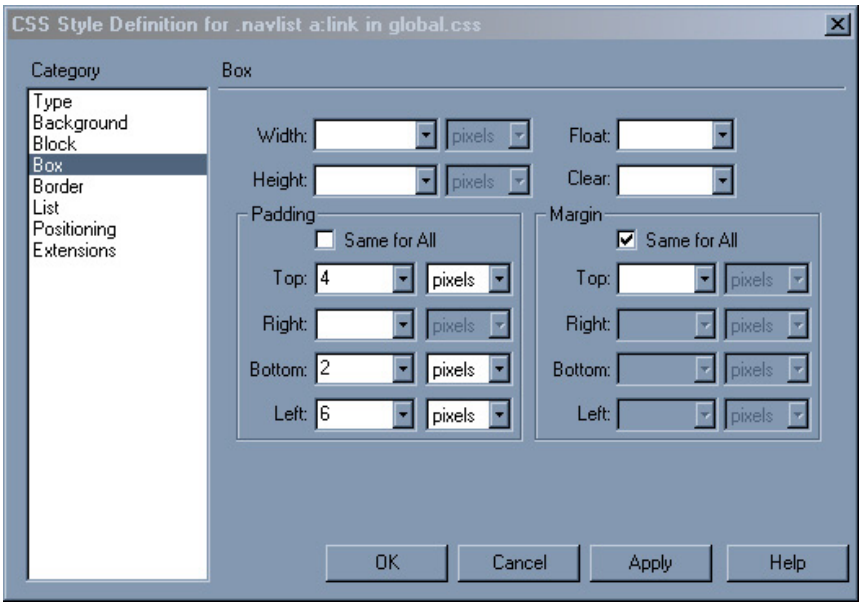
```
.nav td {
    background-color: #FFFFFF;
    height: 24px;
}
.nav a:link {
    font-family: Georgia, "Times New Roman", Times, serif;
    font-size: 16px;
    color: #666666;
    text-decoration: none;
    font-weight: bold;
    display: block;
    padding-left: 6px;

}
.nav a:visited {
    font-family: Georgia, "Times New Roman", Times, serif;
    font-size: 16px;
    color: #666666;
    text-decoration: none;
    display: block;
    padding-left: 6px;
    font-weight: bold;

}
.nav a:hover {
    font-family: Georgia, "Times New Roman", Times, serif;
    font-size: 16px;
    font-weight: bold;
    color: #999999;
    display: block;
    padding-left: 6px;
}
.nav a:active {
    font-family: Georgia, "Times New Roman", Times, serif;
    font-size: 16px;
```

Copy this section then scroll down right to the bottom of the stylesheet and paste them there. Then all you need to do is change the .nav to .navlist

```
.navlist a:link {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     color: #666666;
     text-decoration: none;
     font-weight: bold;
```

```
     display: block;
     padding-left: 6px;

}
.navlist a:visited {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     color: #666666;
     text-decoration: none;
     display: block;
     padding-left: 6px;
     font-weight: bold;

}
.navlist a:hover {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     font-weight: bold;
     color: #999999;
     display: block;
     padding-left: 6px;
}
.navlist a:active {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     font-weight: bold;
     color: #666666;
     text-decoration: none;
     display: block;
     padding-left: 6px;
}
```

Save this stylesheet, switch back to Design View and you should see the change, again: preview in a browser to check that it is all working properly. One final thing to do is to tweak the top and bottom padding on the **.navlist a** declarations to make the buttons slightly taller as in the first section. I added 4pixels padding to the top and 2 pixels to the bottom in the Box category for each state - remember that you need to do this for link, visited, hover and active.



You can now delete the navigation table, leaving the list navigation in its place - before you do so you might like to switch into Code View and see how much neater our new navigation is.

There is a cell above the cell containing the navigation which is only there to push the navigation further down the page, we can get rid of this and apply the same top margin as we did to the boxout to create the same spacing. Merge the two cells by selecting both and clicking 'Merge Table Cells' in the Property Inspector. Then open up the class .navlist and add a top margin.

Two tables to one

We are still left with two tables in our layout. An outer table that does nothing more than hold the content in the center of the page and the content table, which is now split into two cells – navigation and content. We could take this a step further and remove the outer table, then use CSS to center the content table on the page.

Note: If you need your layout to look more or less 'the same' in earlier browsers - in particular Netscape 4 there are going to be some layouts where you will need to retain this kind of nesting. The issue of browser compatibility is one for a separate chapter (see "CSS and Old Browsers") but be aware that once you start using more advanced CSS you must check in several browsers to ensure that your work does not render your page unreadable in any browser. This is easily achieved in Dreamweaver MX 2004, and in Dreamweaver MX is a matter of adding to the list of browsers MX can use for Preview by clicking **Edit > Preferences > Preview in Browser** then the **+** button to add another installed browser to the list. If you don't have a browser that you need to check against, visit http://browsers.evolt.org/ where there is an archive of almost every browser known to man.

First we need to remove that table, switch into Code View and simply delete the table mark-up from the top:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Layout</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="global.css" rel="stylesheet" type="text/css">
</head>

<body>
<table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td height="50" colspan="3"> </td>
  </tr>
  <tr>
    <td width="100"> </td>
    <td height="500" valign="top" bgcolor="#eeeeee">
      <table width="100%" border="0" cellspacing="5" cellpadding="4">
        <tr>
          <td width="192" valign="top"> <ul class="navlist">
              <li><a href="#">Home</a></li>
              <li><a href="#">Photographs</a></li>
              <li><a href="#">Resume</a></li>
              <li><a href="#">Links</a></li>
            </ul>
            <div class="boxout">
              <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nam
                sit amet lorem. Ut sed nulla ut libero tempor egestas. Phasellus
                blandit, purus in facilisis tempus, leo arcu tempor elit, in bibendum
                lacus sem at nunc.</p>
            </div></td>
          <td valign="top"> <h1>Heading One</h1>
            <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nam sit
              amet lorem. Ut sed nulla ut libero tempor egestas. Phasellus blandit,
```

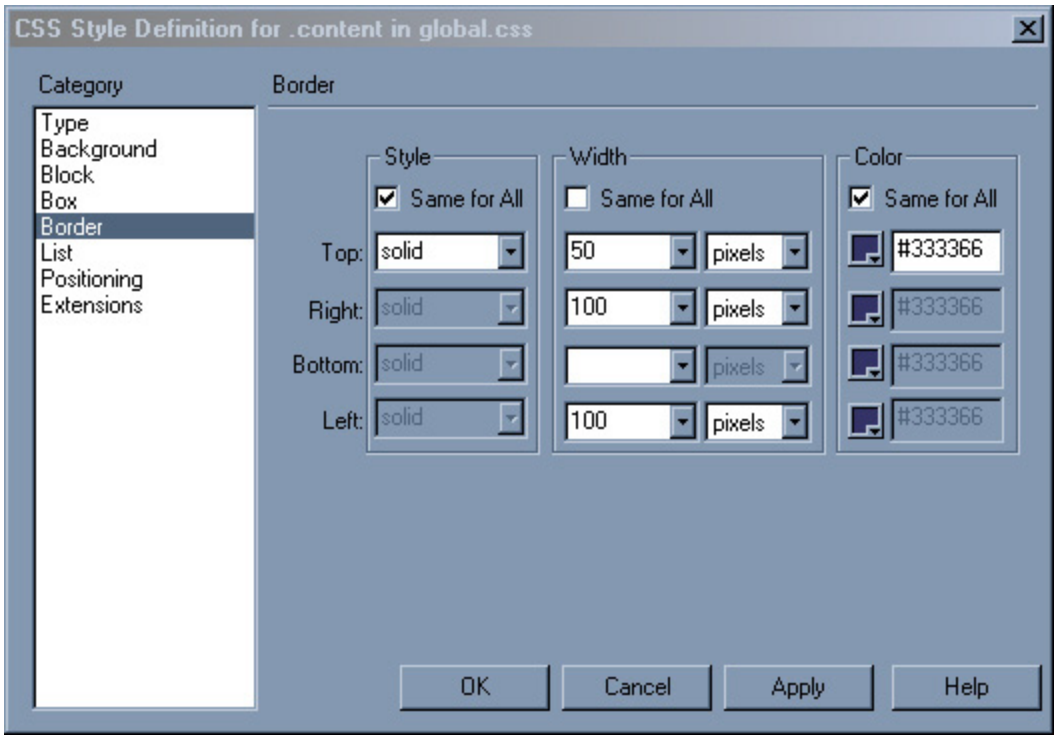
and the bottom of the content table.

```
              id, justo. Proin eget massa sit amet eros faucibus ultricies. </p>
            <p>Pellentesque at urna vitae <a href="#">arcu volutpat viverra</a>.
              Maecenas diam tortor, tincidunt id, condimentum nec, vehicula vel,
              purus. Vivamus adipiscing pharetra est. Nulla ut enim. Ut lacinia,
              arcu bibendum vehicula fermentum, dui neque faucibus diam, et accumsan
              metus sapien vitae wisi. Sed pharetra varius lorem. Integer leo
              elit, vehicula ut, luctus non, tristique eget, ante. Donec tellus
              neque, congue id, gravida in, eleifend vel, metus. Nunc eu augue
              ut lorem vestibulum adipiscing. Praesent tincidunt posuere risus.</p>
            <h2>Heading two</h2>
            <p><img src="images/cork.jpg" alt="A Cork" width="140" height="105" align="right">Quisque
              nec turpis. Nulla facilisi. Pellentesque id enim ut metus vehicula
              ornare. Curabitur vitae mi. Sed sollicitudin est id enim. Sed a
              ligula sit amet wisi elementum pharetra. Sed vitae magna. Suspendisse
              potenti. Praesent eget felis quis eros hendrerit aliquam. Nunc facilisis
              adipiscing sem. Pellentesque ac justo. Vestibulum massa risus, euismod
              sit amet, interdum sed, mattis sed, arcu. Integer iaculis justo
              id nulla. Phasellus vestibulum.</p>
            <p></p></td>
        </tr>
      </table>
    </td>
    <td width="100"> </td>
  </tr>
  <tr>
    <td height="40" colspan="3" class="footer">
<div align="center">contact: <a href="mailto:me@mydomain.com">me@mydomain.com</a></div></td>
  </tr>
</table>
</body>
</html>
```

After doing this, go to the opening table tag (which is now the table containing the content) and delete all of the attributes of that tag so you are left with the simple <table> now add a class to that tag <table class="content">

Switch back to Design View and don't be alarmed by the fact that the content area now is the same color as the background and at the top of the screen. Create a new class named '.content'.
In the category Background set the background color to #eeeeee;
In the category Border set the border values as follows:

Style	Same for All – solid
Width	Top: 50pixels
	Right: 100 pixels
	Left: 50pixels
Color	Same for all - #333366



Click OK and you should see your layout snap back to where it was before (if it doesn't check that the class .content is applied to that content table).

When we deleted the bottom of our table we removed the contact link. We can put that back now. In a new paragraph under the table add the text of the contact link, apply the class .footer to it. The text will change size and color but because it is now not contained within a table cell it will stick to the left side of the page. To center it edit the class .footer and in the category Block set Text-align to 'center'.

You can now easily tweak this layout until you are happy with the results. The content is now very close to the edges of the table because we removed the table cellpadding - you can put this back by adding padding to the .content class. The advantage of using CSS to do this is that even if you had created 100 pages from this layout, to make changes

to the width of the area around the content, or to the color of that main table background - you only need to tweak the values in the stylesheet and it will effect all pages that use that stylesheet.

Final touches

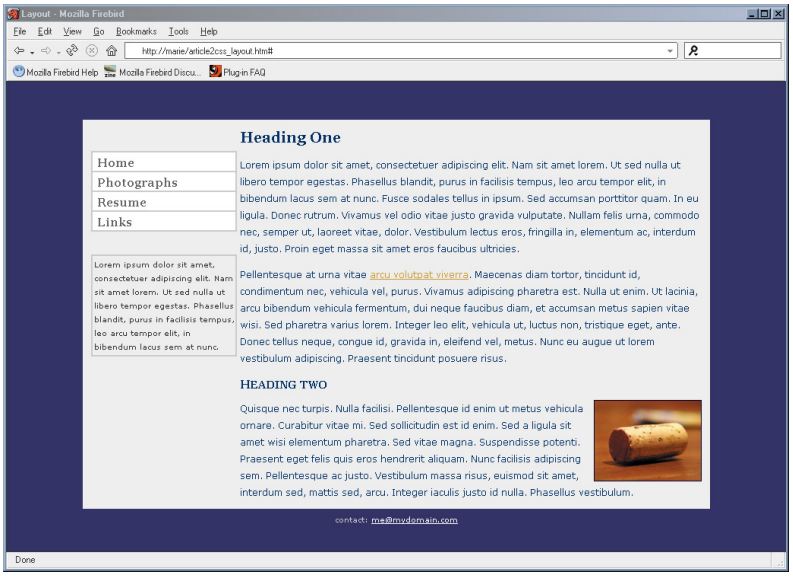
I want to close this chapter with a look at the ways in which you can use CSS to have greater control over the text on your pages.

Edit the CSS style that is defined for the <p> tag.

In the Type category you will see there is a value for line height - set this to 22 pixels. In Dreamweaver you should see the spacing between the lines of body text increase, although MX doesn't calculate this too well so remember to check your work in a browser too. If you don't want the text in the boxout to be so widely spaced - open up the class'.boxout p' and set the line-height to something smaller - I've chosen 18 pixels. Take care not to space your text so widely that it becomes hard to read!

Edit the definition for the <h2> tag - in the category Type under 'Variant' choose 'Small Caps', and your title will transform into 'small caps' text, with the first letter slightly larger than the others.

We can put a border around our cork image, create a new custom class and call it .imgborder - this class can be used for any images that you would like to have enclosed with a neat border. In the category Border create a solid, 1 pixel border with a color of #000033. Hit Ok and then apply this class to the <img> tag of our cork picture. As with many of these things MX doesn't display this - so check in a browser to see the effect. Here's my final layout.



3.Page layout with CSS: Layers and CSS Positioning

In this chapter we will be exploring the subject of CSS layouts in Dreamweaver. We will take a tables-based layout and rebuild it using CSS, and then explore CSS positioning using CSS in an external stylesheet, but we take as a starting point the 'layer' feature in Dreamweaver. After completing this chapter you will understand the basics of CSS positioning and how to work with these layouts in Dreamweaver.

You may have heard CSS advocates saying that tables were never really meant for layout, and this is true; tables were developed in the HTML specification for the laying out of tabular data - something like what you would find in a spreadsheet. Modern, fully featured graphical web browsers are fine displaying tables when they have been used for layout. However more limited devices including some PDAs, text-only browsers and devices such as Braille and screen readers that are used by the visually impaired will simply read the document starting at the top left hand cell, working across the page and then moving onto the next line. For a simple layout this may not be a problem – however where tables have been used in a complex way, nested inside one another the content can quickly become totally unintelligible to the user.

Some Lynx links

One way to get a feel for how a web page will be read by a screen reader or other text only device is to look at the page in Lynx (http://lynx.browser.org/). Lynx is a text-only browser and you can install a copy of it on your own computer or use the online Lynx emulator at http://www.delorie.com/web/lynxview.html.

If your content is easy to follow and your site navigable when viewed here, then you have achieved much of what is necessary to create an accessible web site.

As we saw in Chapter 2, if you don't feel confident to move to pure CSS for layout then the best solution is to move to as clean a tables layout as possible, avoiding nested tables and bearing in mind the way that the page would display in a device that would read the page as described above.

It is not impossible to make a tables-based layout accessible, however for a complex layout that would require nested tables for precise control over page elements it can become very difficult. CSS gives you the ability to create complex layouts while still retaining accessibility for all users.

CSS Layouts in Dreamweaver

As I mentioned in the introduction to this chapter, we will start by looking at the layer feature in Dreamweaver – layers are simply Dreamweaver-speak for inline CSS positioning and not to be confused with the Netscape 4 proprietary layer tag.

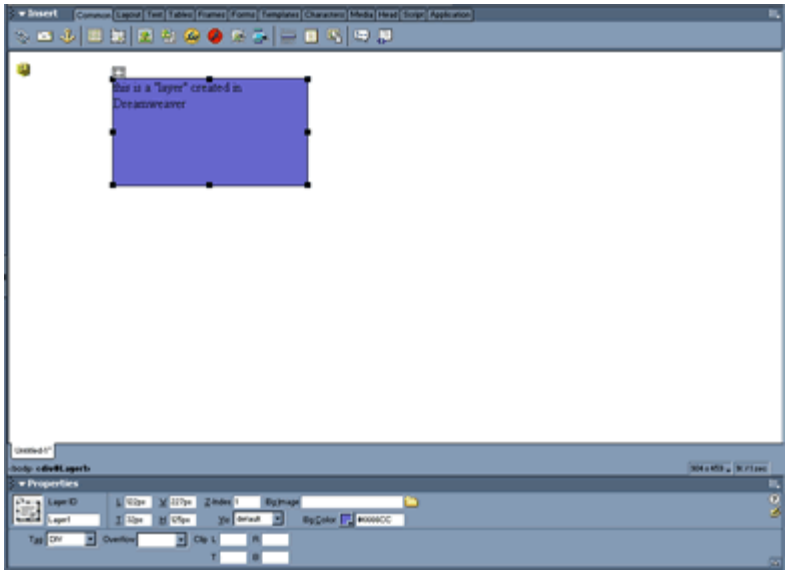
Creating a layer

In a new blank document in Dreamweaver, click on the 'Draw Layer' icon on the common tab of the Insert toolbar.

The Insert Layer icon (circled) on the Insert Toolbar

You will find that your cursor turns into a crosshair and you can draw a box on the Design View window.

Once you have drawn your layer, you can click inside it to add text or images. If you select the layer by clicking on the outer edge you can change the background color and size of the layer in the Property Inspector. You can use percentage or pixel widths for the layer just as you would for a table cell.



Drawing a Layer in the design view of Dreamweaver MX

Switch into Code View to have a look at what Dreamweaver inserts.

The mark-up for the layer shown in the image above looks like:

```
<div id="Layer1" style="position:absolute; left:122px; top:32px;
width:227px; height:125px; z-index:1; background-color: #6666CC; layer-
background-color: #6666CC; border: 1px none #000000;">this
  is a &quot;layer&quot; created in Dreamweaver</div>
```

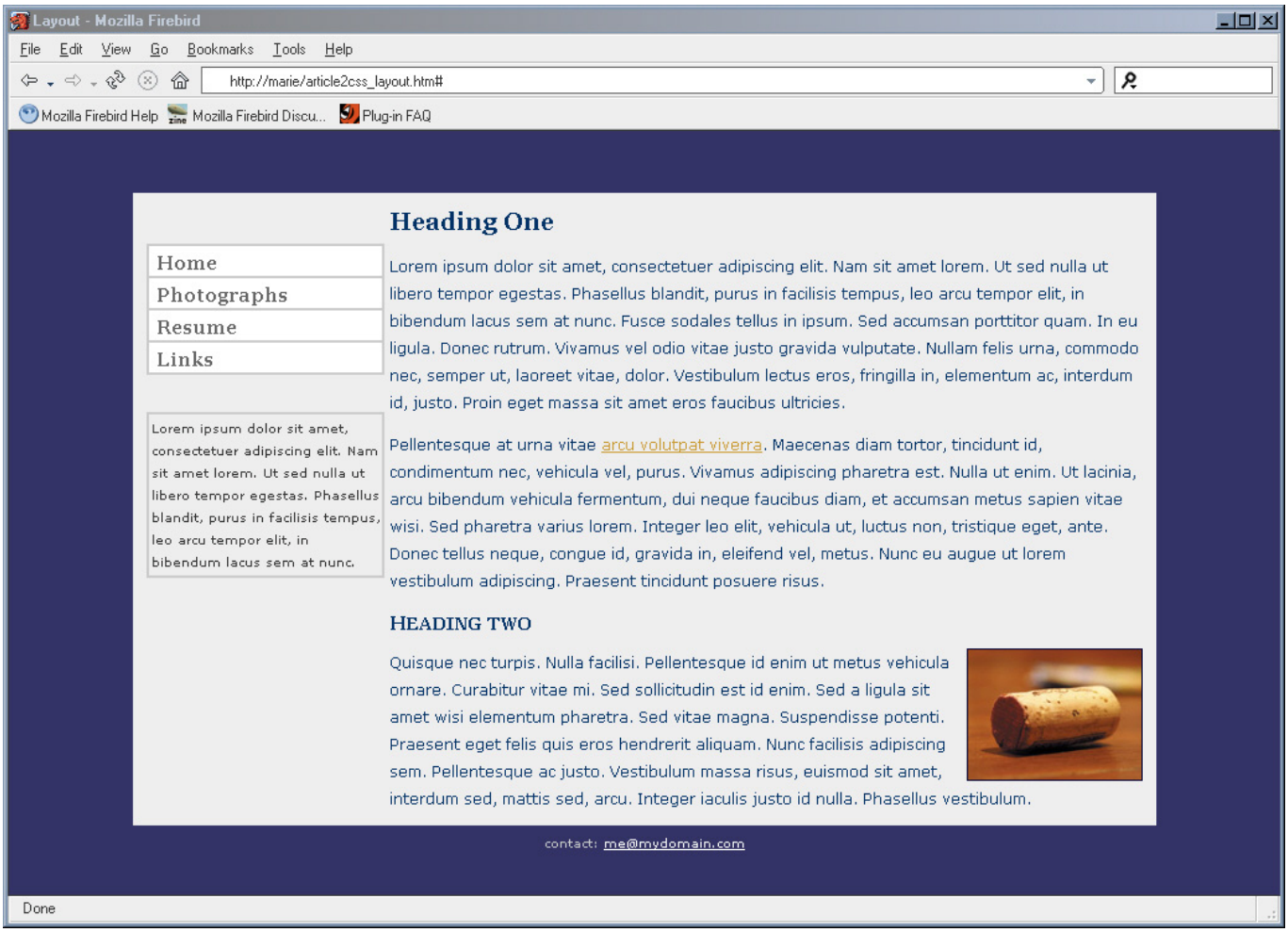
You will recognize that this is CSS, – but note that it is applied to the tag itself and not in an external style sheet.

Netscape 4 and CSS

Dreamweaver also adds a section of JavaScript to the head of the document. This is to get round a problem with Netscape 4 browsers where if the document is resized and it contains CSS positioning, the positioned areas all jump to the left hand side of the document. The 'Netscape Resize Fix' simply reloads the page if the window is resized. If you are not concerned about Netscape 4 you can remove it, otherwise a good plan is to put it into an external JavaScript file so that you do not need to have it on every page of your site.

Another Netscape 4 specific item added by Dreamweaver is the rule layer-background-color in the CSS – again this is so that Netscape 4's layer tag will pick up the style information. The CSS will not validate with this in. Again it is up to you whether you decide to remove it or keep it in.

We can use the layer feature in Dreamweaver to begin to recreate the tables based layout I in the last chapter.



The simple tables based layout created in the last chapter.

In a new document in Dreamweaver, select Draw Layer, and draw a layer roughly in the center of the Design View.

One thing you need to remember about these layers is that they are absolutely positioned from the top and left of the document, which means that even if you specify them with percentage widths they will always remain the same distance from top and left – only the right hand margin will stretch so set the values for top and left 't' and 'l' in the Property Inspector to 0px and set the values for height and width 'h' and 'w' to 100% (we will remove the height and width values later, but if we don't put them in you will find it difficult to work with the layer in Dreamweaver). While you are in the Property Inspector name the layer 'content'.

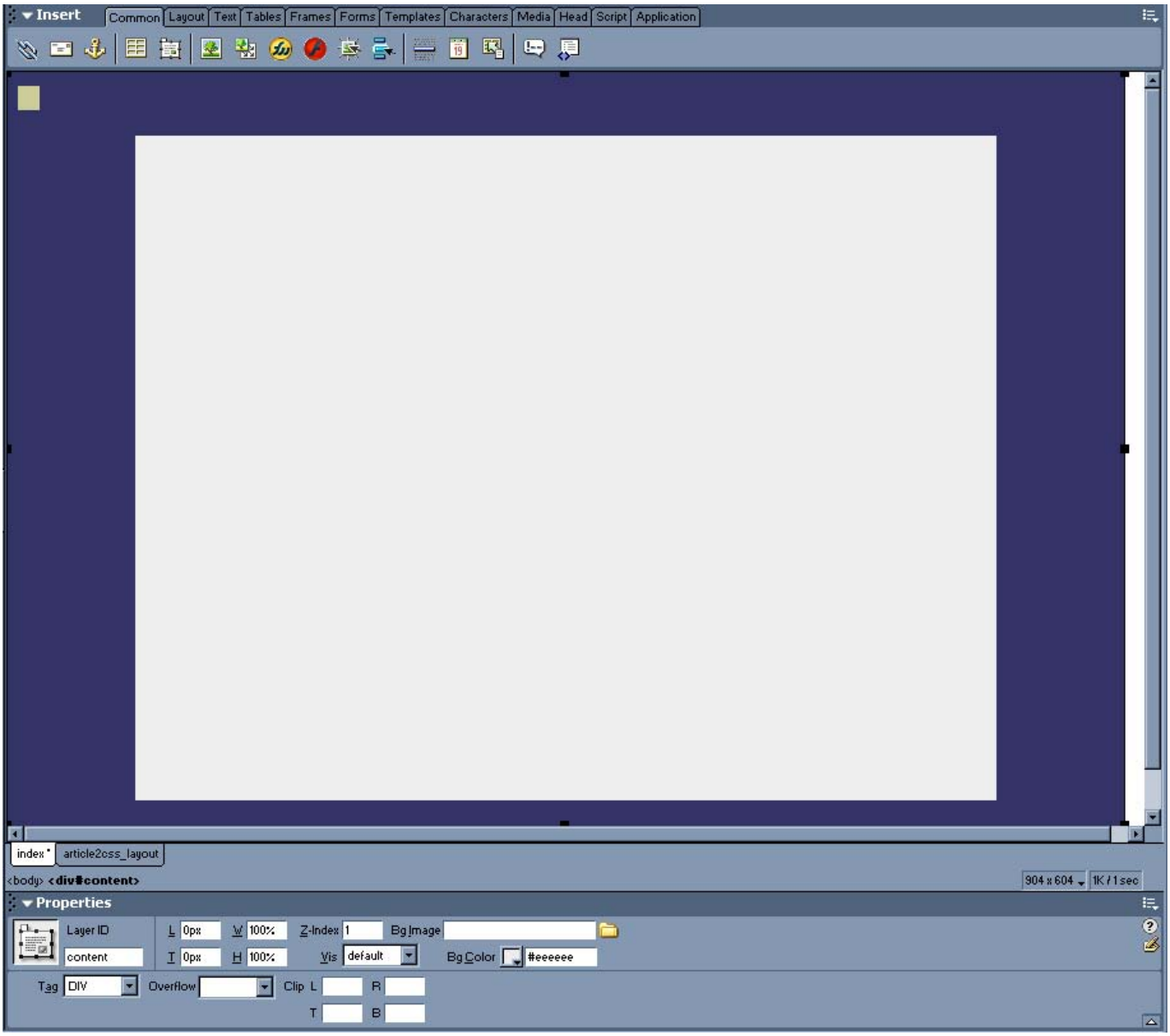
We want to add a deep border to this area in order to create a centered box that stretches with the page. Dreamweaver does not have any function to let you do that, however it is simple to add it to the rules for this layer. Switch into Code View and at the end of the style rules for this layer add the following:

```
border-top: 50px solid #333366; border-left: 100px solid #333366; border-
right: 100px solid #333366; border-bottom: 20px solid #333366;"
```

This makes your entire section for this area look like this:

```
<div id="content" style="position:absolute; left:0px; top:0px;
width:100%; height:100%; z-index:1; background-color: #eeeeee; layer-
background-color: #eeeeee; border-top: 50px solid #333366; border-left:
100px solid #333366; border-right: 100px solid #333366; border-bottom:
20px solid #333366;"></div>
```

In Dreamweaver Design View my layer now looks like this:



The #content Layer in Design View

Note: when working with a CSS layout in Dreamweaver MX, take care that you do not accidentally drag the positioned areas around, as Dreamweaver will cope with this by either altering the inline style properties, often by changing percentage widths to pixel widths, or add strange values to your external style sheet.

CSS Positioning in an External Style sheet

This method of writing CSS for layout using inline CSS, while perfectly valid, means that you lose out on many of the benefits of having a CSS layout that you will get should you use external CSS. Using inline CSS, if you created 20 pages using layers and then wanted to change the background color of an area of the page you would need to edit each page individually. Using an external style sheet means that you only need to change the background-color once and all of the pages that use the style sheet will pick it up. If you have been styling text in an external style sheet you will understand how easy it is to change styles of fonts on your entire page just by changing the one style sheet, using CSS for layout in the external style sheet gives you this same control over your page layout. Additionally by only needing to write out that information once in your style sheet as opposed to in every page, you make your file sizes smaller and pages quicker to load.

Let's look at how we can move these rules to our external style sheet.

Moving the rules to an external style sheet

I already have an external style sheet that simply contains some simple rules for text formatting, the content of that style sheet is below if you wish to copy and paste it, or you can use your own, or simply create a new blank style sheet and attach it to your page.

```
body {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     font-size: 12px;
     color: #333366;
     background-color: #333366;
     margin: 0px;
     padding: 0px;


}
p {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     font-size: 12px;
     color: #003366;
     line-height: 22px;

}

h1 {
```

```
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 20px;
     color: #003366;


}
h2 {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     color: #003366;
     margin-bottom: 0px;
     padding-bottom: 0px;
     font-variant: small-caps;



}
.footer {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     font-size: 10px;
     color: #CCCCCC;
     text-align: center;
     margin-top: 0px;
     padding-top: 0px;

}
a:link {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     color: #CC6600;
}
a:visited {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     color: #CC9933;
}
a:hover {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     color: #CC6600;
}
a:active {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     color: #CC6600;
}
.footer a:link {
     color: #FFFFFF;
}

.nav a:link {
     font-family: Georgia, "Times New Roman", Times, serif;
```

```
     font-size: 16px;
     color: #666666;
     text-decoration: none;
     font-weight: bold;
     display: block;
     padding-left: 6px;

}
.nav a:visited {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     color: #666666;
     text-decoration: none;
     display: block;
     padding-left: 6px;
     font-weight: bold;

}
.nav a:hover {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     font-weight: bold;
     color: #999999;
     display: block;
     padding-left: 6px;
}
.nav a:active {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     font-weight: bold;
     color: #666666;
     text-decoration: none;
     display: block;
     padding-left: 6px;
}

.boxout {
     margin-top: 30px;
     border: 2px solid #cccccc;
}

.boxout p {
     font-family: Verdana, Arial, Helvetica, sans-serif;
     font-size: 10px;
     color: #333333;
     padding-top: 1px;
     padding-bottom: 1px;
     margin-top: 1px;
```

```
     margin-bottom: 1px;
     padding-left: 2px;
     line-height: 18px;
}
.navlist {
     list-style-type: none;
     margin-left: 0px;
     border-top: 1px solid #cccccc;
     border-right: 2px solid #cccccc;
     border-bottom: 1px solid #cccccc;
     border-left: 2px solid #cccccc;
     padding-left: 0px;
     margin-top: 30px;
     margin-right: 0px;
     margin-bottom: 0px;

}
.navlist li {
     background-color: #ffffff;
     display: block;
     border-top-width: 1px;
     border-bottom-width: 1px;
     border-top-style: solid;
     border-right-style: none;
     border-bottom-style: solid;
     border-left-style: none;
     border-top-color: #CCCCCC;
     border-right-color: #CCCCCC;
     border-bottom-color: #CCCCCC;
     border-left-color: #CCCCCC;
}

.navlist a:link {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     color: #666666;
     text-decoration: none;
     font-weight: bold;
     display: block;
     padding-left: 6px;
     padding-top: 4px;
     padding-bottom: 2px;


}
.navlist a:visited {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
```

```
     color: #666666;
     text-decoration: none;
     display: block;
     padding-left: 6px;
     font-weight: bold;
     padding-top: 4px;
     padding-bottom: 2px;

}
.navlist a:hover {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     font-weight: bold;
     color: #999999;
     display: block;
     padding-left: 6px;
     padding-top: 4px;
     padding-bottom: 2px;

}
.navlist a:active {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 16px;
     font-weight: bold;
     color: #666666;
     text-decoration: none;
     display: block;
     padding-left: 6px;
     padding-top: 4px;
     padding-bottom: 2px;

}
.imgborder {
     border: 1px solid #000033;
}
```

Open up the style sheet in the Code View.

In the Code View for the page that we created with the layer, select the rules that are attached to the div tag. This is everything between the quotation marks of style.

```
21
22 <div id="content" style="position:absolute; left:0px; top:0px; width:100%; height:100%; z-index:1; background-color:
   #eeeeee; layer-background-color: #eeeeee; border-top: 50px solid #333366; border-left: 100px solid #333366; border-right:
   100px solid #333366; border-bottom: 20px solid #333366;"></div>
23 </body>
24 </html>
```

The rules select in the Code View

Copy this to your clipboard and switch to your style sheet. Type the following into your style sheet:

```
#content {

}
```

then paste the rules that you copied in between the curly brackets:

```
#content {
     position:absolute; left:0px; top:0px; width:100%; height:100%; z-
index:1; background-color: #eeeeee; layer-background-color: #eeeeee;
border-top: 50px solid #333366; border-left: 100px solid #333366; border-
right: 100px solid #333366; border-bottom: 20px solid #333366;
}
```

Return to your page and delete the style attribute and all the rules so you are simply left with:

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

Save your page, view it in Dreamweaver or in a browser and you should see that the page remains the same, except that now the mark-up for the layer (or 'div' as it really should be known) is in your external style sheet.
You could continue on adding elements to this page in the same way – using layers and then pasting the rules into your external style sheet – or you can simply set up your divs in the Code View and then work on the CSS to add the rules yourself.

To add the divs by hand simply switch into Code View and type them in – for example: in my layout the main content area is split into two, a left hand column for navigation and a right hand column which is for text. These are inside the div 'content'. In Code View, add two div tags inside the content div. Put some dummy text in there just so you can see where they are.

```
<div id="content">
<div id="side">navigation here</div>
<div id="main">page content here</div>
</div>
```

In Design View you will see that these just appear one under the other as you would expect, as the divs have no rules applied to them to tell them how to behave.

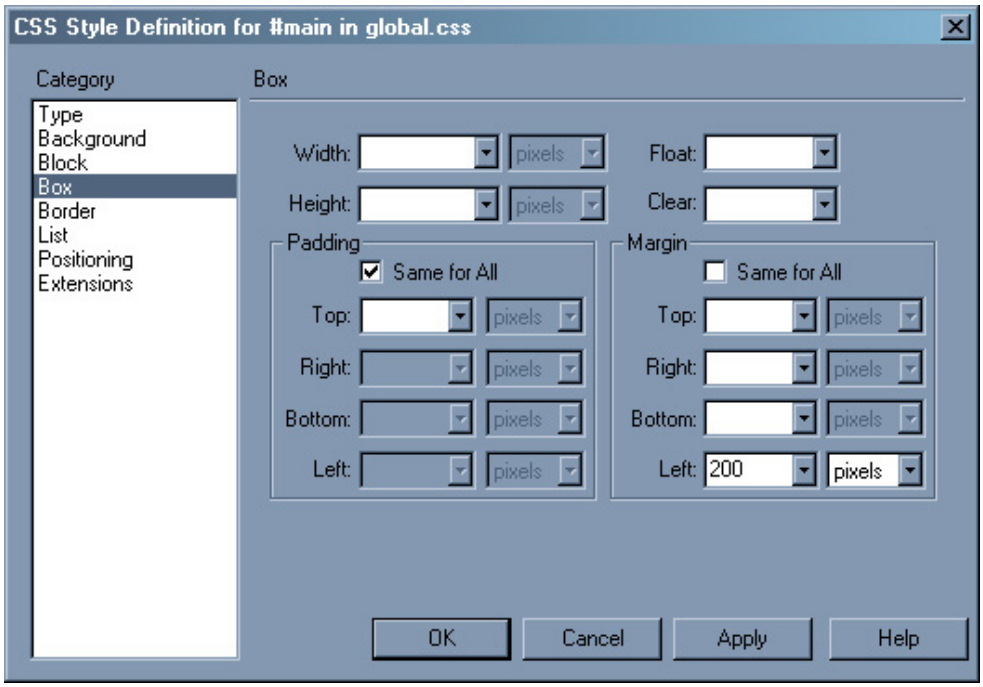
Now switch to your style sheet and add the following:

```
#side {

}

#main {

}
```

Save the style sheet, switch back to the page and you will see that these areas are now showing up in the CSS Styles Panel which means you can now set their properties from the CSS dialogue.

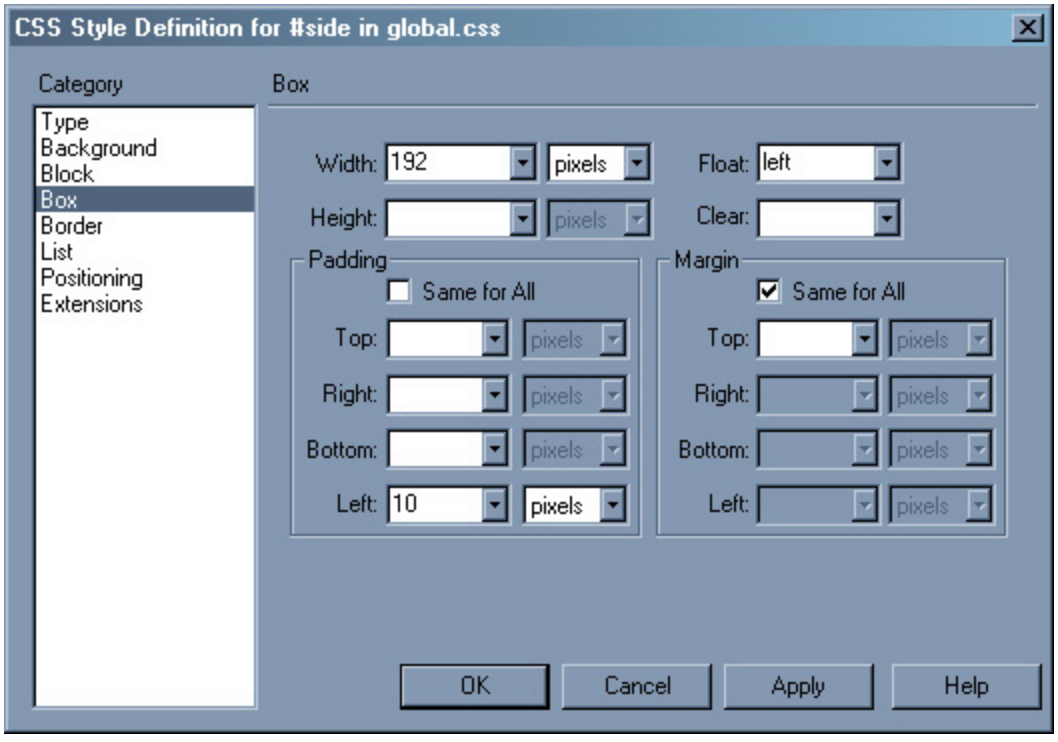
The CSS Styles Panel

Edit the definitions for #main. In the Box category give it a left margin of 200px:



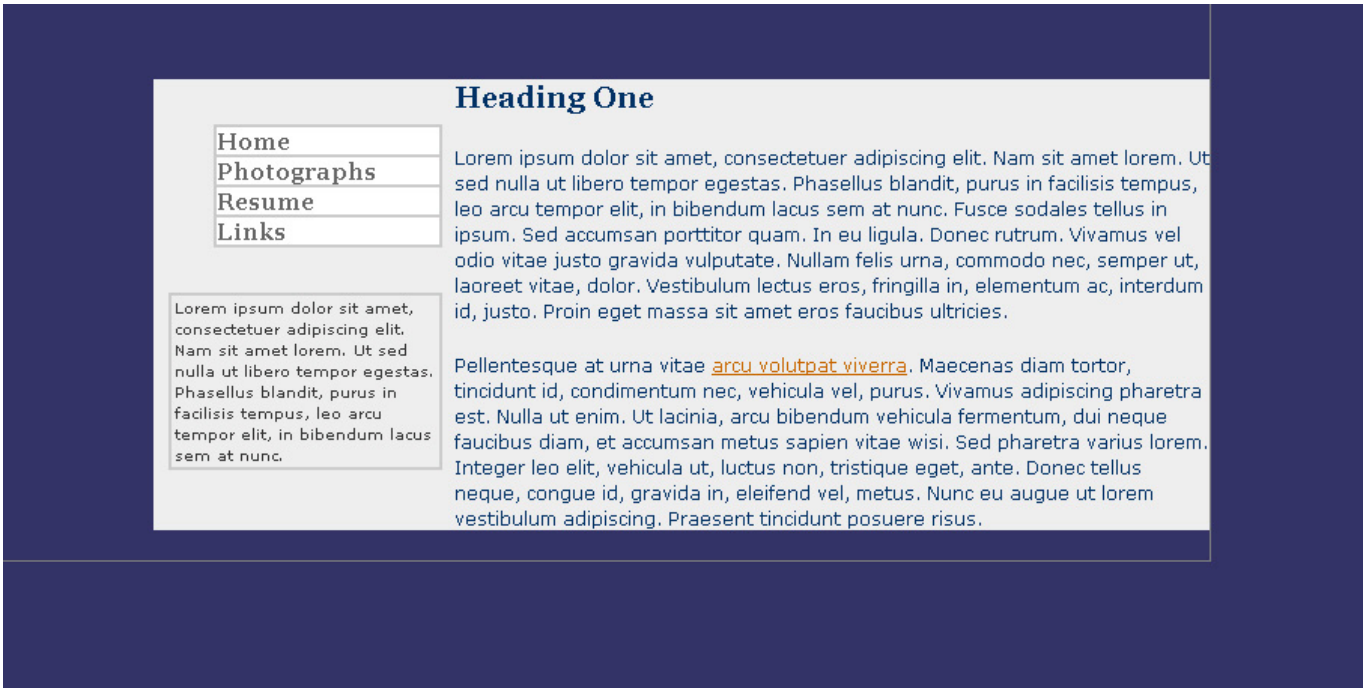
Editing the CSS Style Definitions for #main

Click ok, the text for the content area will move over to the right.
Now edit the definitions for #side. In the Box category set the width to 192pixels, Float to 'left' and padding left to 10pixels.



Editing the CSS Style Definition for #side

In the #side div, I have added the mark-up for the navigation and small boxout that I used in the last chapter, and also some dummy content for the text area. I then switched to my style sheet and removed the attributes set on #content for height and width. This left me with a layout that looks like this.



The layout in the Design View

The HTML mark-up for this page is below (filler text removed for brevity!).

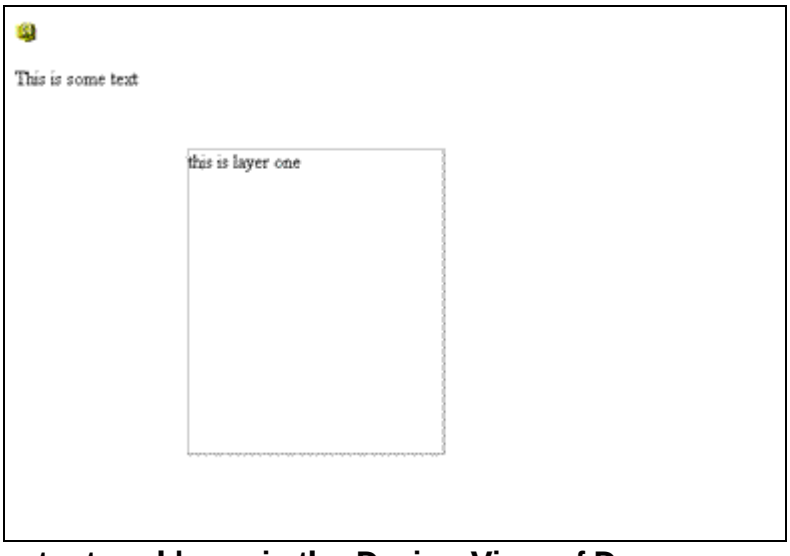
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>CSS Layout</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="global.css" rel="stylesheet" type="text/css">
</head>

<body>

<div id="content">
  <div id="side">
    <ul class="navlist">
      <li><a href="#">Home</a></li>
      <li><a href="#">Photographs</a></li>
      <li><a href="#">Resume</a></li>
      <li><a href="#">Links</a></li>
    </ul>
    <div class="boxout">
```

```
      <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nam
sit amet
        lorem. Ut sed nulla ut libero tempor egestas. Phasellus blandit,
purus
        in facilisis tempus, leo arcu tempor elit, in bibendum lacus sem
at nunc.</p>
    </div>
  </div>
<div id="main">
      <h1>Heading One</h1>
      <p>Main page text here<p>
</div>
</div>

</body>
</html>
```

CSS Positioning Techniques

To create this layout we have used a variety of CSS positioning techniques. The CSS that controls the positioning of our main page area (the grey box) is positioned using absolute positioning. Absolute positioning is the technique used by Dreamweaver to position its "layers". When positioning something using absolute positioning you take it right out of the flow of the document.

For example, in a new document draw a layer using Dreamweaver and type some text into it, drag it to the center of the document. Now click your mouse cursor at the top of the Design View window and type a sentence. The sentence will remain at the top of the document.



The text and layer in the Design View of Dreamweaver

If you switch into Code View you can see that the sentence outside the layer comes after the content of the layer.

```
<div id="Layer1" style="position:absolute; left:149px; top:117px;
width:208px; height:248px; z-index:1">this
  is layer one</div>
<p>This is some text </p>
```

If you delete the style attribute of this div while in Code View, so that you end up with the following mark-up:

```
<div id="Layer1">this
  is layer one</div>
<p>This is some text </p>
```

Then switch back into Design View you will see that the content has returned to the logical order in which it is found within the html. In complex layouts you can use this to your advantage as you can order the actual content in the most appropriate way for those using devices and browsers that have no support for CSS, but lay the page out for graphical browsers exactly as you want it to display.

Float

We have also used float to position our side bar area. Float is often used to allow text to wrap around images within a paragraph (in the way we used to use align="right" on image tags). However it can be used on any item that you want to position within its container without having to absolutely position it.

To see an example of float, open a new document in Dreamweaver and type:

This is my logo

This is some banner text
In Code View this should look like:

```
<p>This is my logo</p>
<p>This is some banner text</p>
```

Create a new CSS style (Custom Style), name it logo, and in the Box Category select Float: left.

Create another CSS style and call this one bannertext, in the Box category select Float: right.

Now apply the CSS class logo to the <p> tag of the "this is my logo" text and the bannertext class to the other text. You should see the two elements end up at either side

of the top of the document. Here is the complete document - I have inserted the CSS in the head of the document so you can see just how little mark-up goes into creating this.

```
<!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>
<title>Float Demo</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>
<style type="text/css">
<!--
.logo {
     float: left;
}
.bannertext {
     float: right;
}
-->
</style>
</head>

<body>
<p class="logo">This is my logo</p>
<p class="bannertext">This is some banner text</p>
</body>
</html>
```

By positioning images, or CSS styled text in this way, you can replace the need to use a 2 cell table to get this kind of effect. For the user reading the page with a text only device, as long as you use alt text on your logo, they will be able to read your company name and strapline in the banner easily and understand exactly where they are.

There are huge amounts of ways in which you can position page elements using CSS. As always, experimenting with these ideas is the best way to learn how these techniques work - with just these simple techniques you can begin to create interesting layouts, doing things that wouldn't be possible using tables as well as replacing tables.

4.Borders, Backgrounds, Blocks & Boxes

CSS is the language of Web design--a language that transcends the limitations of presentational markup and offers new opportunities. Designers are just now getting to explore those opportunities. The support for CSS in Dreamweaver MX is fairly decent, better in MX 2004, and especially helpful for the designer just starting out with CSS as you can use a range of dialogs to set numerous useful properties. However, if you're working with more complicated CSS layouts or want to edit your CSS all at once it does mean doing a lot of hand-authoring.

Before we get into the actual creation of designs, I want to provide you a two-part foundation overlook that combines a look at some great CSS designs and describes what Dreamweaver MX offers by way of its CSS related tools, and how you can use Dreamweaver to style many aspects of your designs such as borders, backgrounds, blocks, boxes, and lists--culminating in a terrifically well-optimized yet fully visual design.

Working with CSS in Dreamweaver MX

While you can create inline styles using Dreamweaver MX, I'm going to focus on how you can define the styles you need for your page or site.

To create a new style, follow these steps:

1. From the CSS Styles panel, click the New Style button.
2. The New CSS Style dialog box appears. In the Define In field, choose to add the new style to an external style sheet.
3. Select a style type:

    - **Make Custom Style (Class**) - Creates a class style. If you select this option, you need to name the class in the Name field above the style type selector. If you don't precede the class name with a period (.), as is required by the style sheet, Dreamweaver adds it for you.

    - **Redefine HTML Tag** - Applies a style to an HTML tag. When you select this option, you also must select a tag from the Tag field above the style type selector. These styles are automatically applied to the appropriate tags after they're defined.

    - **Use CSS Selector** - Applies a style to one of the link types listed in the Selector field above the style type selector. These styles enable you to remove the underlining from links and otherwise change the appearance of the various link states. They're automatically applied after they're defined.

4. Click OK.
5. The Style Definition dialog box opens. Set the style rules by choosing from the various style categories and options

6. Click OK to complete the style definition and return to the Document window.

After you create a style, it's easy to apply. Styles defined for an HTML tag are automatically applied when viewed in a browser. Class styles are applied by doing the following:

1. In the Document window, select the content to which you want to apply the class style.

2. In the CSS Styles panel, select a style from the list.

If the selection is only a small portion of content within a tag, the tag is used with the class attribute. If the selection extends across multiple paragraphs or tag pairs, the style is applied using the tag.

Setting a Background

The Background offers control over background images and colors. Not only do these styles ensure consistency throughout the site, but they also offer greater control over the repeating and scrolling of background images.

The properties in this category are:

- **Background Color** - Sets the background color for an element. This style can be applied to the <body> tag to set a color for the entire page. It can also be applied to <p> and other tags to set a background color only for that particular selection. Using this style with link tags makes them stand out on the page.

- **Background Image** - Sets a background image for the page or element. This is most commonly used with the <body> tag or table cells (<td>).

- **Repeat** - Sets the repeat tiling for a background image. No Repeat sets the image to display from the upper-left corner of the element to which it's applied and not repeat at all. Repeat tiles the image horizontally and vertically as needed to fill the entire area used by the element. Repeat-x tiles the image horizontally, but not vertically. Repeat-y does the opposite.

- **Attachment** - Sets the scrolling for the background image. A fixed image remains anchored to its original position, even as the text is scrolled. This creates the effect of the text moving over the background image and also enables you to set a background image to specific dimensions to avoid tiling. A scrolling image scrolls with the text, which is the default.

- **Horizontal Position** - Sets the initial horizontal position of the background image. The position can be set with numerical coordinates or relative to the positioning of the element to which the style is applied.

- **Vertical Position** - Sets the initial vertical position of the background image.

So let's say you wanted to create a background like Joseph Mathew did on his new site, Local Foreigner (Figure 1).

Figure 1: The image of the woman walking is positioned in the background using CSS

The image of the woman walking is managed by CSS. If you sneak a peak at the style sheet, you'll find the background is integrated with the document using the body selector, as shown in Listing 1.

```
body {
     background-color: #F4F4F4;
     background-image: url(woman_crossing.jpg);
     background-repeat: no-repeat;
     background-position:675px 340px;
     background-attachment: scrolling;
}
```

Listing 1: Positioning a background graphic

To create this CSS using Dreamweaver MX, follow these steps:

1. From the Design Panel, choose CSS Styles.
2. Bring up the context menu by right (shift) clicking in the panel.
3. Choose New CSS Style. The New CSS Style dialog appears.
4. Under Type, select "Redefine HTML Tag"
5. In the Tag drop-down box, select "body"
6. Select Define In "New Style Sheet File"
7. Click OK.

You'll be asked to save your CSS file. Give the file a name such as style.css, click Save. The Style Dialog appears. Once it does, select Background under the Category list. Figure 2 shows the Background Style dialog filled out with the information in Listing 1. Once you've added your own selections, Click OK. Your styles will be saved to the external style sheet, and you can continue working on your current design.

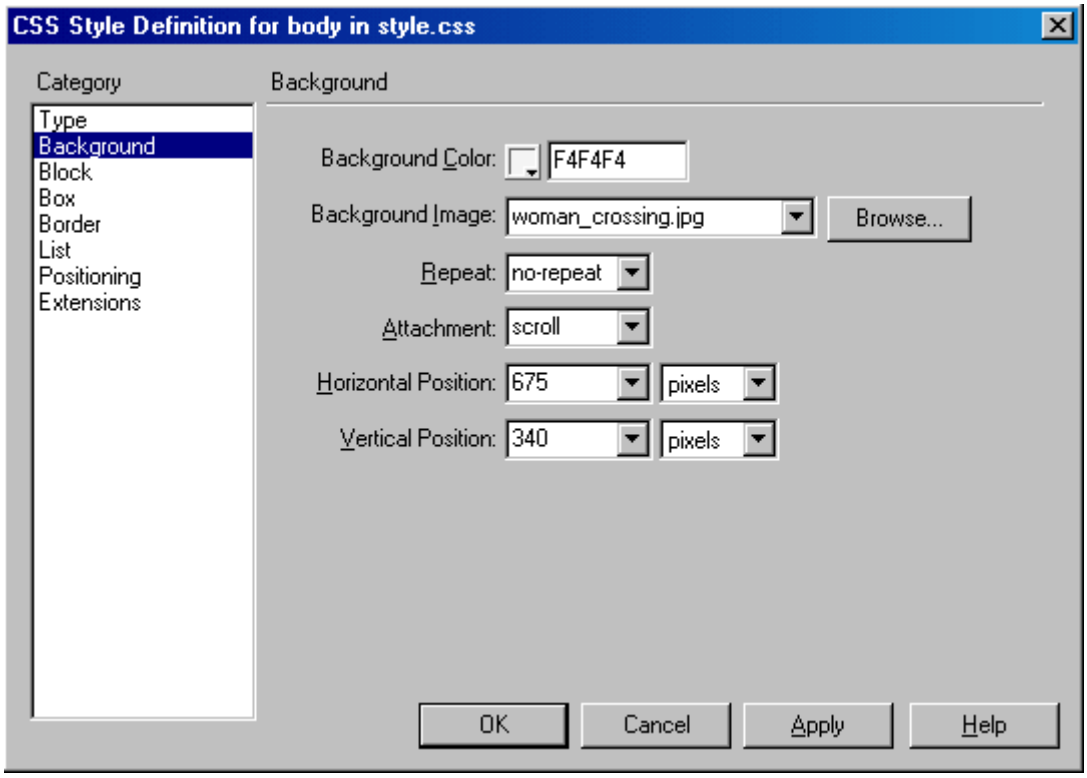


Figure 2: Using Dreamweaver MX to set background styles

Setting A Border

Border styles are used to set borders to surround any element. Each side of the rectangular border can have a unique line thickness and color. Borders can also be applied to select sides of the element, creating text surrounded on top and bottom while the sides remain open, or similar combinations. Along with thickness and color, eight border styles exist, giving the border a specific appearance.

Figure 3. Backgrounds and borders on Meyerweb.Com

The properties of the Dreamweaver Borders styles are:

- Style - Sets the style of the border. The eight border options are:
  - Dotted
  - Dashed
  - Solid
  - Double
  - Groove
  - Ridge
  - Inset
  - Outset
- Width - Sets the thickness of the border for each of the sides.
- Color - Sets the color for the border.

Remember, you can set any border for any element. So, if you want all of your level 1 headers to have a bottom, colored, dotted border only, you can set that up using the CSS dialog. To do so, bring up the CSS dialog (described in the last section) and select the Border category. Then, fill in the parameters.

To get the effect I just described, you'd fill in the dialog as I have in Figure 4.

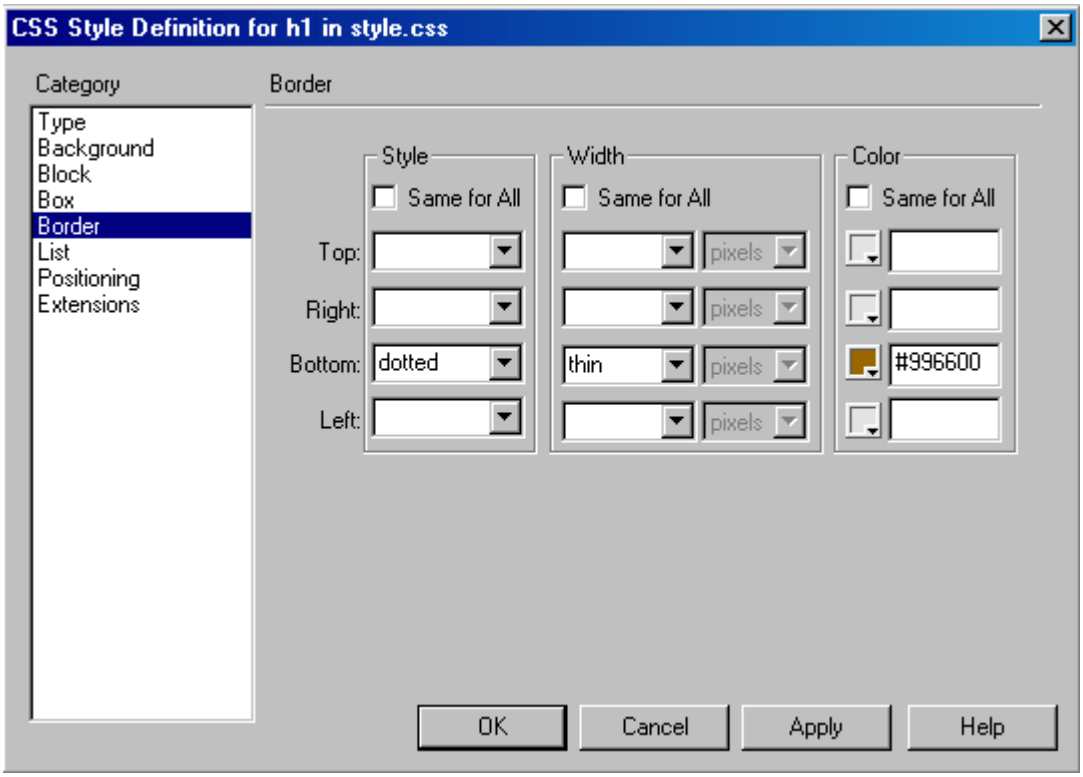


Figure 4: Setting a Bottom Border on an H1 selector in Dreamweaver MX

```
h1 {
     border-bottom-width: thin;
     border-bottom-style: dotted;
     border-bottom-color: #996600;
}
```

Listing 2 - The CSS Dreamweaver generates for the border styles

Figure 5 shows the visual results.

Figure 5 - Header style results

As you can imagine using borders in a variety of ways, applied to other elements such as anchors, paragraphs, and divisions. Using border styles are an excellent way to style great data tables, too.

Block Properties

Block styles are used to control the alignment and spacing of text blocks. D. Keith Robertson's "asterisk" Web log uses aspects of block styles (Figure 6).

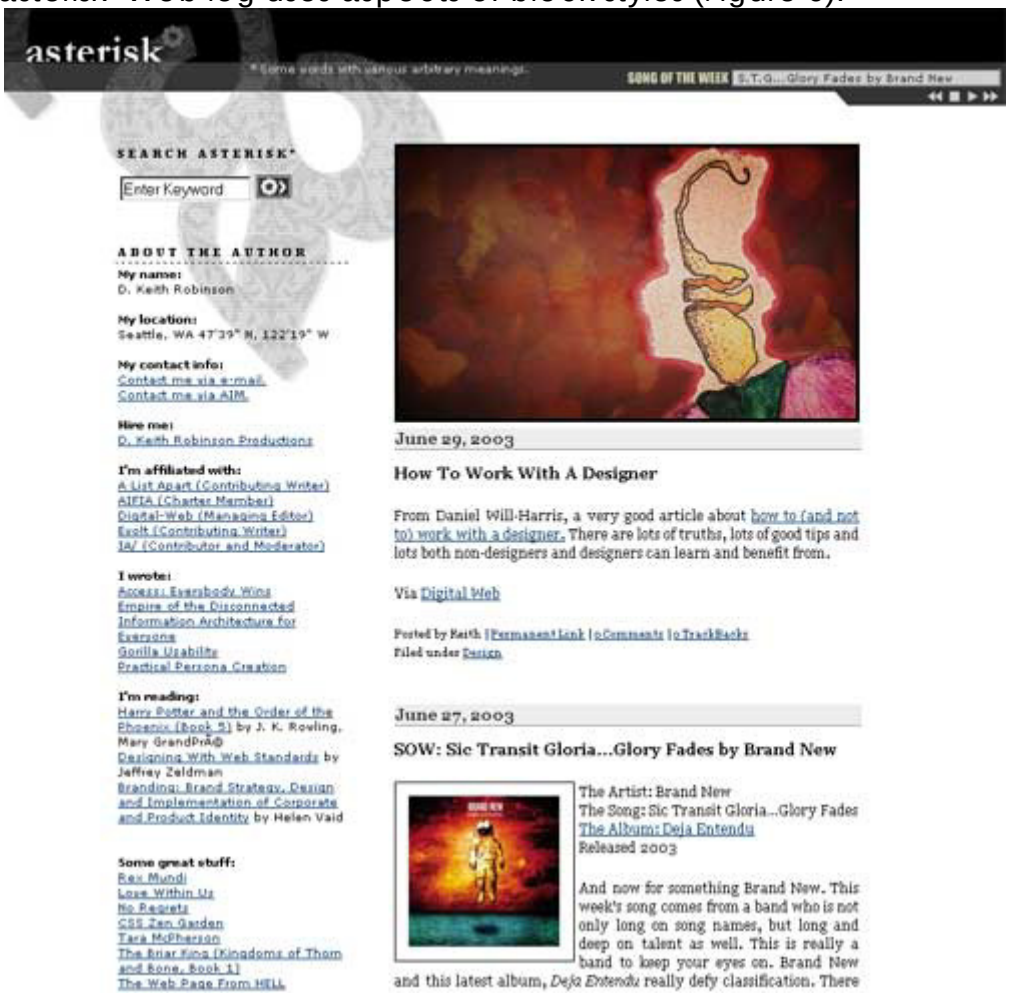


Figure 6 - Text blocks can be managed using CSS block properties

The block style properties are as follows:

- **Word Spacing** - Sets the space between words. The default unit of measure for word spacing is an em, which is the space taken up by the m character, although the unit of measure can be changed. Positive values increase the spacing between words, whereas negative values set words closer together.

- **Letter Spacing** - Sets the space between letters.

- **Vertical Alignment** - Sets the alignment of the element relative to the elements near it.

- **Text Align** - Aligns text relative to the elements surrounding it. Text can have left, right, center, or justified alignment.

- **Text Indent** - Sets the indentation of the first line of the text block by the specified value. To outdent text, use a negative value.
- **Whitespace** - Sets the control of spaces and tabs within an element. Normal causes the text block to be formatted in the same way as a default paragraph tag, where extra whitespace is ignored. The Pre value preserves whitespace. The Nowrap value causes text to extend horizontally until a
 tag is encountered, rather than wrapping to conform to the browser window.

Block properties are set in Dreamweaver MX using the Block dialog found under "Category" in the CSS dialog, as shown in Figure 7.

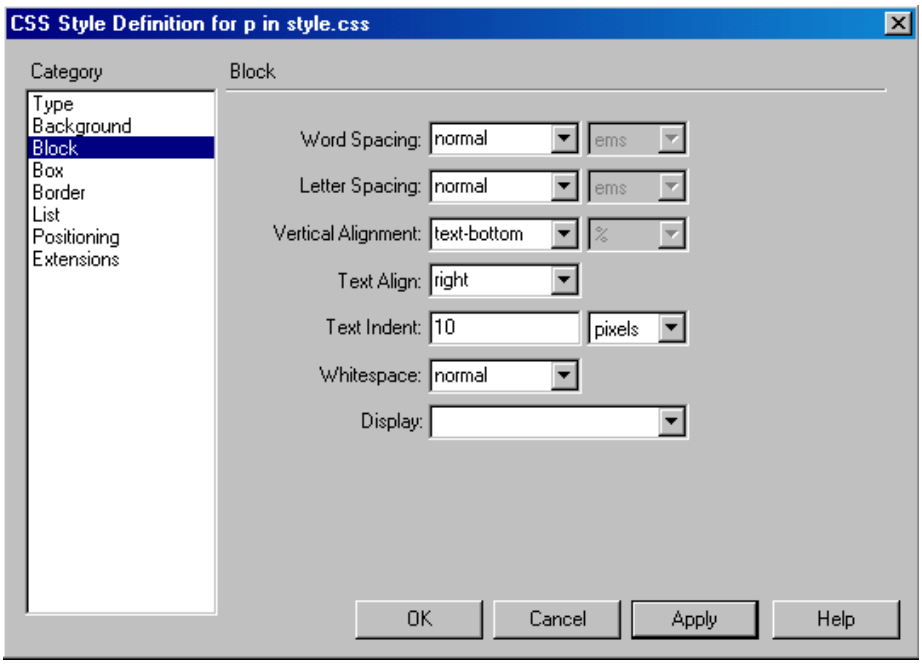


Figure 7 - Defining CSS Styles in the Dreamweaver MX Block dialog

The resulting CSS can be found in Listing 3.

```
p {
     letter-spacing: normal;
     text-align: right;
     text-indent: 10px;
     vertical-align: text-bottom;
     word-spacing: normal;
     white-space: normal;
}
```

Listing 3 - Setting block properties for a paragraph

Box Properties

Box styles are used to control the positioning and spacing of elements, much in the same way as tables.

The Box style properties are:

- **Width** - Sets the width of the element.
- **Height** - Sets the height of the element.
- **Float** - Sets the positioning of the element. Floating elements are positioned against the margin for which they are set, with the other elements of the page flowing around them.
- **Clear** - Clears the area around the box and doesn't let other elements flow around it.
- **Padding** - Sets the amount of space between the element and its border or margin.
- **Margin** - Sets the spacing between the element and other page elements.

Owen Briggs' CSS: A guide for the unglued reference page uses a combination of floating and fixed position boxes to achieve its design (Figure 8).

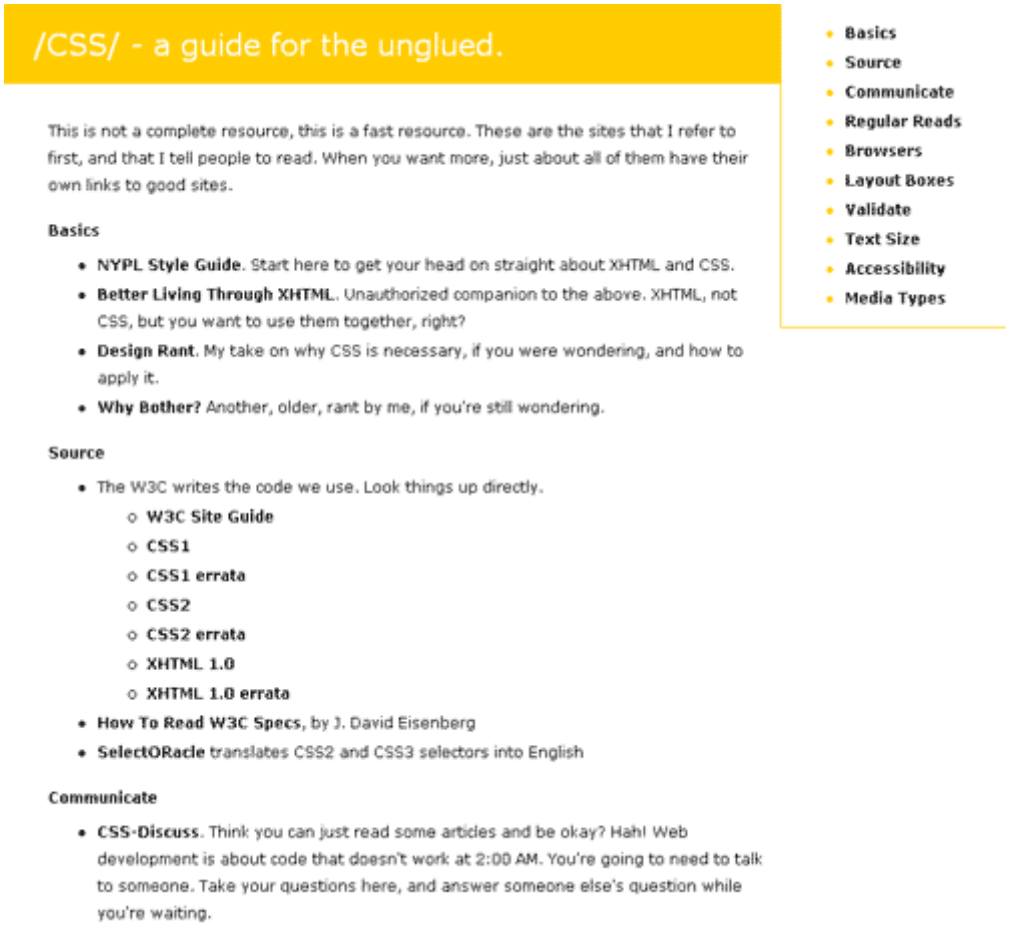


Figure 8 - This clean, crisp design uses CSS floats and positioning for its layout

5.CSS Design with Dreamweaver MX: Working with Type, Lists, Positioning and CSS Extensions

In this chapter, we don't just look at how to make CSS mimic stuff we've been doing for ages with HTML, we look at things that cannot be achieved without CSS. This chapter focuses on working with type, lists, and positioning features. I'll also show you the Extensions dialog and how you can use CSS extensions to style your pages, teaching you to use CSS as a primary means of presenting and visual enhancing your pages far beyond the limitations of HTML and XHTML.

CSS Text Styling with Dreamweaver MX

As so many designers are aware, typography is a major factor in making or breaking a design. One of the real difficulties with the Web has been the limitations on typography. And, while there were early attempts to create embedded font technologies to allow fonts to be downloaded to browsers upon reaching a page, this technology has never really emerged. As a result, Web designers interested in creating interesting typographic designs for their pages use a combination of HTML formatted text, CSS, and graphics. Flash, of course, offers designers extended opportunities to work with type.

Most readers are well aware by now that the use of font tags and similar HTML formatting for text is considered problematic for a variety of reasons. On the other hand, CSS is especially powerful in its typographic options. Firstly, there are numerous options for sizing type that don't exist in HTML or XHTML. Secondly, you can use multiple style sheets for different needs - one document can have styles that differ for screen, print, and small screens. (See the chapter "Creating A Print Stylesheet" later for the process of making most common alternate stylesheet). From a typographic standpoint, that means you can set up a style for your document that is suitable for screen while at the same time having different type styles and sizes suitable for print. Perhaps the most important aspects of CSS typography is that it is mostly part of CSS1 and is therefore widely supported by Web browsers - even Netscape 4.x versions can manage aspects of typographic style, making CSS for type a much, much better option than those available in HTML.

Figure 1 - Sardonic, an attractively designed page whose type has been styled using CSS. The line spacing, font sizing, and link effects on this page simply could not have been created with presentational HTML or XHTML.

To access the CSS type editor in Dreamweaver MX, follow these steps:

1. From the Design panel, select the CSS styles tab and click the New Style button at the bottom of the panel.
2. The New CSS Style dialog box appears. In the Define In field, choose to add the new style to an external style sheet.
3. Select a style type:
    a. Make Custom Style (Class) - Creates a class style. If you select this option, you need to name the class in the Name field above the style type selector. If you don't precede the class name with a period (.), Dreamweaver adds it for you,

as is required by the style sheet. Classes allow you to create your own styles and apply them to selectors as you wish.

b. Redefine HTML Tag - Applies a style to an HTML tag. When you select this option, you also must select a tag from the Tag field above the style type selector. These styles are automatically applied to the appropriate tags after they're defined. This option takes a standard HTML element, such as H1, and makes it a selector.

c. Use CSS Selector - Applies a style to one of the link types listed in the Selector field above the style type selector. These styles enable you to remove the underlining from links and otherwise change the appearance of the various link states. They're automatically applied after they're defined. This option allows you to use pre-defined pseudo selectors, mostly used to style links.

4. Click OK.
5. The Style Definition dialog box opens directly to the Type dialog.

Figure 2 shows the CSS Style Definition dialog with Type options available.

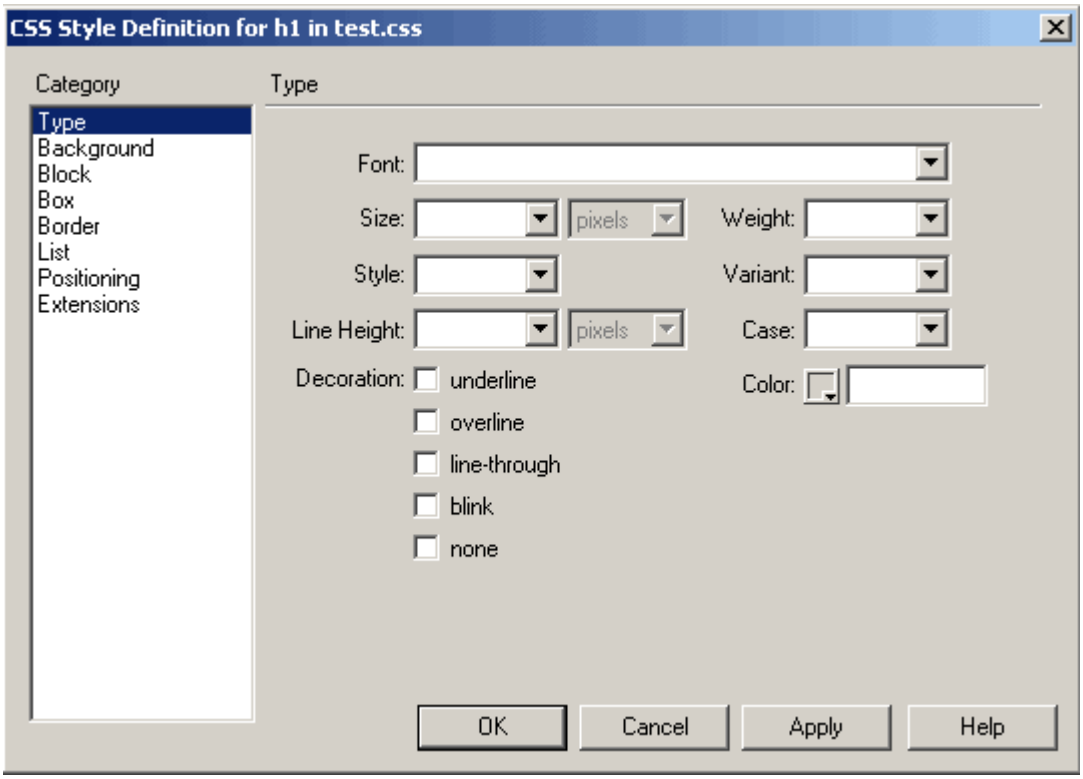


Figure 2 - The CSS Style Definition editor Type dialog.

Each of the dialog box options allows you to create CSS rules for the particular class, id, or HTML selector you wish to style.

The options are:

- **Font.** This property sets the font family, using font groups established in the Font List settings

- **Size.** This property sets the font size. If you specify a numerical value (small, larger, and so forth), you can also set the unit of measure. Choosing a percentage unit of measure increases or decreases the size of the font relative to the default. The most common unit is pixels.

- **Weight.** Weight sets the heaviness of the text boldness. Normal text has a weight of approximately 400. Bold text has a weight of 700. A weight below 400 results in lighter text.

- **Style.** Use style to set the font as normal, italic, or oblique. Normal refers to the standard font style, usually upright. Italic is a variation of that font specifically designed to have a slant. Oblique is the slanting of the normal version without any specific design changes (Figure 3). If you come from a word processing background, you're used to setting bold type in the same manner as normal and italics, but this isn't the case here. Boldness is set by weight, not style.

- **Variant.** This property allows you to set the text to display in small caps. Small caps have the same appearance as capital letters, but are the size of lowercase letters. Not all fonts will comply, and not all browsers support this feature, in which case text will show up however the author formatted the text in the first place. You can type content in all lower case, all upper case, or sentence case and then apply the small caps variant and if there is no support for the variant, it will simply appear as the author typed it in. A good tip here is to use sentence case, or ALL CAPS depending upon what you want the default results to be.

- **Line-height**. This helpful CSS property sets the leading before a line of text. Leading is the space above a letter to separate it from the text above within a paragraph.

- **Case.** This property sets the text to display in uppercase or lowercase, or with initial caps.

- **Decoration**. Using the decoration property, you can set additional properties for the display of the text, whether it should be underlined, overlined (a line appearing over the text), line-through (strikethrough), or blinking. Of course most readers know that blinking text is one of the most annoying options available to web designers, so use this with caution. Overlined can also be very confusing, especially if the text doesn't have a large line-height, as it can be confusing to the audience who may think that it as underline in the text above and click in the wrong place!

- **Color**. This property allows you to set the color for the text using the standard color picker

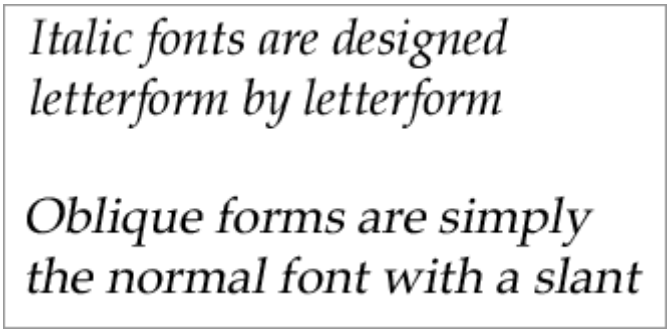


Figure 3 - The same font in italic and oblique forms.

You'll also notice that there are a variety of options for sizing. It's important to remember that there are two kinds of sizing methods: Absolute and relative. Absolute sizing is that sizing which is inflexible and does not adjust to the screen environment. Relative sizing does adjust to the screen environment. The sizing options include:

- **Pixels (px).** Measures the type in pixels, relative to the resolution of the screen, which makes it a very suitable measurement option for flexible design. Pixels are the most widely used measurement for CSS screen design because designers can size type in relation to other design features with greater control. However, pixels cannot be resized by the user, causing a significant accessibility barrier.

- **Points (pt).** This is an absolute measurement is mostly used in CSS for print and is not a suitable option for screen.

- **Inches (in)**. Also an absolute measurement, sets the type in inches, rarely used.

- **Centimeters (cm).** An absolute measurement that sets the type in centimeters, also rarely used.

- **Millimeters (cm).** Sets the type in millimeters, is absolute, and rarely used.

- **Picas (pc).** Sets the type in picas. One pica is equivalent to 12 points, and as with points, is more suitable for print.

- **Em (em).** Em is a relative measurement, equal to the value of the font-size property of the parent element. Let's say you have style the body to have a 16 pixel font. Ems will modify the size of any child of the body. Ems are commonly used in CSS for screen design, especially for sites that are meant to be accessible.

However, Ems are problematic in IE browsers if the site visitor has the browser set to font sizes lower than medium. The text becomes very small and difficult to read as a result, so many people opt for pixels instead.

- **Ex (ex).** This is "x-height" which measures a font's size from the baseline to the top its lower-case "x". Ex is relative, and can be used for screen and print but it's rarely used in screen CSS.

- **Percentage (%)**. Allows you to use percentages for type sizing. The percentage is relative to the size defined for the parent element, just as with Ems. This measurement is used by some CSS designers, especially in combination with Ems to address scalability and avoid the accessibility problems associated with pixels.

There are several terrific resources to help you make the best choices for your screen and print type measurements. Jeffrey Zeldman writes his perspective in his article "Give Me Pixels or Give me Death", http://www.alistapart.com/stories/fear4/, and Eric Meyer has an excellent article, "Going to Print" about print style sheets at A List Apart, http://www.alistapart.com/stories/goingtoprint/.

Setting List Properties

Another helpful aspect of the CSS Style Definition editor is that it allows you to easily modify the way your lists look. There are three options within the List dialog, which you can get to by going to the Category listing to the left and simply highlighting "list" in the editor.

List properties enable you to control the appearance of bullets and the wrapping of the list contents.

- **Type** sets the appearance of bullets in unordered lists from the following options: disc, circle, square, decimal, lowercase roman (such as "iv"), uppercase roman, lowercase alpha, and uppercase alpha.

- **Bullet Image** sets a custom image for unordered list bullets. This image can be any of the common formats, including animated GIF. Note that bullet images aren't supported by Netscape 4 browsers, but have been implemented in Netscape 6 and above. In Netscape 4, the image will simply not appear, but the default bullet style will.

- **Position** sets the wrapping of the list item. An outside position wraps the text to the indent of the list, while an inside indent wraps the text to the page margin.

Figure 4 shows examples of list features.

Figure 4 - Bullet images, bullet styles, and list positioning in CSS.

An important and growing area of interest with lists is using them to create navigation. The rationale behind this is that navigation is essentially a *list* of links, and that using lists is a proper structural approach to managing such lists, instead of using paragraphs, breaks, or numerous table cells.
By styling lists with CSS you can create tabbed or other styles of navigation (Figure 5) without ever touching a graphic. You can use CSS to set a list's display as inline rather than block, allowing you to use lists for horizontal navigation, as well as the familiar list style.

Figure 5 - Mark Pilgrim's site (divintomark.org) sports this tabbed navigation, which uses an unordered list and CSS to style it.

Positioning

Positioning is the heart and soul of CSS layout. For the purposes of this chapter, I'm going to stick to providing you with an explanation of the available options for positioning in Dreamweaver. In the next few chapters, you'll be working with these features a great deal, so getting a handle on the fundamentals is important.

Positioning properties form the basis of working with Dreamweaver layers. The options are:

- **Type** sets the positioning format. The formats are relative, absolute, and static (at its exact placement within the document, rather than independently of the rest of the content).

- **Visibility** sets the visibility of the layer. Layers can inherit the visibility of their parent elements, or can be set independently of the parent to be either visible or hidden.

- **Z-Index** sets the stacking order of divisions. A higher z-index means a division is closer to the top of the page in depth. A lower value means a division could be hidden under others. This technique is usually used in DHTML.

- **Overflow** sets the flow of the layer's content when it overflows the dimensions of the layer. The overflowing content can be hidden, scrolled using scroll bars that are added to the element, or auto, which automatically applies the appropriate formatting.

- **Placement** sets the actual positioning of the layer on the page.

- **Clip** sets the size of the element, which then determines where the element is clipped.

Figure 6 shows a site that uses CSS positioning - not tables - for layout.

Figure 6 - http://www.srccld.org/ uses positioned DIVs for this design.

Setting Extensions

CSS Extensions are specialty properties. The options available in Dreamweaver MX are:

- **Pagebreaks** are used to facilitate printing a web page, this style forces a page break in a long document.

- **Cursor** sets the style of cursor that appears to the user while on your page. It's probably wise not to change this property; most people are well acquainted with the 'hand' cursor above links (for example) and changing this can make your page harder to use. Of course, there are always exceptions – if you're making an experimental, artistic site, you may be eager to challenge your audience's expectations.

- **Filter** sets effects independently of Fireworks or other graphics packages. These effects control the opacity, glow, and masking features of the element (Figure 7).

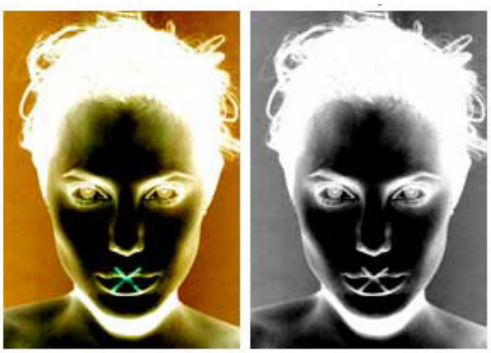


Figure 7 - A tutorial at http://echoica.net/tutorialcssfilterimages.html shows this impressive use of the invert and x-ray filter extensions as applied to images.

CSS Extensions are non-standard, only work in Internet Explorer (and then only in 'quirks' mode – "DOCTYPE Switching and MX" later for more information on DOCTYPES) and therefore their use is very limited.

6.Creating a Two-Column Layout, the Box Model Hack and Using @import to hide styles from Netscape 4

CSS layouts are giving the progressive Web designer both a tool that is changing the world of Web design, and also challenging us as we work to think in different ways about designing the Web. You've more than likely read a lot of general information about the separation of structure from presentation but you'll also know it's the concept at the heart of some of the most exciting design going.

Part of the cool factor of clean HTML and XHTML documents with CSS for layouts is that you can take the information in the markup document and lay it out as well as create styles for numerous media types including print and wireless devices. A great working example of this can be seen in Doug Bowman's redesign earlier this year for Wired News. Not only did he create a rich visual design for the Web (Figure 1), but he also created an alternate CSS for PDAs (Figure 2).

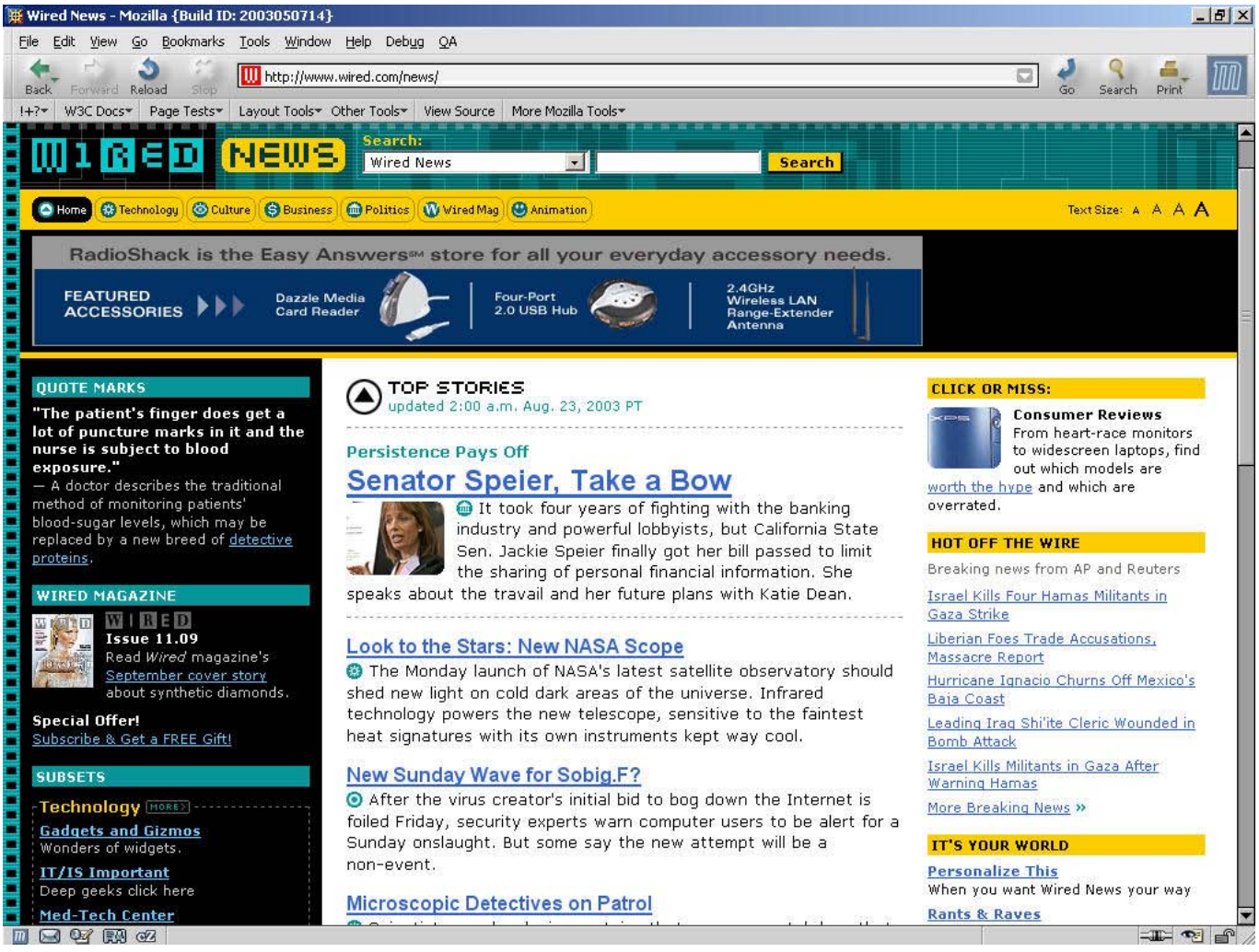


Figure 1 – Wired News Weekend Style

Figure 2 – Wired News PDA Style

Certainly, this kind of flexibility is very appealing, as are the numerous means of creating layouts with CSS. You can design some very simple layouts (as will in this and the next chapter) and then style them in unique ways to make them as individual in terms of design as you like.

Dreamweaver users have an advantage over those using other commercial visual editors in that Dreamweaver has some decent support for working with style sheets. However, Dreamweaver MX offers no pre-designed CSS layouts which you can modify. MX 2004 does, but they are not the most inspiring of designs. As a result, developing complex style sheets or modifying an existing template means balancing the tools that Dreamweaver does have and taking advantage of additional techniques to manage the rest.

For this example, I began with a two-column layout freely available from the Layout Reservoir at bluerobot.com (Figure 3). Rob Chandanais put together this site so people could begin using the templates as a great starting point for CSS layout design. The value-added beauty is that Chandanais wrote in to all his templates important workarounds useful for managing browser quirks - especially those centered around the Box Model, the visual model within browsers that CSS interacts with in terms of positioning.

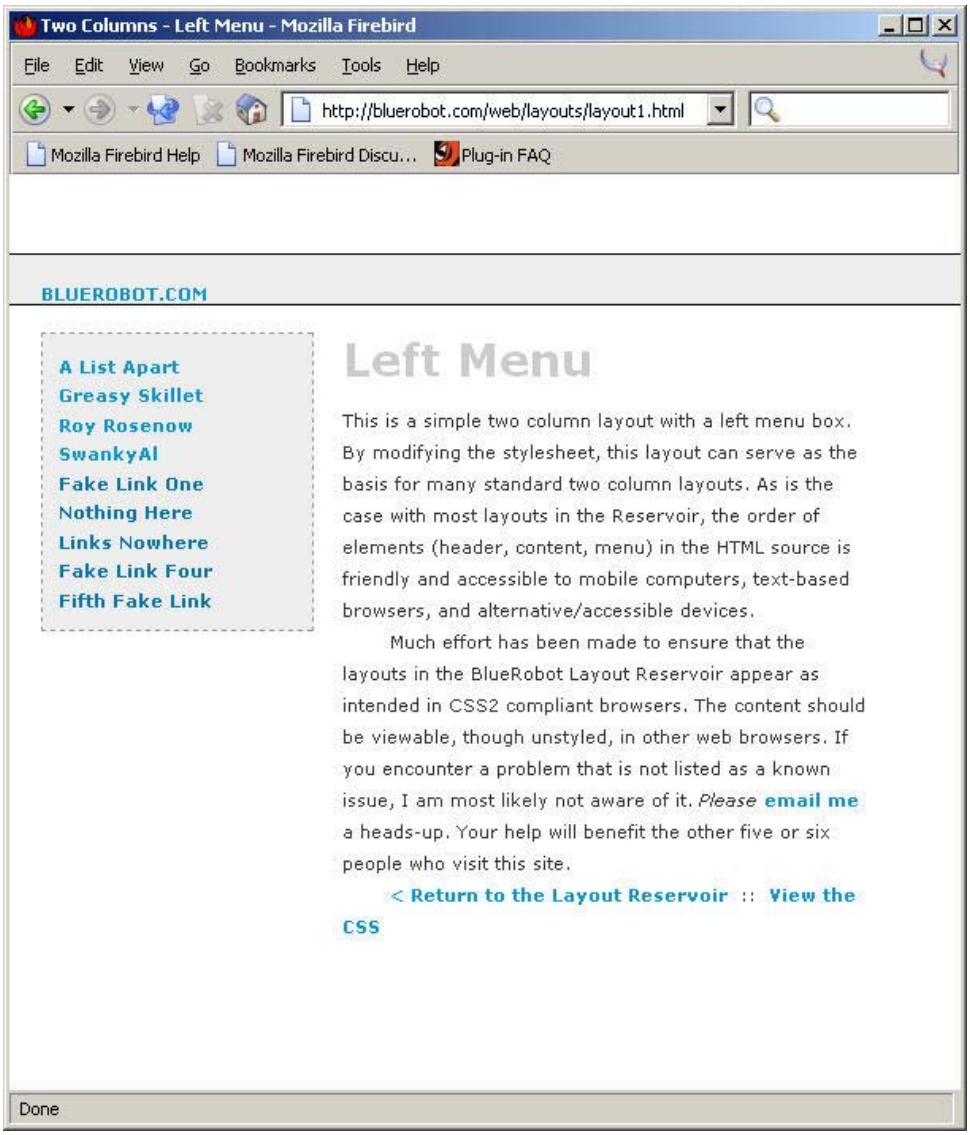


Figure 3 – The Original Layout Template

So what I did was take his basic two column, left-menu template, modify its rules somewhat, and then I separated the single template into two distinct CSS documents - one with just layout and the other with visual styles. The reason for this is to show you how to use linking and importing using the @import rule in CSS at the same time. This provides an un-styled but still readable version of your page for those users without CSS or using Netscape 4.x. This is a very commonly used technique - you may have seen it or if you work with CSS layouts a lot, you've likely used it yourself. We'll also review some other hacks and workarounds in the process of building and styling the two-column layout design.

Setting up the Markup

The first step in the process is to attach the sheet with just the styles in it. All the files are available in the zip folder of files for this chapter, next to the link where you downloaded this ebook, and I would recommend copying the directory onto your local desk and

defining your site within Dreamweaver to get all your assets localized. Make sure you place all the images in an /images subdirectory, and all the .html and .css files in the main folder. Once that's done, you're ready to create a new document and attach the style sheet.

1. From the Main menu select File > New. The New Document dialog appears.

2. Under the Basic Page Category, highlight HTML. If you prefer to work in XHTML, click the Make Document XHTML Compliant checkbox. Either language is fine for this exercise so you can use whichever you like. I'm going to use HTML 4.01.

3. Click Create. Dreamweaver will generate your page. Because we're working with CSS layouts, we will want to tap into DOCTYPE Switching wherever it's available. That means you will possibly need to modify the DOCTYPE declaration at the top of your document - this will occur while creating an HTML page. You can do this manually or use an extension (please see the chapter "DOCTYPE Switching in MX"). To see if you need to change the declaration, look to the top of your document. If your document contains the following DOCTYPE:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Then you do in fact need to modify it for best performance. To do so, simply add the URL to the HTML 4.01 Transitional DTD(marked in bold in the code snippet below) underneath this line of markup. Your results will look like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
```

4. Save your document immediately to the main level of your project folder as index.html

Linking the Default Styles Sheet

Now you'll attach the default style sheet by linking to it. Browsers that recognize the link element and have CSS support will pick up on these styles, which are not specific to layout. Rather, they are the styles of your body styles (background, color, fonts), header styles, paragraph styles and link styles, all inspired by the default style originally created by bluerobot.com, but pulled out of the main style sheet for the purposes of demonstrating the @import workaround. The code for the styles.css style sheet is below, this will serve as the template from which to base your own visual styles once everything is set up.

```
body {
     margin:0px;
     padding:0px;
     font-family:verdana, arial, helvetica, sans-serif;
     color:#333;
     background-color:white;
}

h1 {
     margin:0px 0px 15px 0px;
     padding:0px;
     font-size:28px;
     line-height:28px;
     font-weight:900;
     color:#ccc;
}

p {
     font:11px/20px verdana, arial, helvetica, sans-serif;
     margin:0px 0px 16px 0px;
     padding:0px;
}


a {
     color:#09c;
     font-size:11px;
     text-decoration:none;
     font-weight:600;
     font-family:verdana, arial, helvetica, sans-serif;
}

a:link {
     color:#09c;
}

a:visited {
     color:#07a;
}

a:hover {
     background-color:#eee;
}
```

To link the style document to your main document is easy to do, simply follow these steps:

1. With your newly created index.html document open, switch to Code view if you're not there currently.
2. Place your cursor directly above the closing </head> tag.
3. From the Design panel, select the CSS Styles tab.
4. Right-click or hold in the panel to bring up the context menu.
5. Select "Attach Style Sheet" The Link External Style Sheet dialog appears (Figure 4).

Figure 4 – The Link External Style Sheet dialog

6. Click Browse and find the style sheet named styles.css.
7. Highlight the file and click OK.
8. Under Add As, make sure Link is selected. Click Ok.

Your style sheet containing visual styles is now linked to your markup document.

Importing the Layout Styles

Using the @import rule provides a workaround that will deliver a document to Netscape 4, which doesn't have support for the @import rule but does support the link element and some styles. This way, site visitors will see your content with some style but not your nicely laid-out design. In browsers with no CSS support, this technique will also deliver the basic page, but the linked styles obviously won't be supported. Again, in this scenario, the content is available.

What's more - there are various techniques to make these pages look better, such as ordering your layout divisions in such a way as to have a logical order to them without the visual presentation, and to use skip links to get to your navigation so people can more easily deal with the page. I'll be covering all of these issues in more depth throughout the series.

To tap into the @import trick and import the layout styles for this design:

1. With the index.html document open in Code view, place your cursor below the link element and above the closing </head> tag.
2. In the CSS Styles panel, highlight the file "index.html" and bring up the context menu. Select "Attach Style Sheet."

3. Browse to the style sheet named layout.css. Select that file and click OK to return to the Link External Style Sheet dialog.
4. This time, under the Add As option, click the "Import" option.

Dreamweaver will generate the @import markup, seen here directly beneath the link to styles.css.

```
<link href="styles.css" rel="stylesheet" type="text/css">

<style type="text/css">
<!--
@import url("layout.css");
-->
</style>
```

Be sure to save your files to update the changes you've made. Now that you've got the style sheets integrated with your HTML or XHTML document, it's time to add the layout sections.

Add Divisions

As you are by now aware, positioning is managed by a combination of CSS classes or IDs attached to divisions. These divisions, created using the div element, create a box that can then be positioned within the browser viewport. In the case of the two-column layout we're using today, there are three important divisions, defined by ID, of which to be aware:

- **#header** – This division creates a box across the top of the page where you can add links, graphics, whatever you like to give the page your own style. Note that the header is not part of the actual layout per se rather it's an additional element of the page's basic design. If you don't want it, you can simply choose not to use it.

- **#content** – This is your main content division.

- **#menu** – This is your menu. It is the content and menu divisions that, when combined, create the two-columns.

Note: Remember, ID's can only be applied to one unique element per document, whereas classes can be applied to numerous elements. While you can use classes or IDs for positioning boxes, the common practice is to use IDs because it helps keep the specific ID related to the positioned box in question, rather than allowing those rules to be applied elsewhere. This will help you avoid mistakes and subsequent debugging frustration.

You can add the divisions in a variety of ways. I'll step you through it using the Tag Chooser and its related Tag Editor, which while I find somewhat cumbersome, I appreciate

because it provides a great deal of additional information about the tags you're using. So, this Dreamweaver tool can in fact be very helpful—especially if you are dedicated to learning markup and CSS—because you'll learn more about the languages and their components as you work with this editor.

1. In Code view, place your cursor directly beneath the opening <body> tag.
2. Select Insert > Tag, or hit CTL+E (Win) or CMMD+E (Mac). This brings up the Tag Chooser (Figure 5).

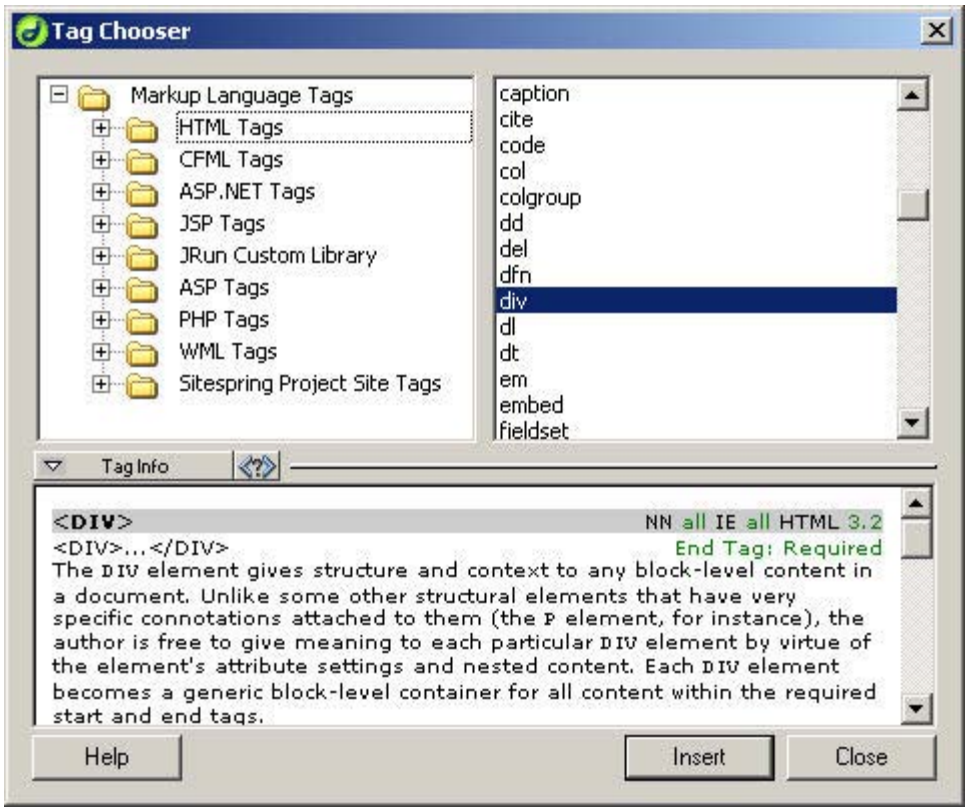


Figure 5 – The Tag Chooser

3. Highlight HTML Tags in the top left pane of the Tag Chooser. From the list that appears in the right pane, highlight div. You'll note a description of the element within the Tag Info window. Go ahead and read this, it's a decent description of the element and some of its features.
4. Click Insert. The Tag Editor appears (Figure 6). You can now see why I say the process is cumbersome - that's two dialogs to do a fairly simple task, but again, there are some advantages to using this method.

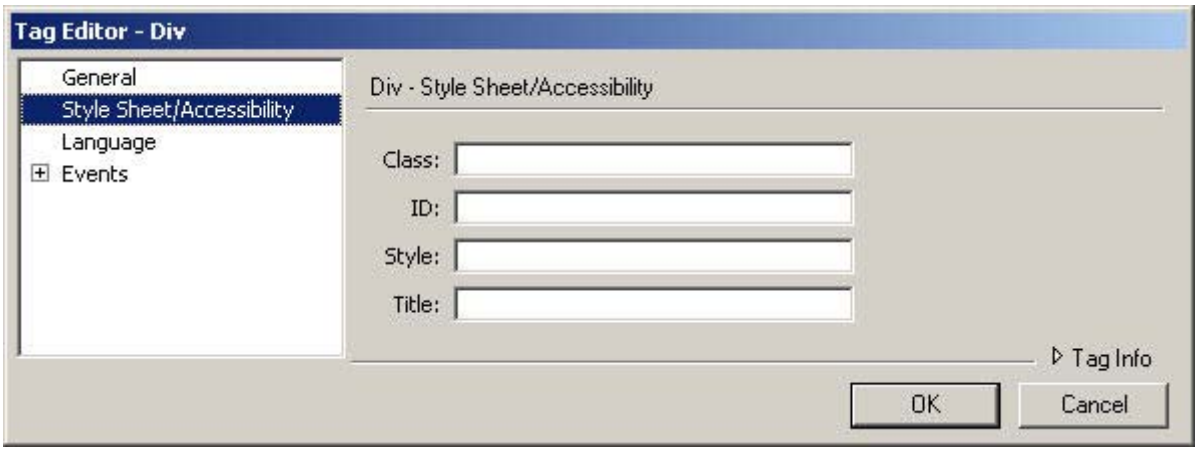


Figure 6 – The Tag Editor

5. You'll notice that in the Tag Editor for div you have several options in the left-hand pane. Highlight Style Sheet/Accessibility. The Editor will now provide you with several text fields.
6. In the ID field type "header" – all lower case and without the quotes. Click OK. The markup will be added to the document by Dreamweaver, and you'll then be returned to the Tag Chooser.
7. Switch over to Code view, and click once in the document window below the header division.
8. Switch back to the Tag Chooser, and following the same instructions as in steps 3, 4, and 5, add the next division, this time naming it "menu".
9. Repeat the process a third time, using an ID with the name of "content".

Your markup should look like this, with the one exception being if you used XHTML, then of course you'll have different syntax, but the basic structure is the same:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Two Column CSS Layout</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="styles.css" rel="stylesheet" type="text/css">

<style type="text/css">
<!--
@import url("layout.css");
-->
</style>
</head>

<body>

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

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

<div id="menu"></div>

</body>
</html>
```

Close out of the Tag Chooser.

I'd like you to switch to Split view at this point, as I want to show you a few things.

First, look at the header, which appears visually because it has been styled with a background and border. If you examine the header style, you'll see the following:

```
#header {
     margin:50px 0px 10px 0px;
     padding:17px 0px 0px 20px;
     height:33px;
     line-height:11px;
     voice-family: "\"; }\"";
     voice-family:inherit;
     height:14px;
}

body>#header {height:14px;}
```

Examining this CSS, you see that the header itself has been given margins, padding, border styles, a background color. But you'll also notice that there are *two* entries for the "height" property. What's more, there's a child selector defining the height of the header, too. How is this possible and why is it even necessary, you're asking?

Well, this is the by-now infamous Box Model Hack created by ”Tantek Çelik. The problem, as described earlier, is that some browsers, including IE 5 for Windows, incorrectly interpret the Box Model. The *proper* way in which a browser should interpret the model is to add the border and padding values *to* the width or height of a given box. So a box with a width of 300, a border of 1 pixel, and padding of 10 pixels would display properly as being 311 pixels wide.

Improper interpretations of the Box Model, such as the one that exists in IE 5 for Windows place the border and padding inside the box. In this case a box with a width of 300, a border of 1 pixel, and padding of 10 pixels would display as being 289 pixels wide because the browser is subtracting the border and padding from the total width.

The Box Model Hack "fakes out" browsers and allows you to input both values—the correct and incorrect value. By using the voice-family property (a CSS aural property), IE 5 and 5.5

for Windows will read the first height value in the header rule, and completely ignore everything after it because of a parsing bug within those browsers. Browsers that don't have this problem read the correct height of 14 pixels, add the padding and border values to that properly, and display the box normally.

Finally, the body>#header provides a means for those browsers that might not interpret the value in the voice-family property provides the correct height. So you've got your bases covered! A brilliant hack exploiting browser bugs, but giving you much better control over the way your boxes are measured.

Other items of interest to note in both the CSS and display of the divisions:

- The header division has no Draw layer icon because it is not a positioned element.
- The menu division does have a Draw layer icon because it is absolutely positioned (Draw layers in Dreamweaver are simply absolutely positioned boxes in CSS)
- The content division doesn't appear on the page because it has no border or background styling, and no content in it just yet. You should also examine the content division rules, and you'll find that in this case, there is no width and therefore no need for the Box Model Hack, which the menu and header both make use of. The no-width makes the content area fluid, so the content will flow into the available browser window space.

Go Forth and Modify

I went ahead and created three different versions of the styles. Here's what I came up with:

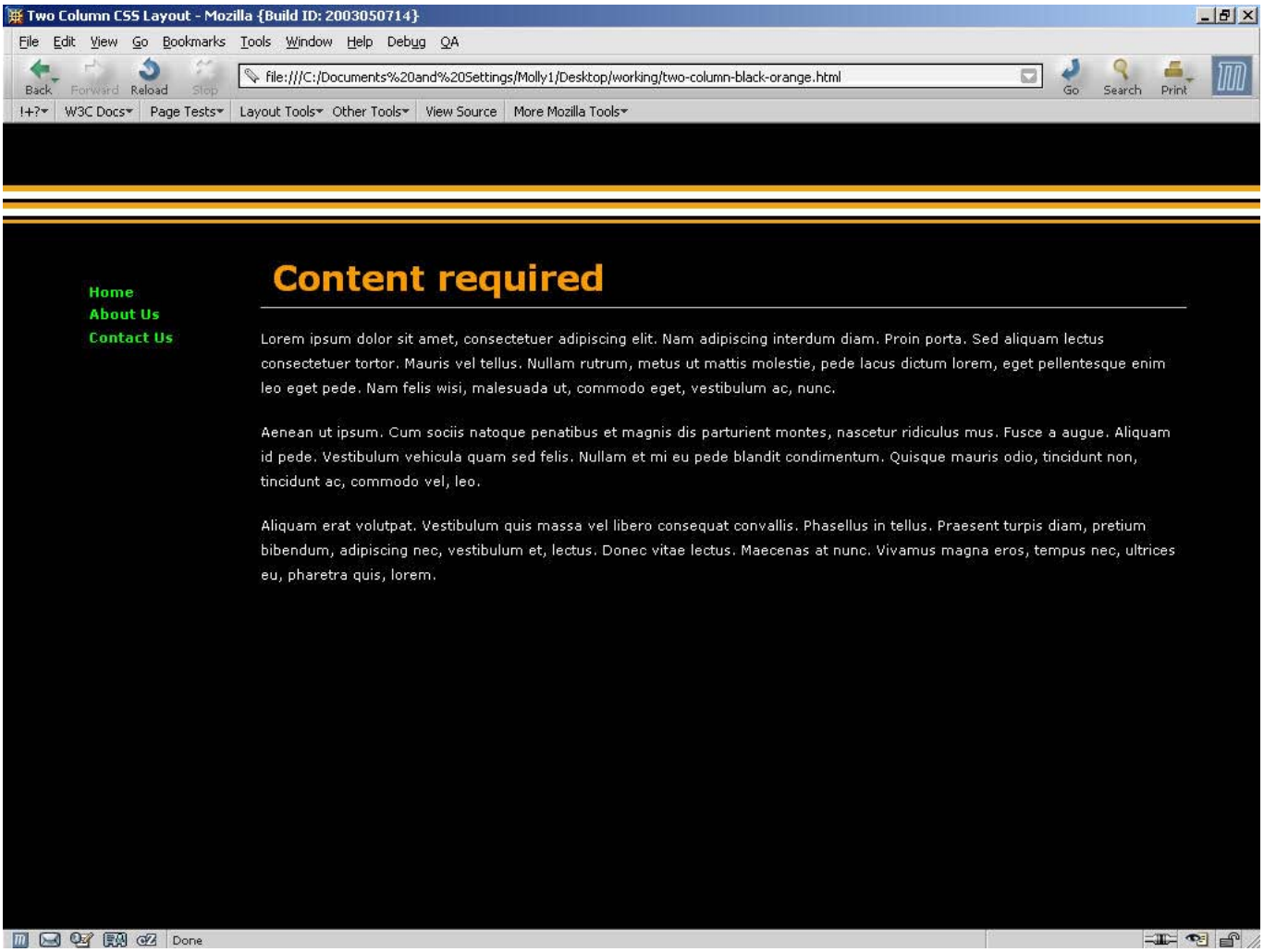


Figure 7 - Black and Orange

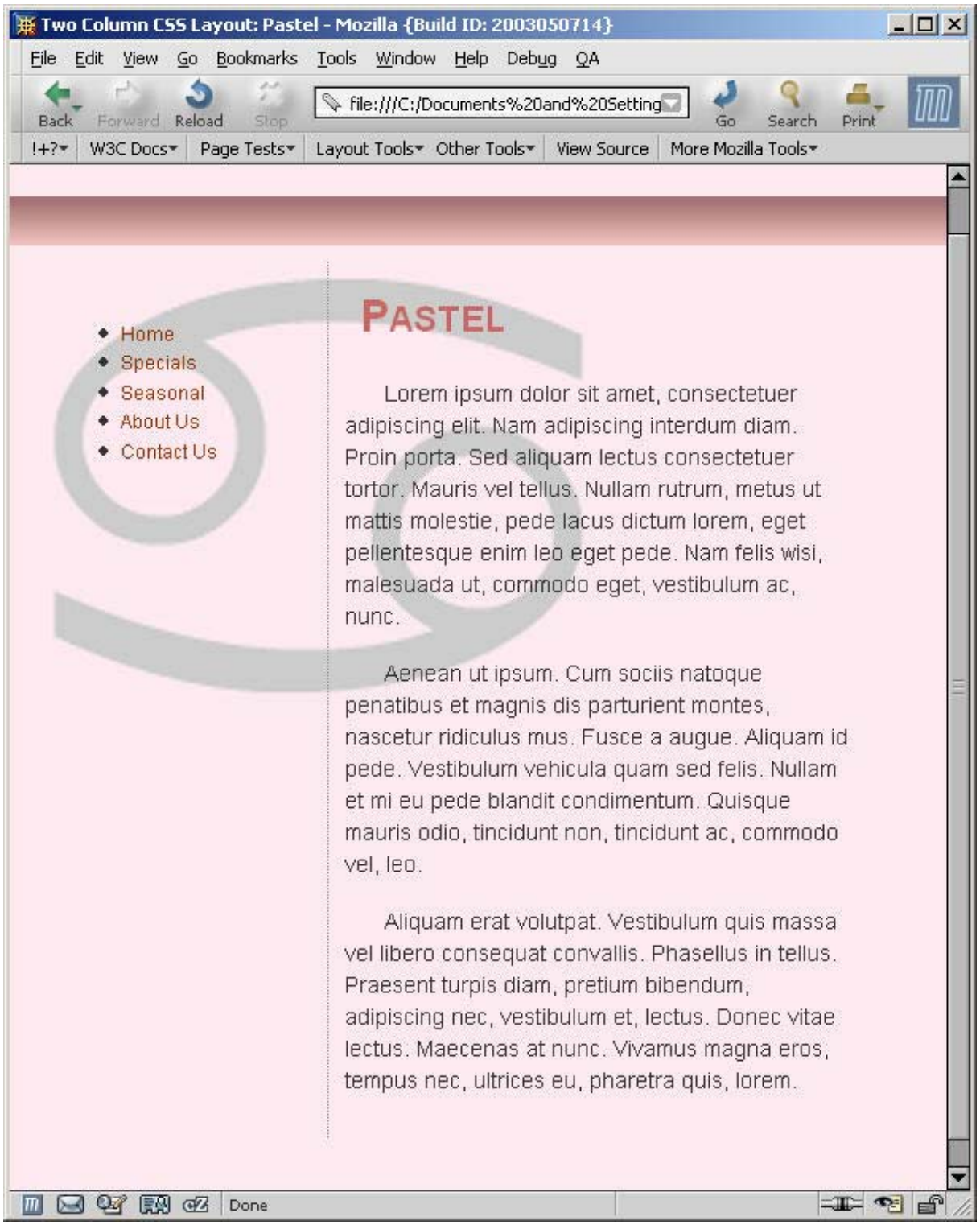


Figure 8 - Pastel

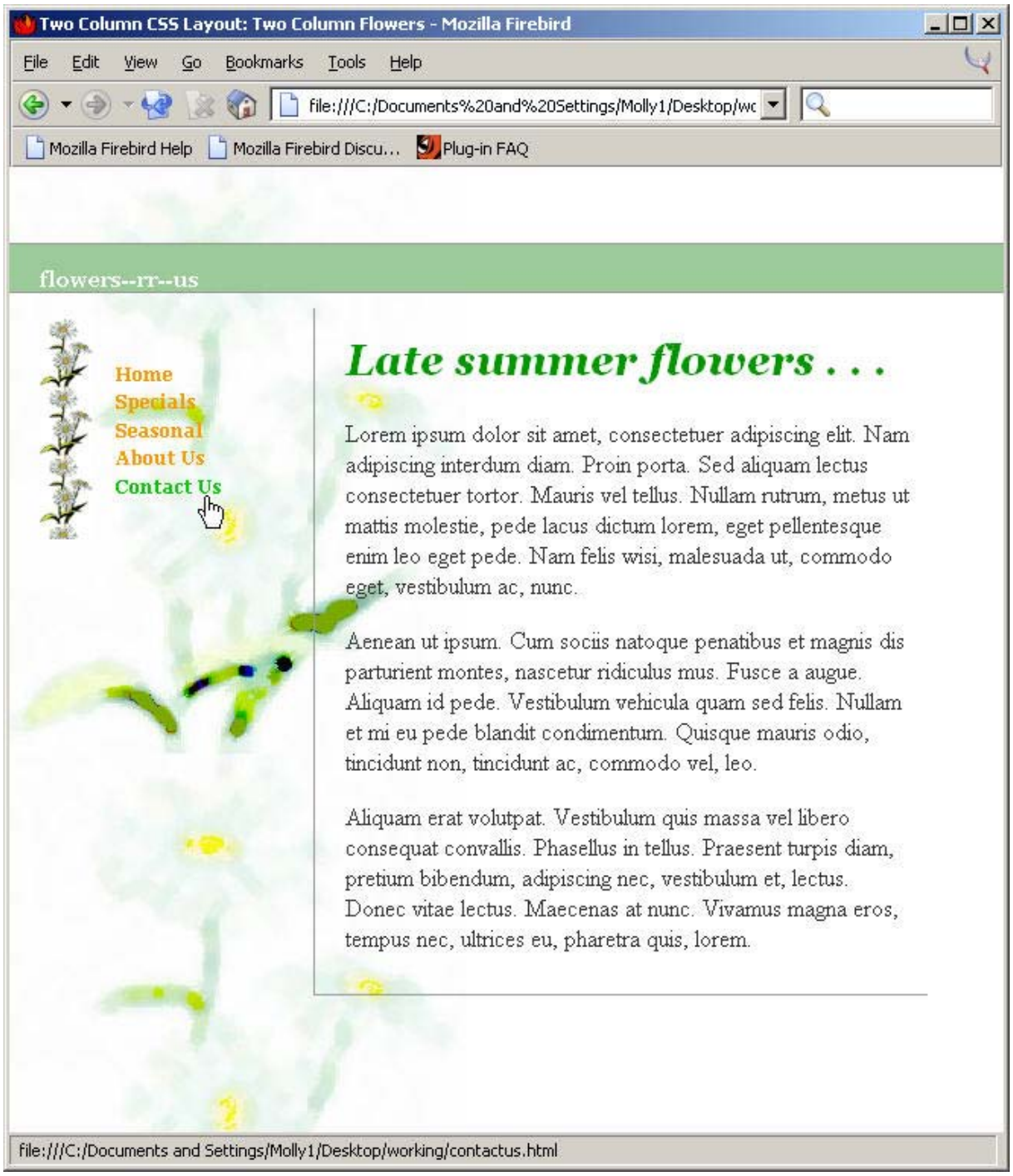


Figure 9 - Floral

Figure 10 shows the floral style in Netscape 4 - not a great layout, perhaps, but the general gist is there:

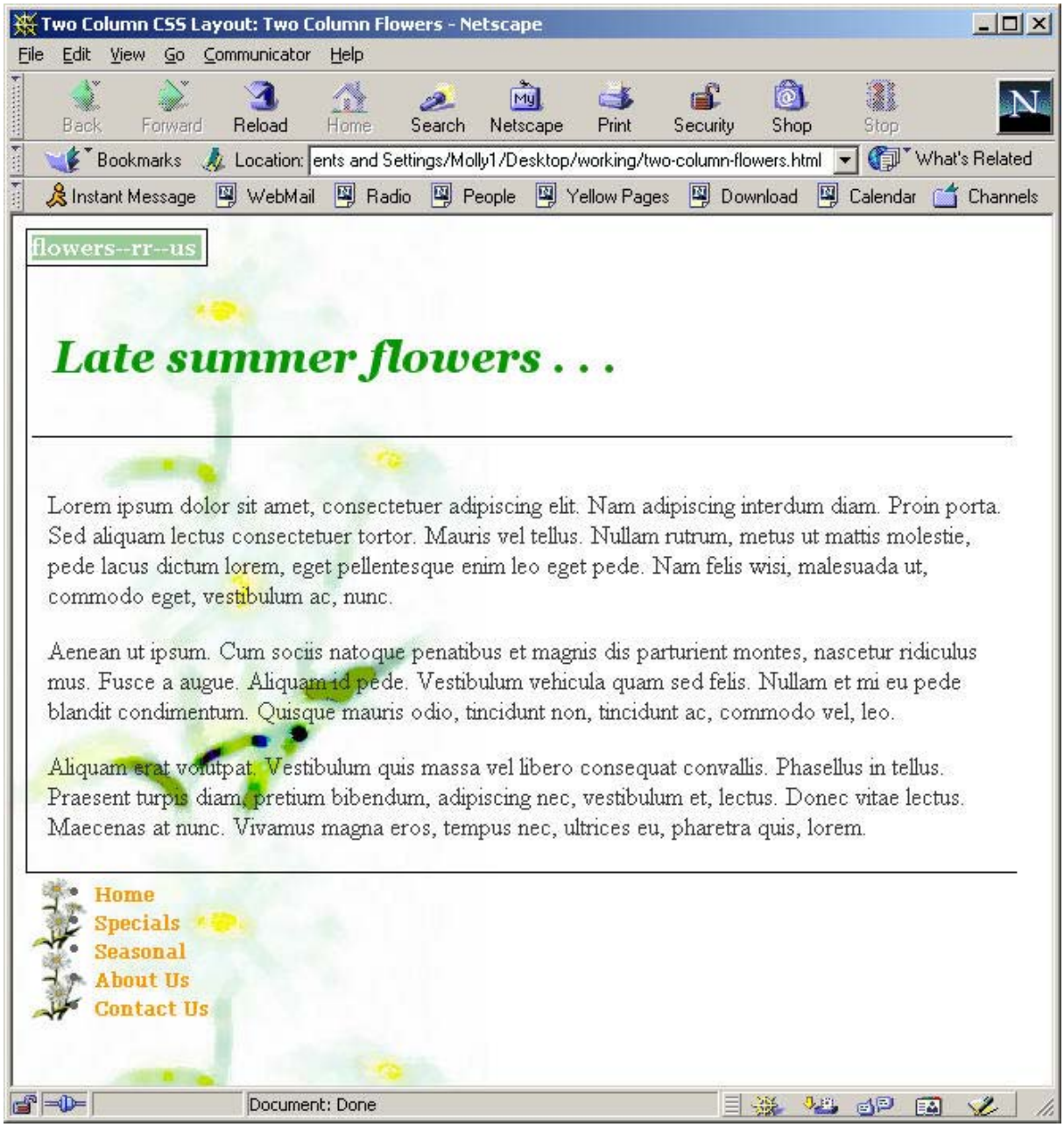


Figure 10 – Floral Style in Netscape 4

Now it's your turn! If you've been following along with this series, you should have lots of ideas of how to style these pages. I've included all the relevant markup, CSS and graphics, too, so download them and modify away!

7.Creating A Three-Column Layout

You see it everywhere—the three column layout. Whether it's achieved by table-based design or CSS, three columns seems to be a very popular means of laying out pages. Creating such a layout in CSS gives you a lot of flexibility because you really will only be creating three actual columns, relying on margins, padding, and border styles to create the white space and presentation you desire. Doing this in tables would require additional cells and spacer GIFs, making the page far less accessible and far weightier.

By relying only on CSS for layout, our actual XHTML document is very lean and mean, as well as extraordinarily bendable in the myriad ways you can re-style it quickly, easily, with no muss, fuss, or need to buy stock in the pharmaceutical company that manufactures the pain killer of your choice.

Defining the Site

The first thing you'll want to do is download the files for this chapter. Unpack them into a location on your hard drive. Then, you'll define your site so all of your assets are to hand. To define your site, follow these steps:

1. Select **Site > New**. The Site Definition Dialog appears.
2. Name the site "3 Column CSS" and click Next.
3. Select the "No, I don't want to use server technology" option, as you'll be working with CSS and XHTML only for this example. You can always change this later if you choose to develop the example beyond this chapter.
4. Select "Edit local copies on my machine, then upload to server when ready." Again, you can always modify this later, but for now we'll just work locally in the folder you created when unpacking the files. Click Next.
5. You'll be asked how you want to connect to your server. For now, select None. Click Next.
6. Click Done. Your site is defined!

Linking and Importing the Site Style

I've created both the layout and presentation style sheets, which you'll be studying and modifying as you proceed. First, let's link the presentation styles.

1. From the Site panel, double click on the **3column.html** file to open. Select Code View.
2. In the head portion of the document, below the **meta** element, click once.
3. From the Design panel, under the CSS Styles tab, click Edit Style Sheet. The Edit Style Sheet dialog appears.

4. Click Link. From the Link External Style Sheet dialog, click browse. Highlight the **styles.css** document and click OK. You'll be returned to the Link External Style Sheet dialog. Be sure that the Link option is selected below the Add As option. Click OK.
5. You'll be returned to the Edit Style Sheet dialog. Click Done.

Examine the style sheet. In it, you'll see some basic styles for color, padding, headers, paragraphs, fonts, and links. What you won't see are any positioning styles, which we'll check out next.

```
body {
     padding: 0px;
     margin: 20px;
     background-color: #99CCCC;
}

h1 {
     margin:0px 0px 15px 0px;
     padding:0px;
     font-size:28px;
     font-weight:900;
     color:#993333;
     border-bottom: #30302a 2px solid;
     font-family: Georgia, "Times New Roman", Times, serif;
     }

h2 {
     font:bold 18px/14px Georgia, "Times New Roman", Times, serif;
     margin:0px 0px 5px 0px;
     padding:0px;
     color: #CC9900;
     }

p {
     font:11px/20px verdana, arial, helvetica, sans-serif;
     margin:0px 0px 16px 0px;
     padding:0px;
     }

/* begin link styles */

a {
     color:#993333;
     font-size:11px;
     font-family:verdana, arial, helvetica, sans-serif;
     font-weight:600;
     text-decoration:none;
     }
```

```
a:link {
     color:#993333;
}

a:visited {
     color:#CC9900;
}

a:hover {
     color:#CC9900;
}

p {
     padding: 0px;
     margin: 0px 0px 16px 0px;
     background: transparent;
}

.footer {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 10px;
     text-align: center;
}
```

Go ahead and import the layout styles. If you recall from the last chapter, importing the layout styles allows us to fashion our content for those browsers with limited style support (such as Netscape 4.x) as those browsers don't understand the **@import** rule. As a result, the layout styles aren't imported, but the presentation styles, which are linked, can be interpreted.

Of course, a browser with no style support will not display either styles. So, you end up with a best-case-scenario in contemporary browsers: Full layout and style; a transitional scenario for browsers with some style; and very plain vanilla results for browsers without style at all. Either way, the site visitor gets to your content. You'll see how this works toward the end of the chapter.

To import the layout style:

1. From the Design panel, under the CSS Styles tab, click Edit Style Sheet. The Edit Style Sheet dialog appears.
2. Click Link. From the Link External Style Sheet dialog, make sure the Add As option is set to import.
3. Click Browse, and find the file **layout.css**. Highlight it, and click OK.
4. Click OK again in the Link External Style Sheet dialog, and when you're returned to the Edit Style Sheet dialog, click Done.

Your layout styles are now imported. Let's take a look.

```
#logo {
     padding: 10px;
     width: 128px;
     position: absolute;
     left: 10px;
     top: 20px;

}

#content {
     padding: 10px;
     width: auto;
     position: relative;
     margin: 0px 210px 20px 170px;
     border-left: 1px dotted #993333;
     }

#nav {
     padding: 10px;
     width: 128px;
     position: absolute;
     left: 10px;
     top: 130px;
}

/* begin navigation styles */

/* end navigation styles */

#right {
     padding: 10px;
     text-align: left;
     width: 168px;
     position: absolute;
     right: 20px;
     top: 20px;
}

#footer {
     padding: 10px;
     margin: 0px 210px 20px 170px;
     width: auto;
     min-width: 120px;
     position: relative;
     border: none;
```

```
}
```

You'll see a total of five IDs plus two comments in between which you'll be adding styles for your navigation. Note that each of the existing IDs have significance within the design we're creating, but only three of them relate to the formation of columns. The IDs are as follows:

- **#logo**. This ID defines the area for the logo. It resides within the left navigation column.
- **#content**. This is the primary content area. The style for this ID defines the center column. You'll notice its position is relative, and its width is set to auto. This allows the content to remain fluid within the division.
- **#nav**. The nav ID contains the styles for the left column, into which we'll place our navigation for this design. You'll notice that its position is absolute, so it never moves.
- **#right**. This ID defines the styles for the right column. Its position is also absolute.
- **#footer**. This defines styles for a footer, which will be displayed below the content. While we could have added our footer content to the content area itself, footer information typically contains copyright, registration, privacy and other information that is updated regularly. Separating it into its own division makes it easier to perform site-wide search and replace functions to update that content with ease.

Both the **styles.css** and **layout.css** form the basis of the layout and presentational elements for the exercise. You can modify them at any time to suit your own design ideas.

Defining Your Divisions

With the majority of the styles in place, it's time to work in the **3column.html** file again. Go ahead and open it in Code View.

Within the body element, you'll manually add the divisions, along with their appropriate IDs. You're going to position the logo first, the content second, the navigation area third and the footer last.

```
<div id="logo">

</div>

<div id="content">

</div>
```

```
<div id="nav">

</div>

<div id="right">

</div>

<div id="footer">

</div>
```

Save your file to update the changes.
Why are we ordering our logo and content prior to the menu? This is because we want the logo and content to be the first thing that gets seen in those particular browsers that do not handle positioning. Browsers with positioning rely on the CSS, not the order of the markup, to position your div elements. So for those good CSS browsers, the order of the divisions simply doesn't matter, whereas for those without it, it does.

Adding Content

We'll work in Code View at first to flesh out our areas.

1. In the content area, type the words **stylin' a three column layout.**
2. Highlight the text, and from the Property Inspector, select Heading 1 from the Format drop-down menu.
3. Add text content in paragraph format below. I've included some dummy text in the dummytext.html document. You can simply open that document and copy and paste the dummy text into the content area. Of course you may add your own text content, too.
4. Above the final paragraph, add the words "and here's more!"
5. Highlight the words, and from the Property Inspector, select Heading 2 from the Format drop-down menu.

Figure 1 shows your progress in Design View.

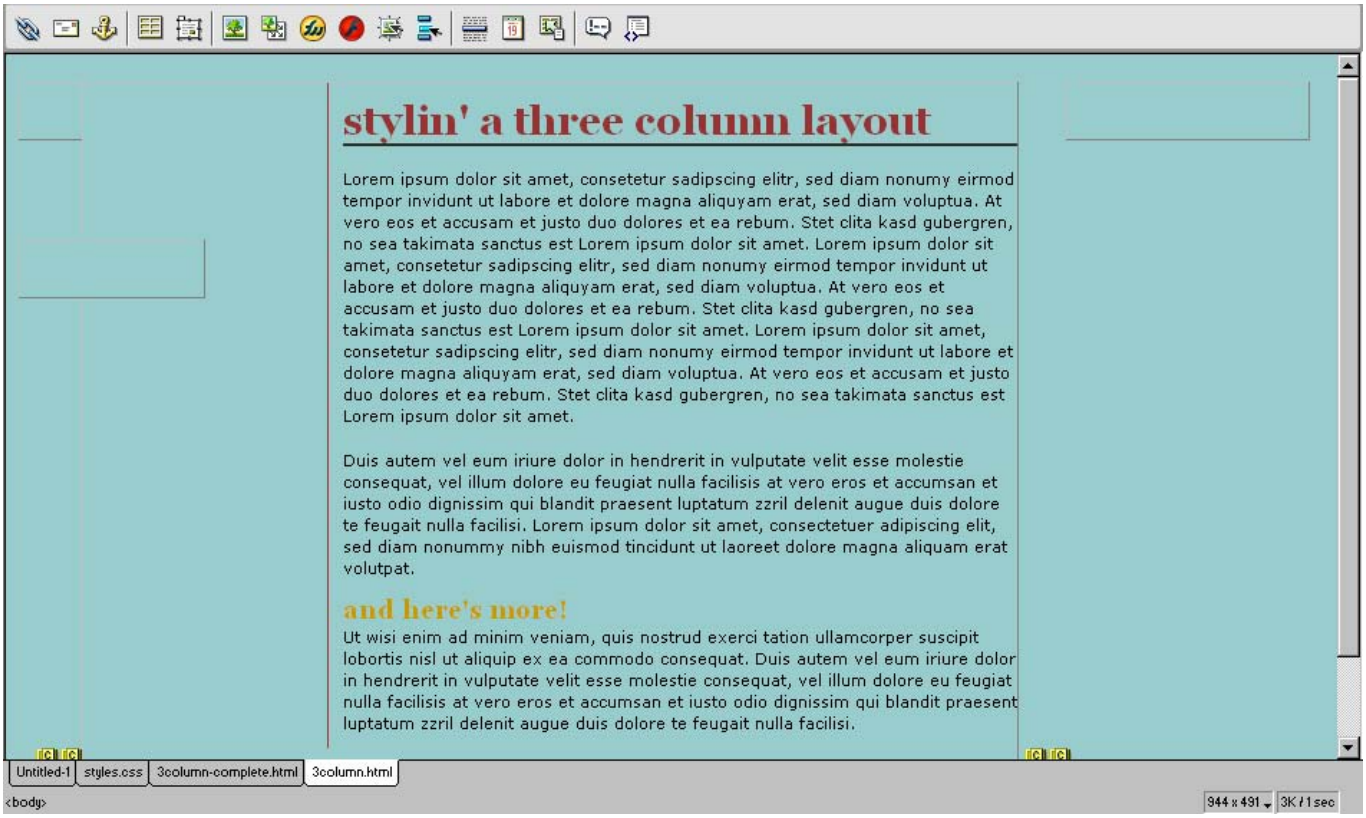


Figure 1

Adding the Logo

Adding the logo is very simple.

1. In the Site window, expand the images folder.
2. With the page open in Design View, click the file **logo.gif** and drag it into the logo division. Be sure to drop it into the specific #logo section!
3. With the logo image selected, open the Property Inspector, and add descriptive text into the Alt textbox.
4. Save your changes.

Adding Navigation

There are lots of ways to style navigation these days, but as you read in chapter 2, a very popular way of styling navigation is to use lists and modify the lists with CSS. We'll tap into the power of lists once again, this time adding some styles that will create a simple but attractive menu.

Begin by adding the following unordered list with links to the **#nav** division in the **3column.html** document.

```
<ul id="navlist">
<li><a href="link1.html" title="Go to Link One">Link One</a></li>
<li><a href="link2.html" title="Go to Link Two">Link Two</a></li>
<li><a href="link3.html" title="Go to Link Three">Link Three</a></li>
<li><a href="link4.html" title="Go to Link Four">Link Four</a></li>
<li><a href="link5.html" title="Go to Link Five">Link Five</a></li>
<li><a href="link6.html" title="Go to Link Six">Link Six</a></li>
<li><a href="link7.html" title="Go to Link Seven">Link Seven</a></li>
<li><a href="link8.html" title="Go to Link Eight">Link Eight</a></li>
<li><a href="link9.html" title="Go to Link Nine">Link Nine</a></li>
</ul>
```

You'll notice that I've added the ID **navlist** to the opening **<ul>** tag. This list can't be styled until we add the navigation styles to the layout style sheet. Go ahead and open **layout.css** (if it isn't already opened) and scroll to the line found between the navigation comments pointed out earlier. Add the following CSS:

```
/* begin navigation styles */

#nav ul   {
     margin-left: 0;
     padding-left: 0;
     list-style-type: none;

}

#navlist {
     padding-left: 0;
     margin-left: 0;
     border-bottom: 1px solid #993333;
     width: 128px;
}

#navlist li {
     margin: 0;
     padding: 0.25em;
     border-top: 1px solid #993333;
}

#navlist li a {
     text-decoration: none;
}

/*end navigation styles */
```

Let's examine these IDs, because some of the selectors here might be unfamiliar to you.

- **#nav ul**. This is a special type of selector known as a descendant selector. Specifically, it's a child selector, and means that *any* child **ul** of the **#nav** division will be affected by the style declarations within this rule.

- **#navlist**. This is the ID that applies style to the entire list.

- **#navlist li**. Here you see another child selector, this time any child **li** of **#navlist** will be affected by the style defined here.

- **#navlist li a**. Another descendent selector in operation, but specifically choosing any anchor element within a **li** within the ID **navlist** to be styled by the style information here.

You can now save your file and view **3column.html** to see how the navigation looks (Figure 2).

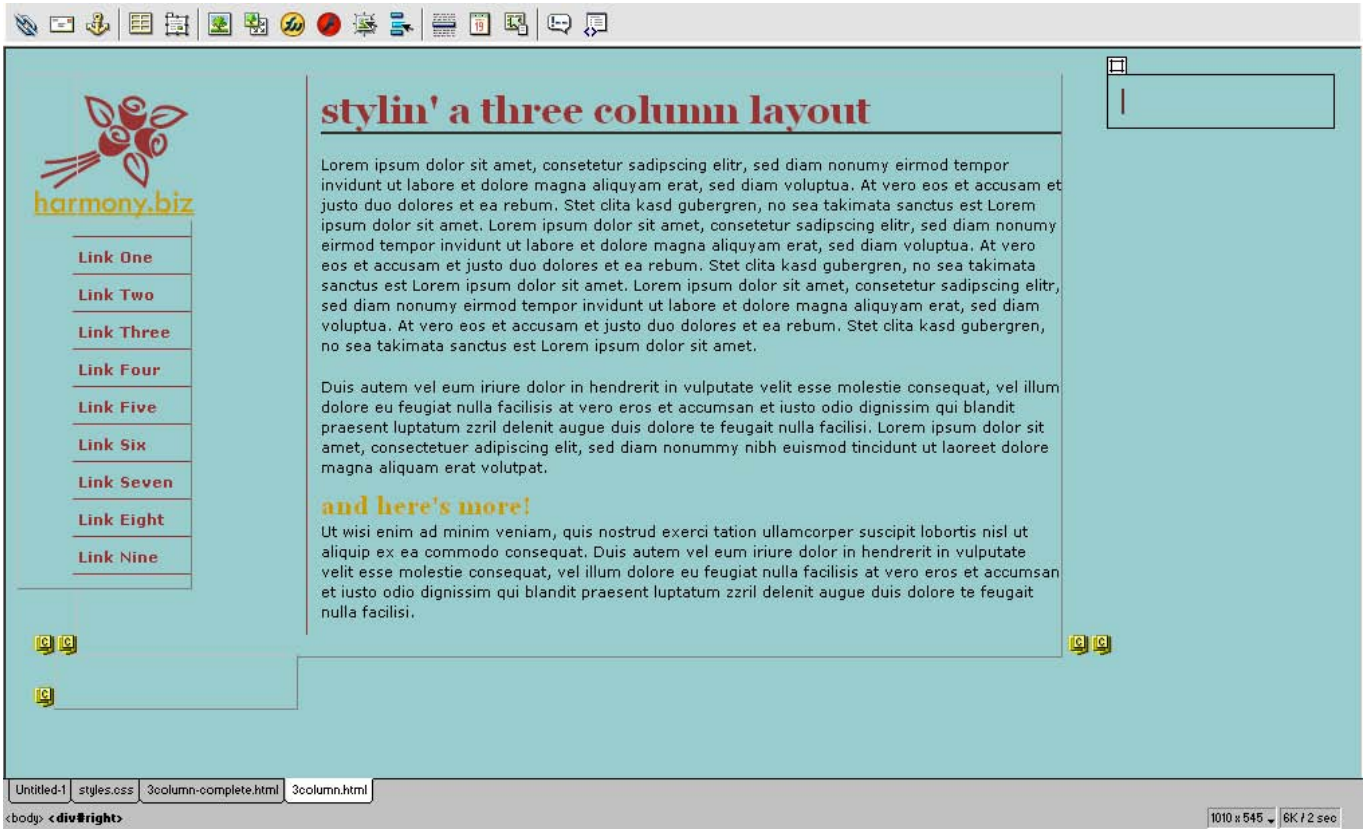


Figure 2

Adding Content to the Right Column

You'll want to add content to the right column. This might be text, images, or a combination of both. I've inserted a series of photographs, and included them in the site for you to use as mockups. To add the photos, have the 3column.html file open. Select Design view. From the Site panel, expand the images folder for your site.

1. Click on red-flowers.jpg and drag and drop it into the right column.
2. Select Enter (Return) to drop yourself down to the next line. Now, drag and drop yellow-flower.jpg into the column.
3. Select Enter again, and drag and drop pink-flowers.jpg into the column.
4. Using the Property Inspector, add Alt text for each image.

Your resulting markup for this is as follows:

```
<div id="right">
  <p><img src="images/red-flowers.jpg" width="150" height="113"
alt="photo of red flowers" /></p>
  <p><img src="images/yellow-flower.jpg" width="150" height="113"
alt="photo of yellow flower" /></p>
  <p><img src="images/pink-flowers.jpg" width="150" height="113"
alt="photo of pink flowers" /></p>
</div>
```

Easy enough! Save the file and view it in an external browser once to see the changes :

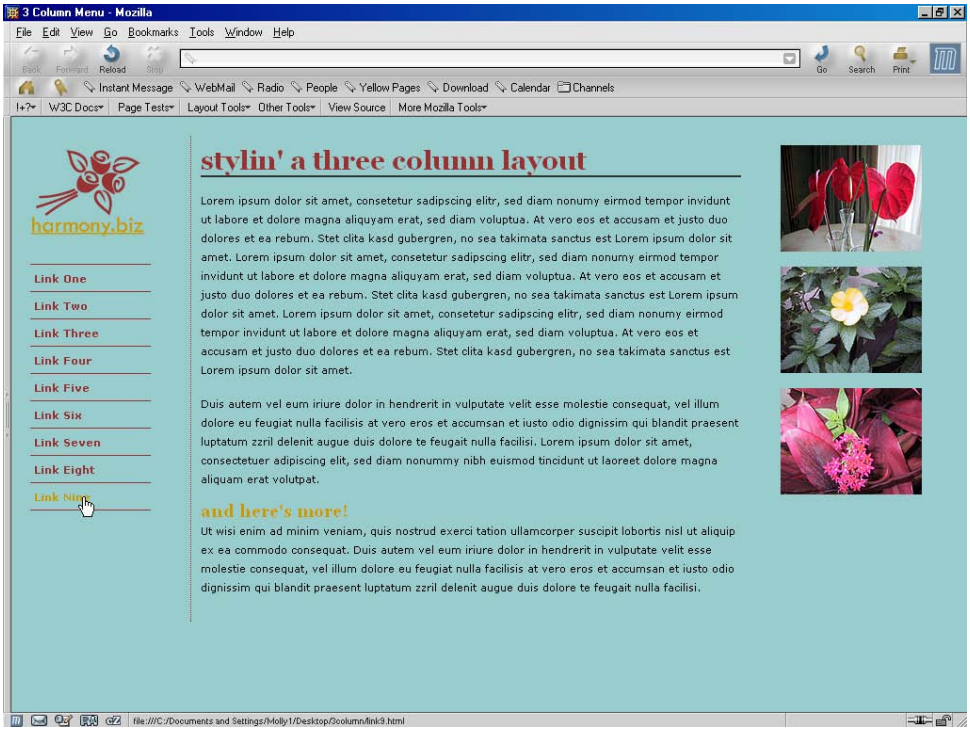


Figure 3

Adding the Footer

This one is even easier! Just type in the text you wish to appear in the footer and format it as a paragraph. Add the class "footer" to achieve the text effects.

Your resulting markup for this division is as follows (with any modifications you yourself made to the text):

```
<div id="footer">
  <p class="footer">&copy; 2003 Molly E. Holzschlag </p>
</div>
```

Cleaning Up and Testing Your Documents

In this section, you'll put your documents through a few housekeeping tasks. First, clean up your XHTML. This helps get rid of unnecessary white space and corrects errors. To do this:

1. Have your 3column.html file saved and at the ready.
2. Select Commands > Clean Up XHTML. The Clean Up HTML / XHTML dialog appears. You can leave everything at the default if you've followed the directions here closely.
3. Click OK. Dreamweaver MX will clean up your XHTML and provide you with a Clean Up Summary dialog. Click OK.
4. Save the file.

Now you're ready to validate the document. To do so:

1. Select File > Check Page. A submenu appears.
2. Select Validate Markup. The validator will now run.
3. When the validator is finished, it will open the Results Inspector. If all went well, you'll have no errors. If you made any mistakes, the validator will tell you the line and a description of the problem.
4. Repair any errors and check your page until no errors are found.

If you'd like to check your CSS, you can do so by using the CSS validator at the W3C. To do so, follow these steps:

1. Point your browser to http://jigsaw.w3.org/css-validator/.
2. Click the link "by upload". The validator will bring you to a new page.
3. Click the Browse button, and browse to layout.css.
4. Choose Submit. If you've followed the directions in this chapter without making any errors, the file will validate. If you do find errors, go ahead and repair them, and run the validator again.

5. Repeat the process for styles.css. Again, if you've followed the directions without inadvertently introducing errors, the styles will be valid.

You're ready to rock! Figure 4 shows the final results in a Mozilla Firebird, a compliant CSS Browser.

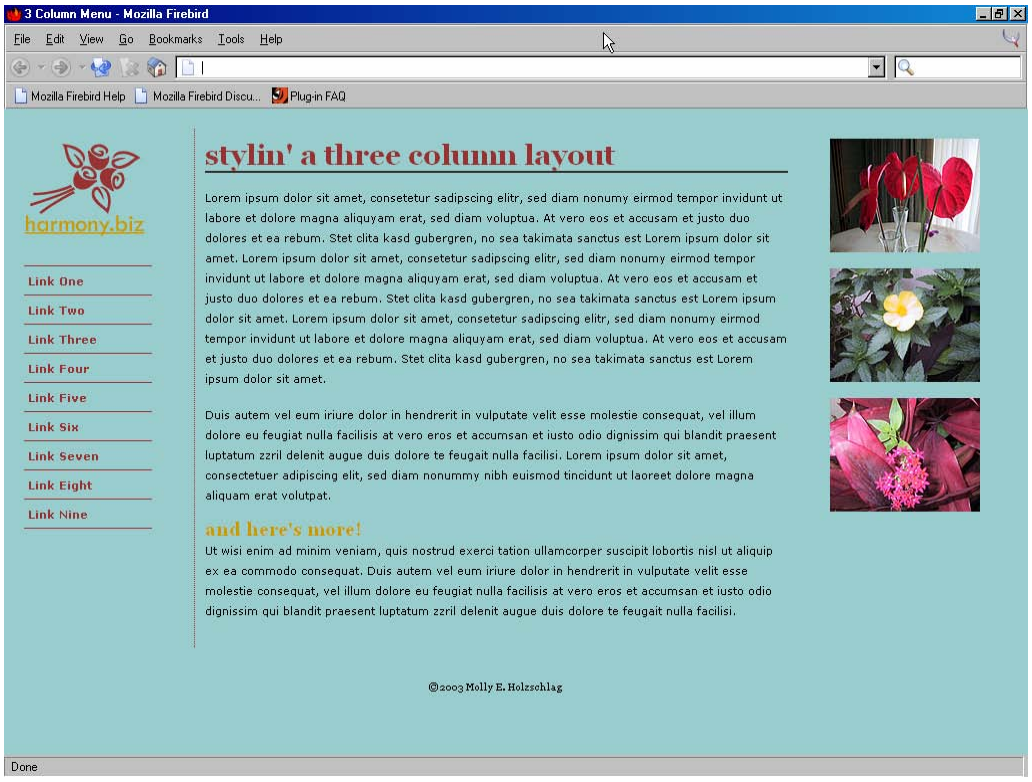


Figure 4

Figure 5 shows the results in the semi-compliant browser, Netscape 4.x. You'll notice the logo and content up top, the navigation below. While not as pretty, it's all still readable (the right column images appear below the footer).

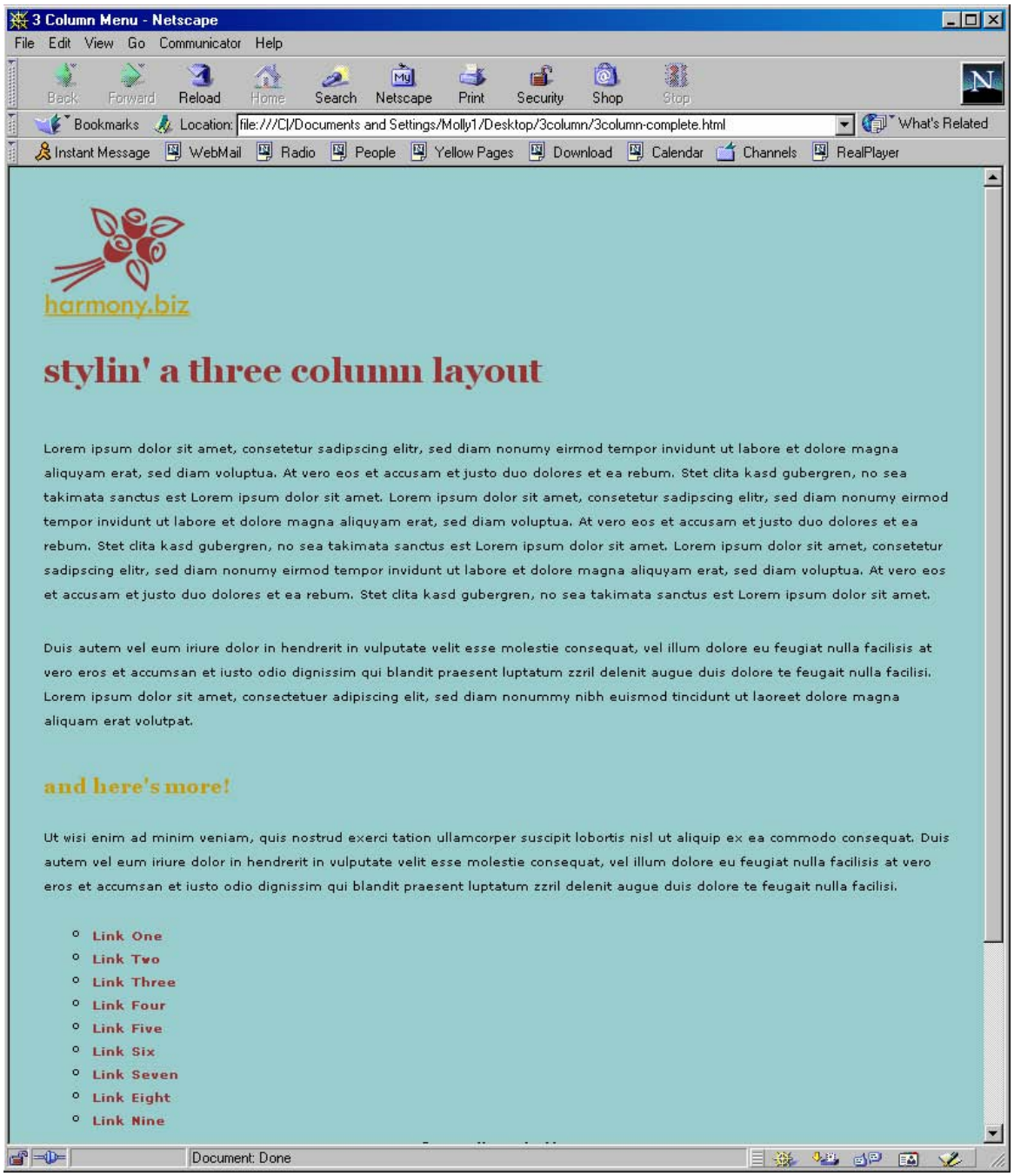
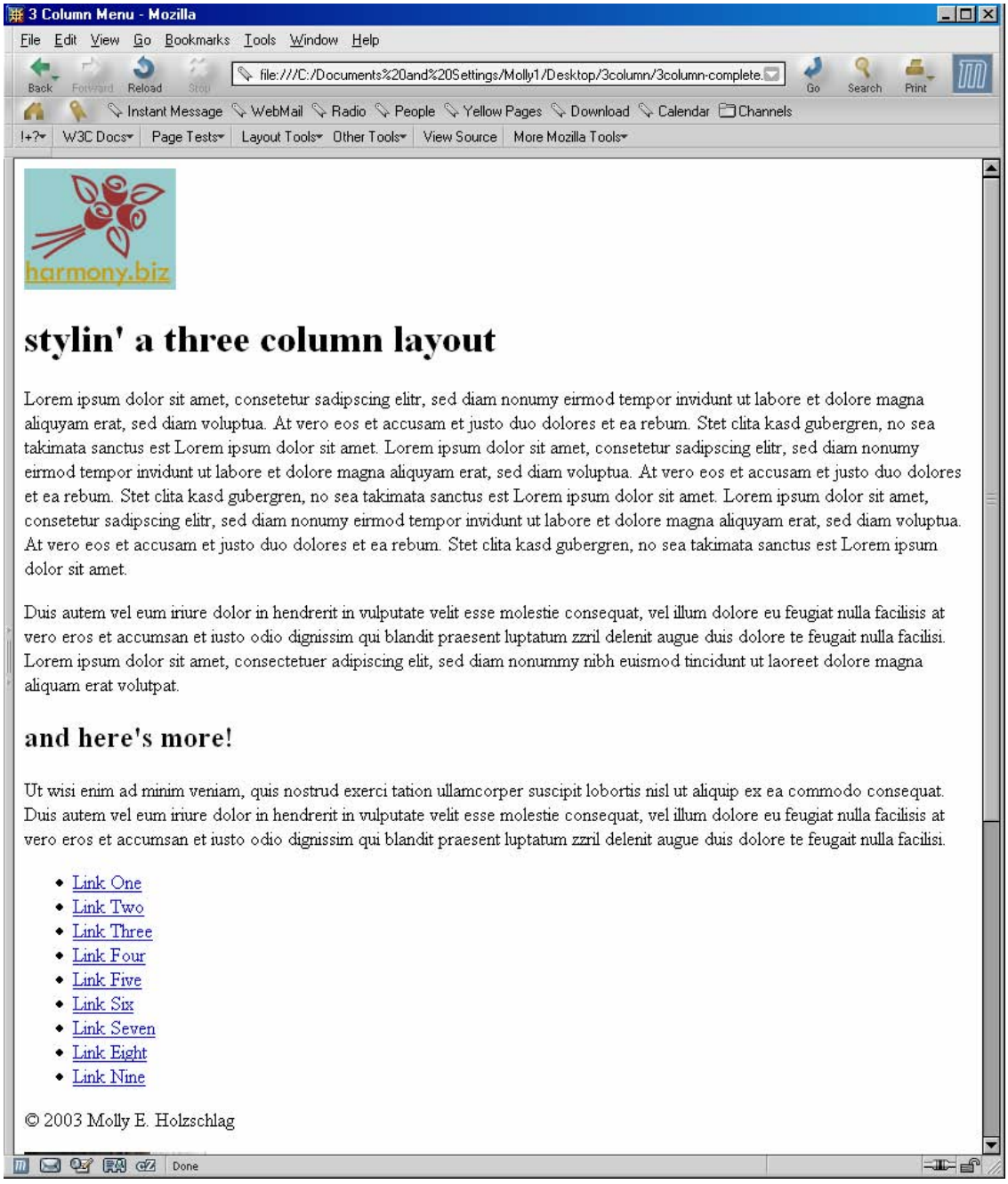


Figure 5

And finally, Figure 6 shows the design in Mozilla with all styles turned off. Still readable, still navigable, just not very stylish!



Modify Away

Of course, your own design skills probably far outweigh my simple approach, so go forth and use the techniques here to create a 3 column design of your very own.

8.Creating a Weblog Layout, and using a horizontal navigation list

Web logs or "blogs" have taken the Web by storm. Blogs can be profound, entertaining, or poignant. They also tend to be where some of the most creative visual design occurs. This is especially true in terms of the use of CSS. Maybe it's the independent nature of Web logs that have allowed them to come to the forefront of CSS design, because most avid bloggers are bound to be using up-to-date browsers with ample CSS support. Or maybe it's because many popular Web log programs such as Blogger and Movable Type tend to offer CSS options as part of their templates, enabling individuals less familiar with Web design to use CSS without a lot of fuss n' muss. Whatever the reasons, it's certainly an intriguing phenomenon to find so much progressive design among blogs, and as an extension it's something many designers want to tap into for personal as well as professional reasons.

For this chapter, I'll design a new Web log for myself using Dreamweaver MX. My intention is to show you a very simple CSS layout that features some important techniques, including:

- Using Dreamweaver MX to create a majority of the source CSS
- Using Dreamweaver to lay out the page
- Using hand-authoring techniques to improve performance
- Creating multiple style sheets to show instant different looks for the same layout
- Setting up **horizontal navigation** using an unordered list (if you've not learned this trick, this chapter is worth the read just for learning this technique!)

While the design is very simple--using only a single column--it's a great starting place for those interested in working more with CSS to *design* sites and not just code them.

About the Design

I had several important goals when setting out to create this design. As mentioned, I wanted to keep the example very simple. That means few graphics and very clean, well structured mark-up. So, when I set out to create the design, I first made some simple sketches to help guide the way. Then, I created a mockup in Photoshop and played with ideas until I came up with a design that met my criteria.

I put together a collage that was relevant to my Web log topics--something that showed me along with scenes from my travels and personal life. I knew I wanted this to run horizontal along the top of the page, offering visual interest. I wanted navigation so people could get to other areas of my site, of course, and I needed an area for content.

Figure 1 shows my initial mockup in Photoshop.

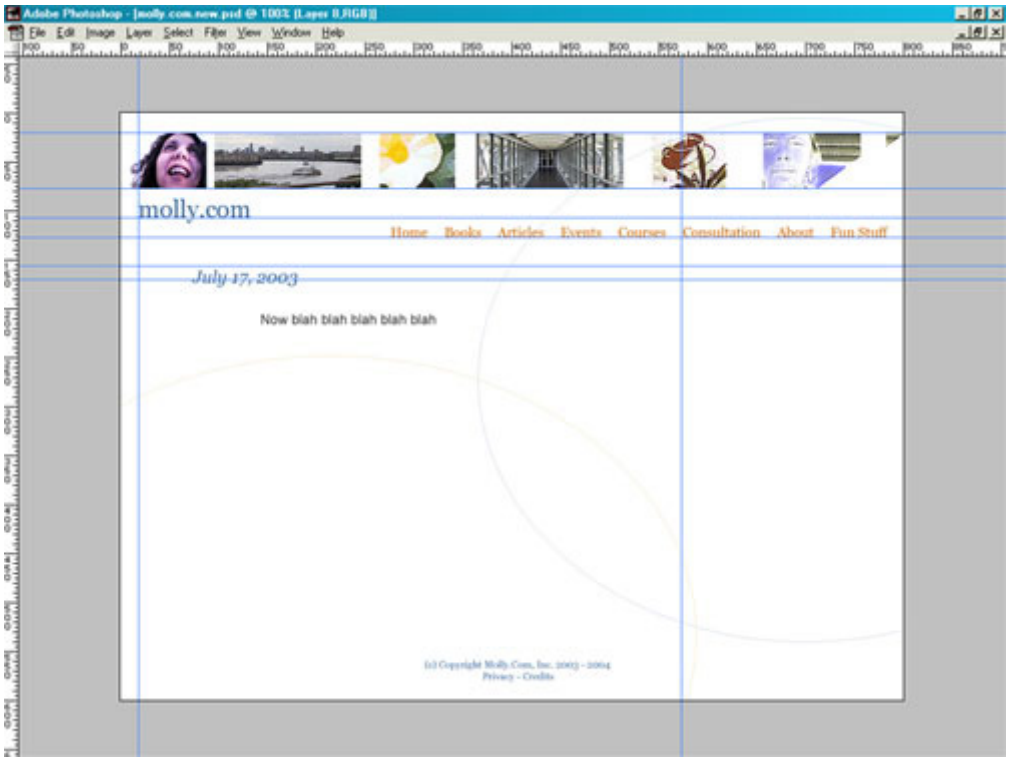


Figure 1: Mocking up the page in Photoshop

Happy with the clean and bright results, I took down some notes on what I wanted for my color palette so I could refer to it easily, and then generated two graphics from the Photoshop file: The collage strip along the top, and the background design with intersecting arcs. I saved the collage strip as a JPG to maintain the photographic quality and color. Its total weight is 21KB. Then, I saved the background image as a GIF since it has flat colors--and very few colors at that. The total weight of the background is 5KB. So that's 25KB for dependencies, which works out very well.

Once satisfied with the images, I fired up Dreamweaver MX and began to work on the markup.

Creating the Markup

Knowing that I wasn't going to use any presentational HTML or tables for layout, that my media needs were very light, and that I wanted to tap into the power of DOCTYPE Switching to better manage my CSS layouts, I chose to use XHTML Strict. So the first thing I did in Dreamweaver MX was to select **File > New**, and checked the "Make Document XHTML Compliant" checkbox. Then I saved my file immediately and used one of my favorite extensions, Jerry Baker's "Insert HTML DOCTYPES", to replace the default Dreamweaver MX XHTML Transitional DOCTYPE with the XHTML 1.0 Strict DOCTYPE.

Listing 1 shows the default Dreamweaver MX XHTML DOCTYPE:

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

Listing 1: XHTML Transitional DOCTYPE

While it would have been perfectly acceptable to use an XHTML 1.0 Transitional DOCTYPE, I knew I wouldn't want to be using any presentational elements or attributes, so using the XHTML 1.0 Strict DOCTYPE ensures that I would catch any deviations from that goal when I validated the document. Listing 2 shows the Strict DOCTYPE the extension replaced the default with:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Listing 2: XHTML Strict DOCTYPE

Note: For more information on DOCTYPEs, seethe chapter "DOCTYPE Switching with MX".

With my document type and basic structure in place, I then moved on to using Dreamweaver MX's **Draw Layers** feature to create the positioning of the elements. First, I drew a layer along the top to contain the collage image. I named this layer "collage." Then, I continued using the Draw Layer feature to position the text header, the navigation, and the content areas, naming them "header", "nav", and "content" respectively.

Tip: Name your Dreamweaver MX layers as you create them, using names that are relevant and understandable to others who might be working on the site as well as yourself.

Positioning the Visual Elements with CSS

It's great using Dreamweaver for doing positioned layouts because it takes a lot of the guesswork out of doing the CSS. Because Dreamweaver MX Draw Layers puts the CSS that positions the boxes inline, my tendency is to use Dreamweaver for the positioning, then go in and pull out the inline markup and put it in an external style sheet. This way, I not only have all the positioning for a site in a single file, making it easier to manage, but I save a little on the document weight as well as keeping the XHTML itself very clean.

Here's the inline positioning that Dreamweaver created:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>Molly.Com, Inc. Welcome</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>
<link href="molly.com.new.css" rel="stylesheet" type="text/css" />
</head>

<body>
<div id="collage" style="position:absolute; left:0px; top:4px;
width:800px; height:58px"><img src="top-main.jpg" width="800" height="58"
alt=”photographic collage” /></div>

<div id="header" style="position:absolute; left:0px; top:76px;
width:800px; height:35px">

<h1>molly.com</h1>

</div>

<div id="nav" style="position:absolute; left:204px; top:93px;
width:596px; height:32px; z-index:1">

<ul>
<li>Home</li>
<li>Books</li>
<li>Articles</li>
<li>Events</li>
<li>Courses</li>
<li>Consultation</li>
<li>About Molly</li>
<li>Fun Stuff</li>
</ul>

</div>

<div id="content" style="position:absolute; left:0px; top:154px;
width:600px; height:400px; z-index:2">

<p>&copy;Copyright 2003 - 2004 Molly.Com, Inc.<br />Privacy - Credits</p>
```

```
</div>

</body>
</html>
```

Listing 3: Dreamweaver's inline positioning.

As you can see, Dreamweaver MX has generated the inline CSS for my positioned areas. As mentioned, I prefer to put this all into an external style sheet, where I can also make modifications to the CSS that Dreamweaver has authored so as to gain further control and get rid of any rules that are unnecessary.

First, I moved the collage style information from the main document into my external style sheet. In my CSS document I created an ID selector, **#collage**, and then input the rules:

```
#collage {
     position:absolute;
     left:0px;
     top:4px;
     width:800px;
     height:58px;
}
```

Listing 4: The CSS positioning for the collage image.

In this instance, I didn't add or take away any CSS, nor did I for the ID selector I created, **#header**:

```
#header {
     position:absolute;
     left:0px;
     top:76px;
     width:800px;
     height:35px;
}
```

Listing 5: Positioning the text header.

I did make one modification to the CSS for the selector, **#nav**. I removed the Z-index that Dreamweaver had input. A z-index is unnecessary in this particular design so, to save a few bytes of the download time, I took it out:

```
#nav {
     position:absolute;
     left:204px;
     top:93px;
     width:596px;
     height:32px;
}
```

Listing 6: Positioning the navigation

Next, I created a **#content** ID selector and moved the CSS from the main document to the external style sheet. However, here I did make significant changes:

```
#content {
     position: absolute;
     width: 600px;
     padding: 10px;
     margin-top: 100px;
     margin-bottom: 20px;
     margin-right: 200px;
     margin-left: 200px;
}
```

Listing 7: Here, I've made significant changes from the Dreamweaver CSS

The only thing I kept from the Dreamweaver MX styles for the content area is the position and width. I removed everything else and added margins that were more effective for the design, as well as adding a 10 pixel padding around the text itself. While I could have made these modifications once the Draw Layer was created using Dreamweaver's CSS editing tools, I chose simply to use Code View and type the declarations directly into the CSS document. It just was faster in this case.

Styling the Page

Now that everything is quite literally in position, I turned toward adding links and styling the document: adding fonts, header styles, colors, link colors, background, and creating two classes, **.nav**, which we'll get to in the next section, and the **.footer** class. I did all of this using Dreamweaver's CSS tools, after which time I went into the external style sheet and moved things around into a more logical order.

Everyone works differently of course, but I tend to be fairly strong-minded about keeping things logical. So, I'll make sure I start my CSS documents with the body selector, and then any HTML selector I use from there down, then links, then ID selectors, then classes. I then add comments into the CSS, denoting each section. The results of my CSS document (except the .nav class as I will discuss that in the following section), are seen in Listing 8.

```
/* molly.com.new.css - global styles */
body {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 14px;
     color: #666666;
     background-attachment: fixed;
     background-color: #FFFFFF;
     background-image: url(molly.new.bak.gif);
     background-repeat: no-repeat;
```

```
     background-position: left top;
}

h1 {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 22px;
     color: #336699;
     text-indent: 10px;

}

h2 {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 18px;
     font-style: italic;
     color: #336699;
     text-indent: 35px;

}

/* begin link styles */

a:link {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 14px;
     color: #CC9900;
     text-decoration: none;
}

a:visited {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 14px;
     color: #996600;
     text-decoration: none;
}

a:hover {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 14px;
     color: #999966;
     text-decoration: underline;
}

a:active {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 14px;
     color: #CCCC33;
     text-decoration: none;
```

```
}

/* begin content positioning */

#collage {
     position:absolute;
     left:0px;
     top:4px;
     width:800px;
     height:58px
}

#header {
     position:absolute;
     left:0px;
     top:76px;
     width:800px;
     height:35px;
}

#nav {
     position:absolute;
     left:204px;
     top:93px;
     width:596px;
     height:32px;
}

#content {
     position: absolute;
     width: 600px;
     padding: 10px;
     margin-top: 100px;
     margin-bottom: 20px;
     margin-right: 200px;
     margin-left: 200px;
}

/* begin custom classes */

.footer {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 12px;
     color: #CC9900;
     text-align: center;
}
```

Listing 8: Global styles including presentation and layout (except the .nav class)

Horizontal Lists for Navigation

An interesting technique being used by a lot of CSS designers lately is the use of lists for navigation. Standards evangelists are big into document structure, and when you think about it, navigation is really just a list of links.

```
<ul>
<li><a href="home.html">Home</a></li>
<li><a href="books.html">Books</a></li>
<li><a href="articles.html">Articles</a></li>
<li><a href="events.html">Events</a></li>
<li><a href="courses.html">Courses</a></li>
<li><a href="consultation.html">Consultation</a></li>
<li><a href="about.html">About Molly</a></li>
<li><a href="fun.html">Fun Stuff</a></li>
</ul>
```

Listing 9: The markup for my navigation list.

We met this technique in Chapter Two – but this time, using CSS, we'll turn it onto a horizontal navigation scheme. Using lists in this way can create a wide range of opportunities for graphic-free navigation with colors and effects. For this design, I've kept the process very simple. To achieve this effect, I created a class called **.nav**. Here's the style for this class:

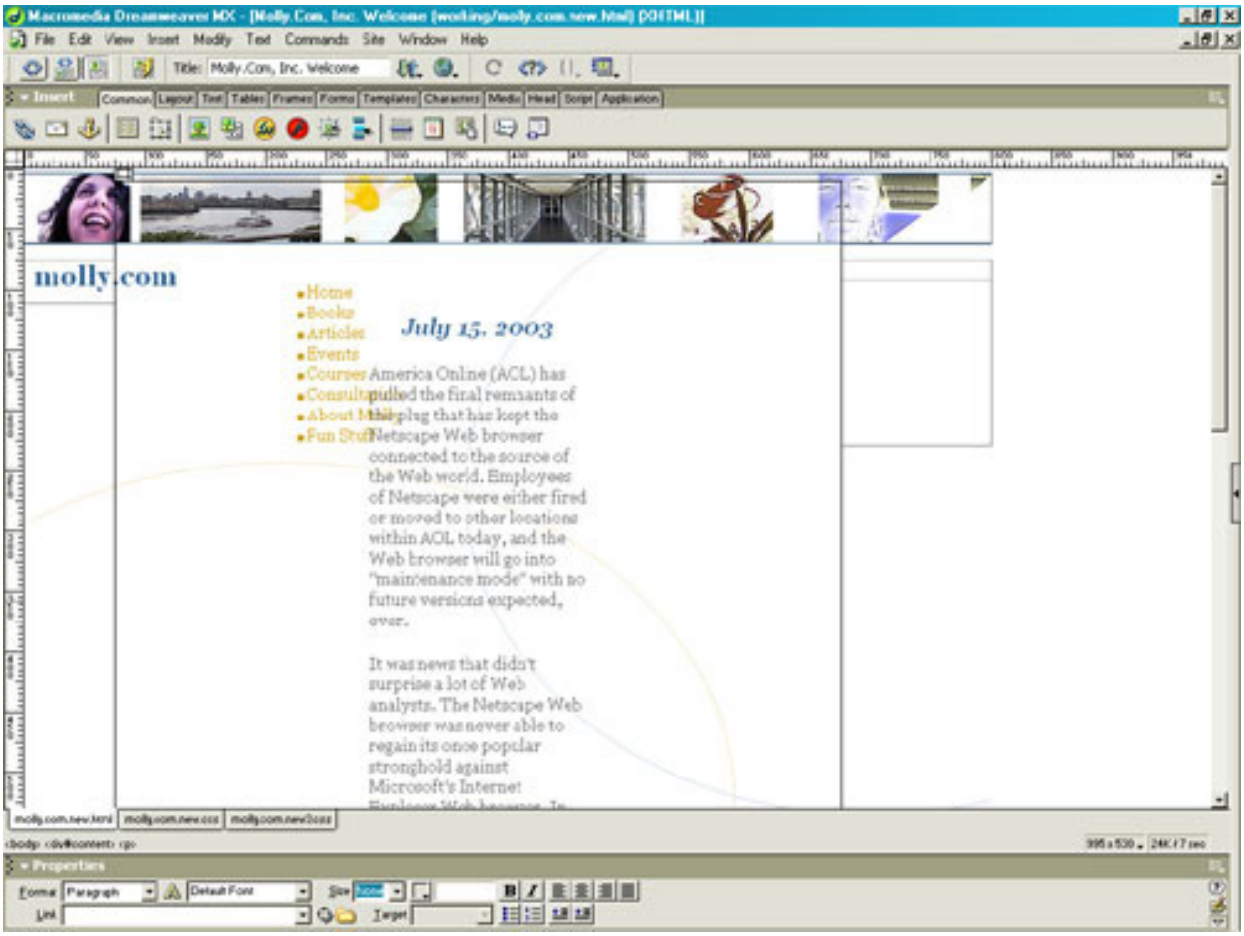
```
.nav {
     font-family: Georgia, "Times New Roman", Times, serif;
     font-size: 14px;
     color: #CC9900;
     display: inline;
     list-style-type: none;
     padding-right: 10px;
}
```

Listing 10: Creating a list-based navigation scheme.

Of course you're familiar with the font, color, and padding properties, but the two properties here that might be new to you are **display** and **list-style-type**.

The display property allows you to change the display type of a given element. Lists are normally considered block elements, which is why there are carriage returns after each closing . Using CSS, however, you can change the display to inline, so that there are no carriage returns! This is a wonderful example of how presentation and structure can truly be separated with CSS --by using CSS to change a list's display, you've successfully modified its presentation without *ever* mucking with the structure of the HTML or XHTML document. The list-style-type property allows you to modify the appearance (or lack thereof) of bullets and numbers within lists.

There is one annoyance when using lists in Dreamweaver MX for navigation. Design View will not display your lists inline (Figure 2), so you have to work in an external browser as you update your changes.



F

figure 2: Dreamweaver Design View won't provide the best representation of your CSS-based layout and list styles.

So, I added the class to both the <ul> tag and the <li> tags. I do this because I've found inconsistencies in the way browsers handle this technique, and putting the class in both places is a safety net. There are other ways this could be done as well--you could create HTML selectors instead of a class--it's really up to your personal CSS methods.

The resulting markup is incredibly clean and well-structured:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<title>Molly.Com, Inc. Welcome</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>
<link href="molly.com.new.css" rel="stylesheet" type="text/css" />
</head>

<body>

<div id="collage"><img src="top-main.jpg" width="800" height="58"
alt=”photographic collage” /></div>
<div id="header">
<h1>molly.com</h1>
</div>

<div id="nav">
<ul class="nav">
<li class="nav"><a href="home.html">Home</a></li>
<li class="nav"><a href="books.html">Books</a></li>
<li class="nav"><a href="articles.html">Articles</a></li>
<li class="nav"><a href="events.html">Events</a></li>
<li class="nav"><a href="courses.html">Courses</a></li>
<li class="nav"><a href="consultation.html">Consultation</a></li>
<li class="nav"><a href="about.html">About Molly</a></li>
<li class="nav"><a href="fun.html">Fun Stuff</a></li>
</ul>
</div>

<div id="content">

<p class="footer">&copy; Copyright 2003 - 2004 Molly.Com, Inc.<br /><a
href="privacy.html">Privacy</a> - <a href="credits.html">Credits</a></p>
</div>

</body>
</html>
```

Listing 11: The final XHTML page, ready for content.

The next step was to add some content, and test in a variety of browsers. Figure 3 shows the results in Internet Explorer 6.0 for Windows.

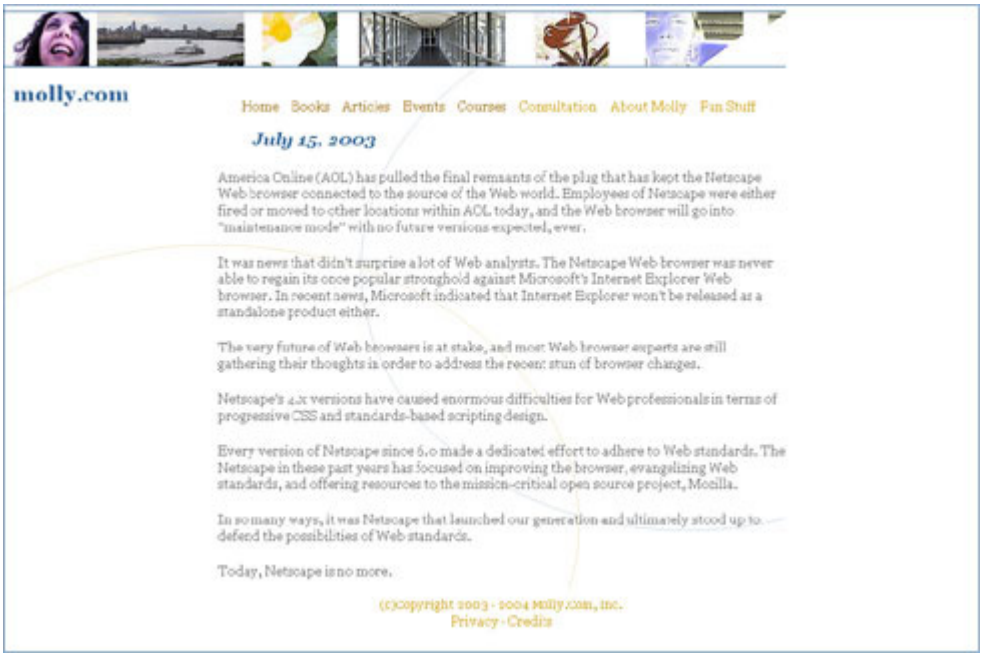


Figure 3: The results!

The results shown here are almost identical within every contemporary browser with comprehensive CSS support. Netscape 4 even retains the layout, but the caveat there is that using **display:inline** won't work in Netscape 4. If you have to support non-CSS browsers, you will have to test adequately if you're using lists for navigation, especially if you're displaying them inline.

Creating Alternative Designs

Of course, the beauty of style sheets is that you can modify them very quickly, completely changing the look of your design without having to touch the HTML or XHTML. I went ahead and created two additional style sheets.

The first variant is the same design, but with a black theme:

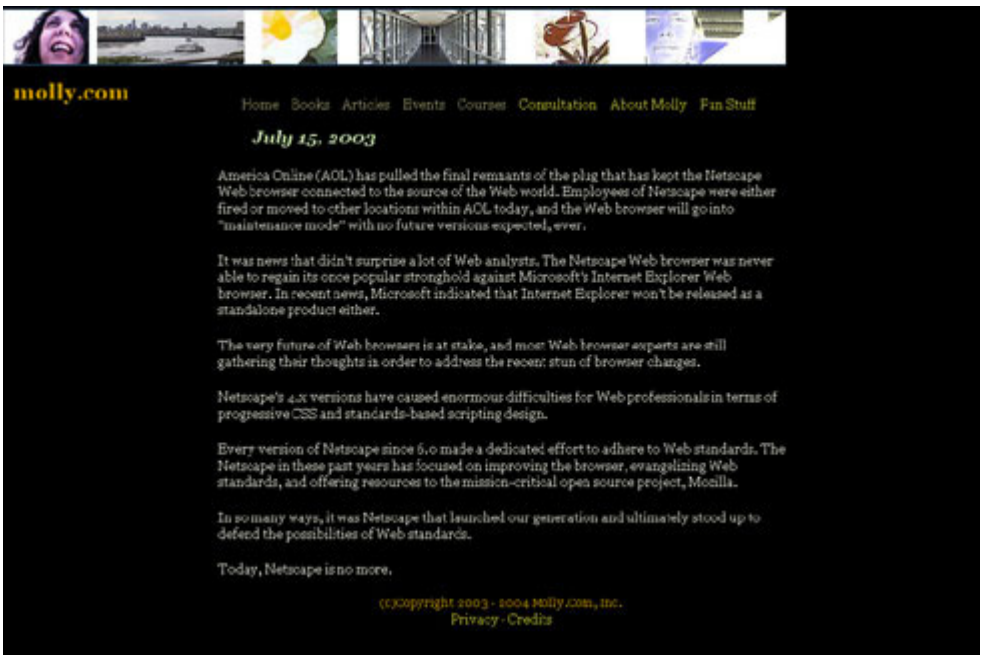


Figure 4: Same layout, modified style sheet.

I also created a more "feminine" version:

Figure 5: Changing style sheets can bring about a dramatically different look.

So, I end up with a visually attractive, easily navigable, accessible, and modifiable Web log design that is also incredibly optimized for speed. Consider that the XHTML document *with* an entry weighs in at a whole 3KB, and the graphics including the background (which is only used in the first version), are 26KB, and you end up with a Web page that is 29KB.

9.Switching Styles: Users-selected style sheets.

Style switching using CSS and JavaScript is not only a fun way to add interactivity to your site, but also very practical. If you've ever wanted to give your site visitors a little more control and access to your site, this chapter will help show you how. You'll also examine some important issues in CSS including how browsers manage multiple linked styles.

About Style Switching

Switching styles is a growing practice used by such CSS notables as Eric Meyer. On his web site, Meyer offers site visitors a number of various styles from which to choose. He's selected a default, but the site visitor can choose a style once there, giving the site a different presentation.

Three views of meyerweb.com: Same content, different styles

This technique is as practical as it is fun. Not only does it provide an enjoyable, enhanced experience for the site visitor, but it can allow you to provide versions of the site that suit different audience needs, such as a high-contrast version for those with vision impairments, or, if your site is all CSS-based in terms of layout, you can provide an alternate design for browsers that do not support CSS, allowing site visitors to get a better visual experience—on their terms.

In a similar vein, the style-switching technique can be used to create text-size switching effects. If your site uses small or normal text sizing, and you'd like to allow your visitor a one-click option to make their text size larger (always helpful for aging eyes!) This is also an important feature for accessibility, and is in use on many web sites, including Jeffrey Zeldman's Daily Report and A List Apart.

Zeldman's Daily Report offers sizing and contrast options for customized viewing.

Even with such high profile individuals using switching, many designers have never tapped into its power and simplicity. Switching styles is a fairly easy process once you've got all the pieces in order. For this chapter's exercise, I've provided you with all the files you need to get started, but for general purposes, style switching requires the following:

- A contemporary web browser with CSS, JavaScript and cookies turned on (Microsoft IE, Netscape 6.x and above, Mozilla, and Opera all support this script)
- A basic style sheet for the page's design
- Additional, alternate style sheets
- A style switching script (I've included an open-source version in this article)
- Anchor elements with event handlers used to invoke and switch the script

And of course, because this technique requires JavaScript, your site visitors will have to have JavaScript enabled on their browsers in order to tap into the switch. If they do not, it will not interfere with your site's basic performance; rather, they simply will not be able to use the feature.

Setting Up

To get set up for the exercises, begin by downloading the exercise files. Once they are available on your hard drive, unpack the downloaded package as follows:

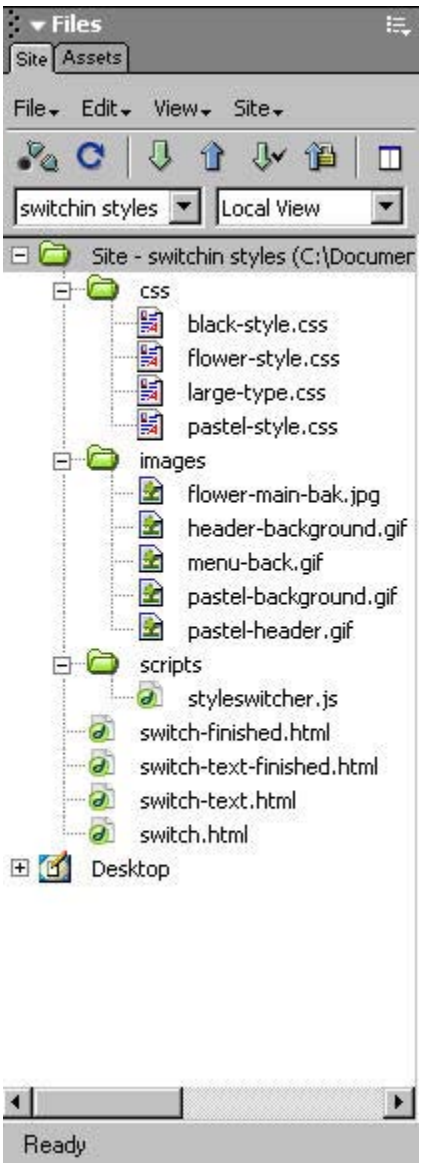
1. Double-click on the compressed file. Your default compression program will attempt to open the file.
2. Choose to decompress (or "extract") the file. You may be asked for a location, if so, provide the location on your hard drive as to where you'd like to place your work files.
3. Once the files decompress, you should see a number of HTML files, and three sub-directories: images (which contain all the images needed for the exercise); css (which contain all the CSS for the exercise); and scripts (which contains the switching script).

If your compression utility did *not* decompress the files with the sub-directories intact, and you've just gotten the individual files all at once, you'll want to create each of these folders. Then, place all files ending with .html into the root directory; all files ending in .css into the css directory, and the file with the .js extension into the scripts folder.

You're ready to define the site in Dreamweaver. To do so:

1. Select Site > New Site. The Site Definition dialog appears.
2. Name your site **switchin styles** and click Next. You'll be asked if you're going to be working with a server technology.
3. Be sure No is selected. You can always change this later if you should so desire. Click Next.
4. Set your editing preferences to edit your copy locally, then upload manually when recommended.
5. Within the same dialog, you'll be asked where you'd like to store the files. Click the folder icon and browse to the directory where you've placed the exercise files. Select it, and once you've returned to the Site Definition dialog, click Next.
6. Since we won't be working with a remote server, choose "None" From the drop-down menu on the Sharing Files menu. Click Next.
7. You'll see a summary report of your site definition. Review it for accuracy, and when happy that your settings are accurate, click Done.

The newly defined site will appear in your Files panel, and you're ready to get to work.



The exercise defined as a site within Dreamweaver.

Using the Style Switcher for Alternative Designs

In this section, you'll use the style switching script in conjunction with three CSS files to allow the site visitor to switch between visual styles.

Linking to the Style Sheets

The first step in the process is to link to the various styles you'll be using for your switch. This is perhaps the most critical part of the process, because the way you author the links is critical to the browser's proper management of the technique. You'll have to manually edit some of the attributes in the link markup as Dreamweaver MX doesn't include all of the necessary attributes for the **link** element.

Note: It's important to know that the style switching technique does not support imported CSS, so if you're using the @import technique as a workaround for your CSS layouts, you'll either have to forgo the workaround and place all of your styles for a given design into a linked style sheet, or keep the layout the same for all designs, import that file, but only change the visual presentation options. Style switching does not apply to embedded or inline styles, only to linked ones.

The first style to be added will be the default style—this is the preferred default style, and will be the one that appears to the site visitor upon first visit.

Adding the Default Style

To add the default CSS style link, follow these steps:

1. Open **switch.html** in Code View. You'll notice that this is a simple document written in table-less HTML 4.01. You can use XHTML if you prefer, it makes no difference so long as you follow appropriate syntax for the language at hand.
2. Place your cursor in the **head** portion of the document, under the **title** and **meta** elements.
3. Click the Head tab for easy access to head-related elements.
4. From the Head tab, click the link icon. The Link dialog appears.
5. You'll add your default link by selecting Browse, highlighting **black-style.css** (found in the CSS folder for this site). Make sure the Relative To drop-down menu is set to Document.
6. Click OK. You'll be returned to the Link dialog. In the Rel textbox, type the word **stylesheet**. Leave the other options blank.
7. Click OK.

Dreamweaver adds the **link** markup. I want you to examine the markup carefully:

```
<link href="css/black-style.css" rel="stylesheet" type="text/css">
```

Pretty straight-forward, but the important thing of which to be aware is that it's the "stylesheet" value of **rel** and the *lack* of a title attribute and value that allows the browser and switcher script to identify this sheet as the default sheet.

Tip: You may wish to add the media type using the **media** attribute and an appropriate value. You'll see that in my markup, I've used "all" so these styles will be applied to all media including the screen. If I wanted to have an alternate style sheet for print or projection, I can define that using the **media** attribute accordingly. Here's my markup after I've manually added this attribute in Code view:

<link href="css/black-style.css" rel="stylesheet" type="text/css" media="all">

Adding the Alternate Stylesheets

To add the alternate style sheets, follow these steps:

1. Place your cursor underneath the **link** to the default style.
2. Click the Link icon on the Head tab. The Link dialog appears.
3. Navigate to black-style.css and highlight it. I'm adding it again because I want it not only to be available as default, but as an option for the switching process, too. Click OK.
4. In the Rel textbox, type **alternate stylesheet**. This identifies to the browser and the script that the style sheet is an alternate—not the default.
5. In the Title textbox, type in the word **black**. This helps the script identify which style sheet to use when a site visitor selects the black style.
6. Click OK. You can manually add the **media** attribute if you so desire.

My markup for the first alternate style option looks like this:

```
<link href="css/black-style.css" rel="alternate stylesheet"
type="text/css" title="black" media="all">
```

Now, repeat the steps in this section to add the two additional alternate styles. Be sure that you use the word **flower** for the flower-style.css title, and **pastel** for the pastel-style.css title. When you're finished, your styles should appear as follows:

```
<link href="css/black-style.css" rel="stylesheet" type="text/css"
media="all">

<link href="css/black-style.css" rel="alternate stylesheet"
type="text/css" title="black" media="all">

<link href="css/flower-style.css" rel="alternate stylesheet"
type="text/css" title="flower" media="all">

<link href="css/pastel-style.css" rel="alternate stylesheet"
type="text/css" title="pastel" media="all">
```

If you've not manually added the **media** attribute, it won't appear. While it's not necessary to identify the media type in this case--everything is set to "all", which is the default behavior of the browser if there isn't another media type specified--it's a good habit to get into when working with styles.

Linking to the JavaScript

In your scripts folder, you'll see a file called **styleswitcher.js**. This is an open-source JavaScript created by Paul Sowden. If you're a JavaScript maven, you can modify the script as you see fit, but for basic style switching purposes, you won't need to modify a thing, only make sure you link to the file properly. To do so:

1. In Code view, place your cursor below the last style sheet link.
2. Click the Script tab to bring up the script options.
3. From the Script tab, click the script icon. Make no modifications in this dialog, simply click OK.
4. The **script** element has now been added, but you'll want to manually modify this. First, highlight the **language="JavaScript"** attribute and delete it. The reason is that this attribute isn't supported in some versions of HTML and XHTML, so it's good practice to keep it out of your documents.
5. Now, type **src="scripts/styleswitcher.js"** into the opening **script** tag. This links the script to your document.

It's very important that the link to your script appears *after* the styles. When you're finished with this step, the markup should look like mine, as follows:

```
<link rel="stylesheet" type="text/css" href="css/black-style.css"
media="all">

<link rel="stylesheet" type="text/css" href="css/black-style.css"
title="normal" media="all">

<link rel="alternate stylesheet" type="text/css" href="css/large-
type.css" title="large" media="all">

<script type="text/javascript" src="scripts/styleswitcher.js"></script>
```

Adding Hooks to the HTML Document

Believe it or not, all you've got left to do is add hooks for the script-switching into your HTML document. To do so, you'll provide a series of links with the **onclick** event handler. I've already provided this markup for you in the document, but you'll want to examine it here so you'll know how to add this to your own documents.

```
<ul>

<li><a href="#" onclick="setActiveStyleSheet('black'); return
false;">black style</a></li>

<li><a href="#" onclick="setActiveStyleSheet('floral'); return
false;">floral style</a></li>

<li><a href="#" onclick="setActiveStyleSheet('pastel'); return
false;">pastel style</a></li>

</ul>
```

In this case, I've added the options to a separate list within the menu division of the document. Clicking the links will now allow the site visitor to switch between my styles. You can (and will!) make this more decorative by adding graphic buttons or styling the text as you see fit.

At this point, be sure to save all of your files, and then go ahead and load switch.html into a supporting browser and switch away!

Using the style switcher to switch back and forth between various looks.

You can modify this any way you like, including adding graphic buttons or differently, more text with more complex styling.

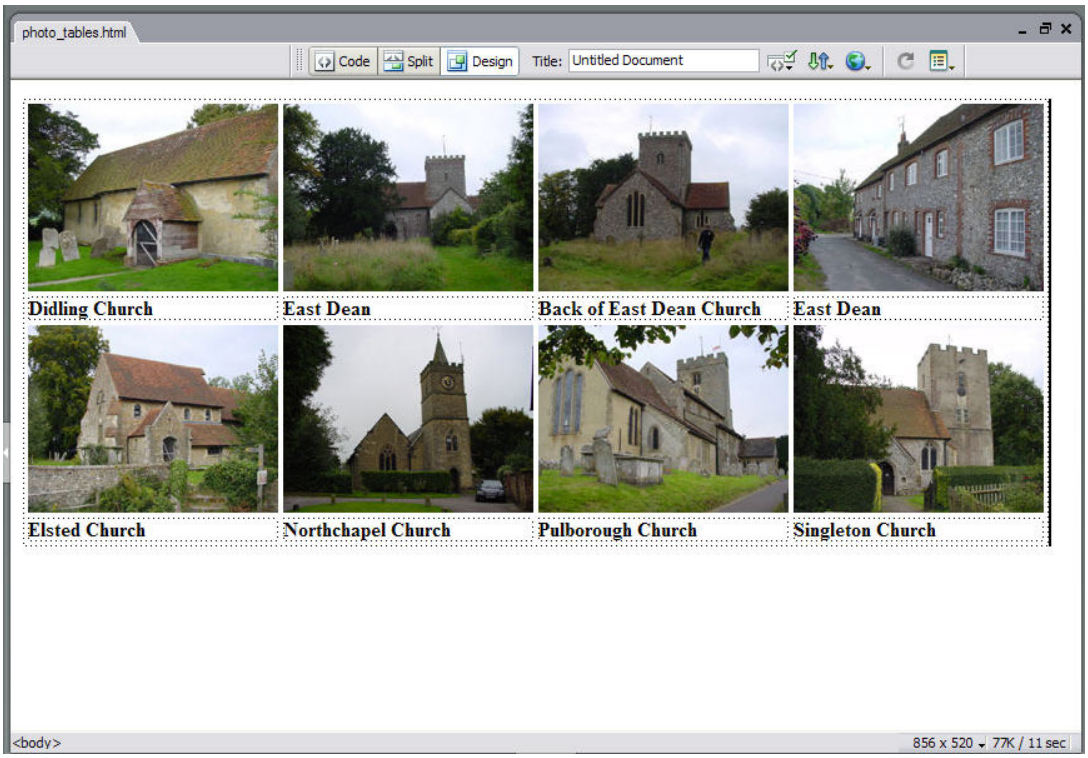
10. Using the Float Property: an all-CSS Photo Album layout

If you have moved to using CSS instead of tables for layout, then you will probably find that you come up against problems that had a simple solution when using tables, but it is difficult to work out exactly how to get round them using CSS. A photo album – or other listing of images, with a grid of photo thumbnails with captions is one of these issues. In this chapter, we will learn how to display these images using CSS and in a way that offers benefits over and above that which can be achieved with tables.

As a result of following this chapter you will not only have discovered a way in which you can replace tables-based grids for the display of items like a photo album listing, but will also have learned to use the useful float property in CSS, which has a whole range of uses and can be put to use in creating interesting and creative effects in your designs.

Old-style layout using tables

The sort of layout we are looking to create would have been created with a table which included a cell for the picture, then a cell beneath it containing a caption that described the picture. I'm sure that you have far more interesting photographs than these churches that I drag my other half around the countryside to photograph, but whichever photographs you choose, it is advisable for them to be of the same height and width – this will work if they are not, but it looks far neater if you have nice neat thumbnails in the listing.



The photo album laid out using a table

```
<table border="0" cellspacing="4" cellpadding="0">
 <tr>
  <td><img src="didling_church_thumb.jpg" width="200" height="150"
alt="Didling Church, West Sussex" /></td>
  <td><img src="eastdean_church01_thumb.jpg" width="200" height="150"
alt="East Dean Church, West Sussex" /></td>
  <td><img src="eastdean_church02_thumb.jpg" width="200" height="150"
alt="East Dean Church" /></td>
  <td><img src="eastdean_street_thumb.jpg" alt="East Dean" width="200"
height="150" /></td>
 </tr>
 <tr>
  <td><strong>Didling Church</strong></td>
  <td><strong>East Dean </strong></td>
  <td><strong>Back of East Dean Church </strong></td>
  <td><strong>East Dean </strong></td>
 </tr>
 <tr>
```

```
  <td><img src="elsted_church_thumb.jpg" alt="Elsted Church, West Sussex"
width="200" height="150" /></td>
  <td><img src="northchapel_church_thumb.jpg" alt="Northchapel Church,
West Sussex" width="200" height="150" /></td>
  <td><img src="pulborough_church_thumb.jpg" alt="Pulborough Church. West
Sussex" width="200" height="150" /></td>
```

```
  <td><img src="singleton_church_thumb.jpg" alt="Singleton Church, West
Sussex" width="200" height="150" /></td>
 </tr>
 <tr>
  <td><strong>Elsted Church </strong></td>
  <td><strong>Northchapel Church </strong></td>
  <td><strong>Pulborough Church </strong></td>
  <td><strong>Singleton Church </strong></td>
 </tr>
</table>
```

There are a couple of immediate problems with this layout. The screen reader is going to linearize the table so it will read out the alt text of each picture and then read out the caption below, so the user will hear, “Didling Church, West Sussex, East Dean Church, West Sussex, East Dean Church, East Dean, Didling Church, East Dean, Back of East Dean Church, East Dean”. Even if the caption explained far more about the contents of the picture, they are divorced from the picture itself and so not very useful in helping to explain the content and context of the image.

Another problem is that although you can make the table flexible by giving it a 100% width, it will always be 4 columns wide, so if the user has a large, high resolution screen the images may seem stuck up in the corner of the site, as they won’t wrap to fit the available space.

Photo Album layout using CSS

To layout our album using CSS, we are going to use the **float** property. A floated element is taken out of the normal flow of the document and positioned as far to the left or right of the parent element as is possible, unless it comes across another floated element. As an example, place an image and a paragraph of text into a document.

```
<p><img src="didling_church_thumb.jpg" alt="Didling church" width="200"
height="150" /></p>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque
habitant morbi tristique senectus et netus et malesuada fames ac turpis
egestas. Sed pulvinar faucibus eros. Mauris et pede eu massa dictum
eleifend. Nulla in tortor eget nunc condimentum ornare. Praesent sed
augue. Vestibulum wisi ante, consectetuer at, scelerisque pellentesque,
semper vel, libero. Curabitur turpis sem, scelerisque a, varius eu,
feugiat sed, neque. Cras id urna. Mauris rhoncus sapien non tellus.
Nullam dapibus, nisl at congue auctor, tortor magna mollis metus, id
aliquet mi turpis a justo. </p>
```

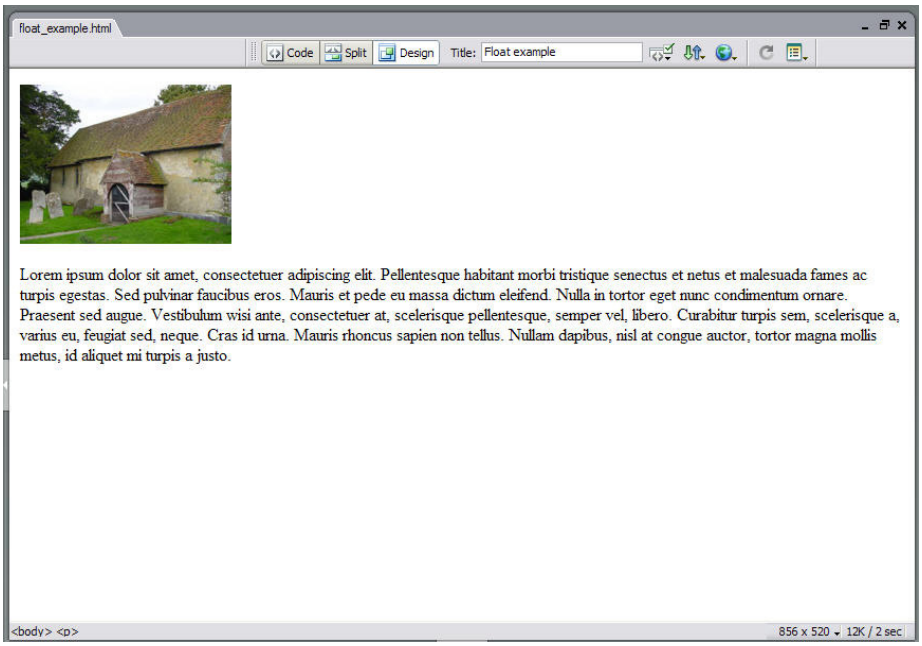
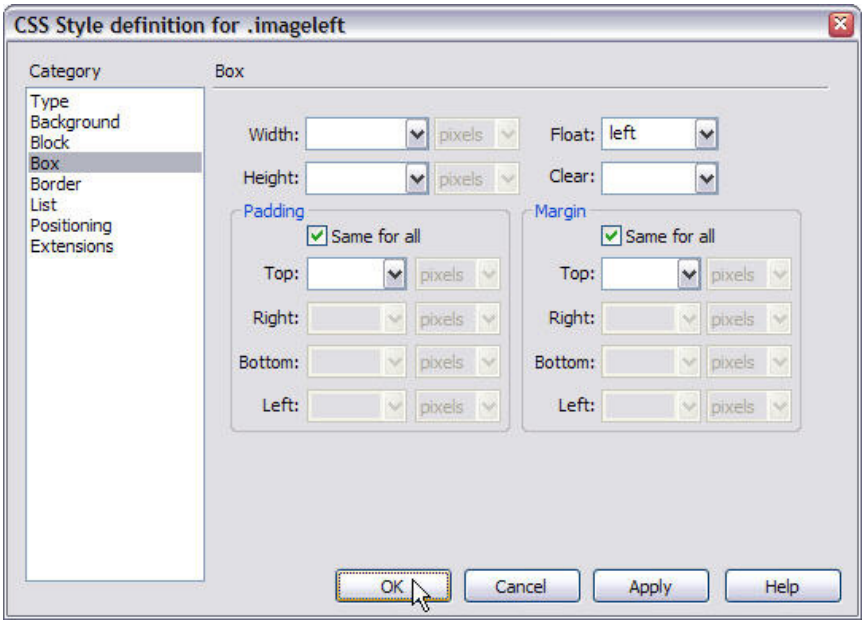


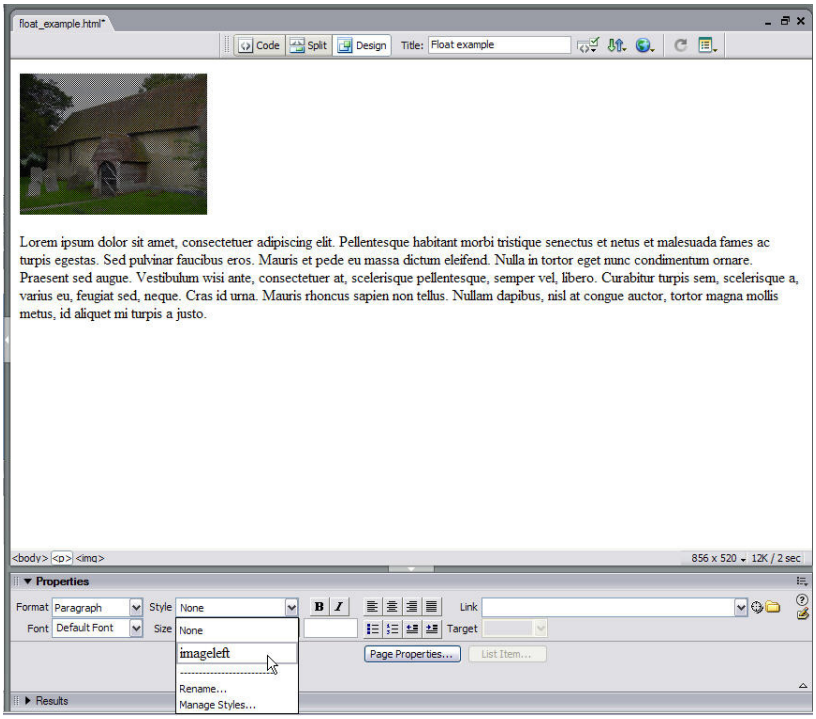
Image and text in the document

Now create a CSS class named **.imageleft**. In the **Box** category of the CSS Styles Definition Dialog, set float to 'left' then click OK.



Creating the .imageleft class

Now apply the class to the paragraph tags that surround the image.



Applying the class 'imageleft' to the <p> tag surrounding the image.

You will find that the text now wraps the image, because the image has been taken out of the document flow.

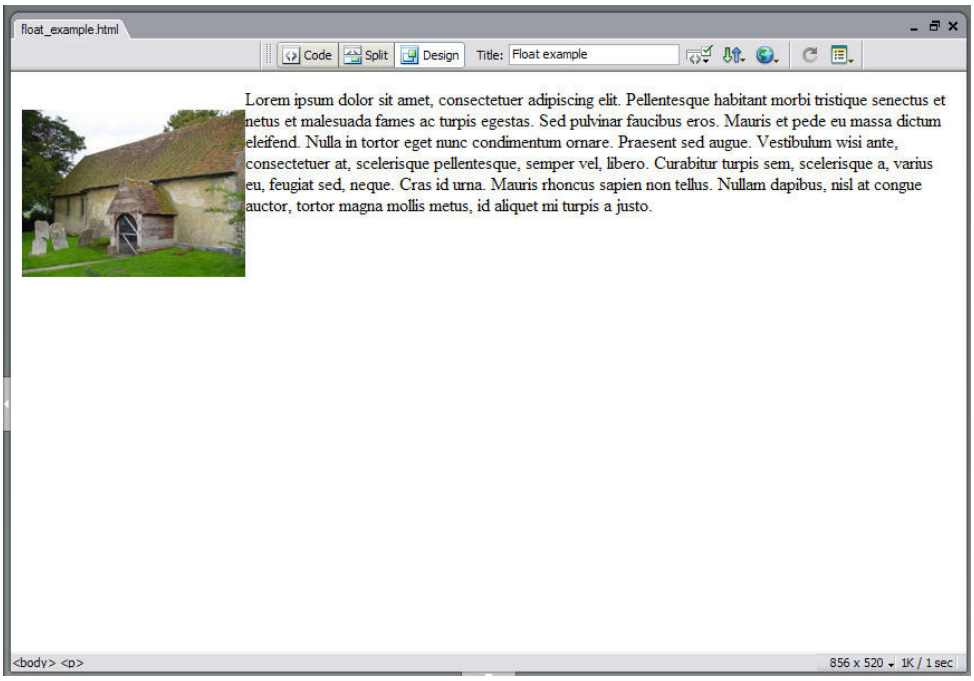
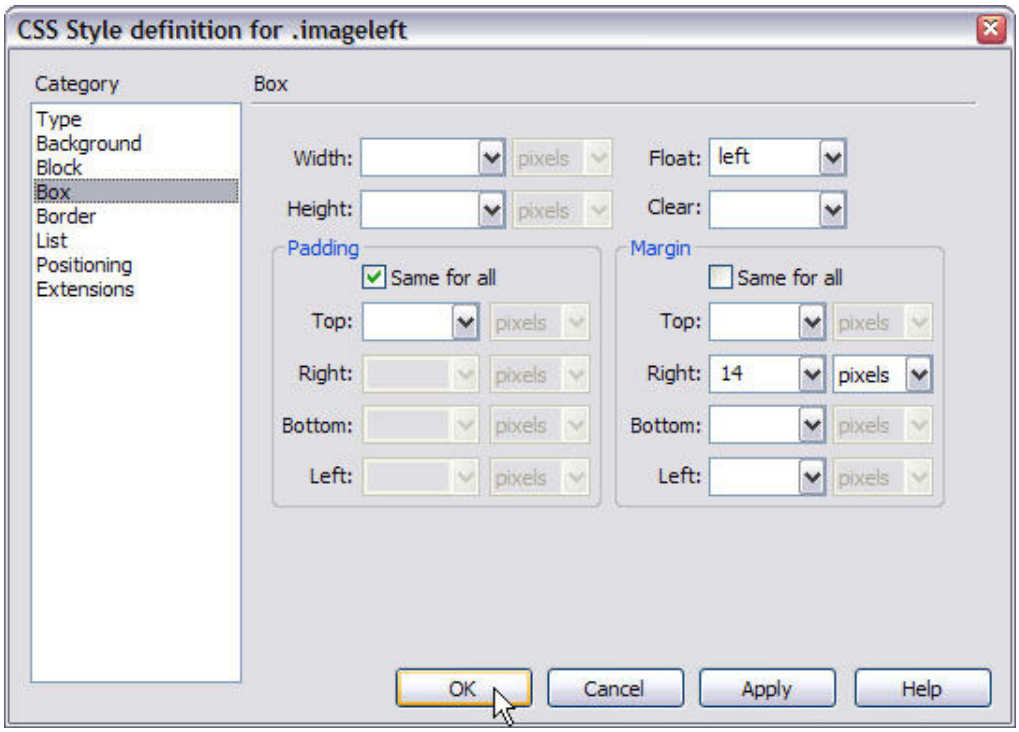


Image floated left

To make some space around the image so that the text does not bump up against it, edit the floatleft class again, and add a right margin.



Add a right margin to the floatleft class

If you try and add another paragraph of text to the page, you will find that it to wraps alongside the image if the page is wide enough to allow that. To stop that from

happening, you need to explicitly clear the floated element. The least troublesome way to do this is to add an empty div before the element that you wish to remain clear of the floated element. The div should have a class applied to it. The CSS class is simply:

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

So the mark-up would look like:

```
<p class="imageleft"><img src="didling_church_thumb.jpg" alt="Didling
church" width="200" height="150" /></p>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque
habitant morbi tristique senectus et netus et malesuada fames ac turpis
egestas. Sed pulvinar faucibus eros. Mauris et pede eu massa dictum
eleifend. Nulla in tortor eget nunc condimentum ornare. Praesent sed
augue. Vestibulum wisi ante, consectetuer at, scelerisque pellentesque,
semper vel, libero. Curabitur turpis sem, scelerisque a, varius eu,
feugiat sed, neque. Cras id urna. Mauris rhoncus sapien non tellus.
Nullam dapibus, nisl at congue auctor, tortor magna mollis metus, id
aliquet mi turpis a justo. </p>
<div class="clear"></div>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque
habitant morbi tristique senectus et netus et malesuada fames ac turpis
egestas. Sed pulvinar faucibus eros. Mauris et pede eu massa dictum
eleifend. Nulla in tortor eget nunc condimentum ornare. Praesent sed
augue. Vestibulum wisi ante, consectetuer at, scelerisque pellentesque,
semper vel, libero. Curabitur turpis sem, scelerisque a, varius eu,
feugiat sed, neque. Cras id urna. Mauris rhoncus sapien non tellus.
Nullam dapibus, nisl at congue auctor, tortor magna mollis metus, id
aliquet mi turpis a justo. </p>
```

We are going to use float to line up our images and captions without using a table. First, however, we need to create the document.

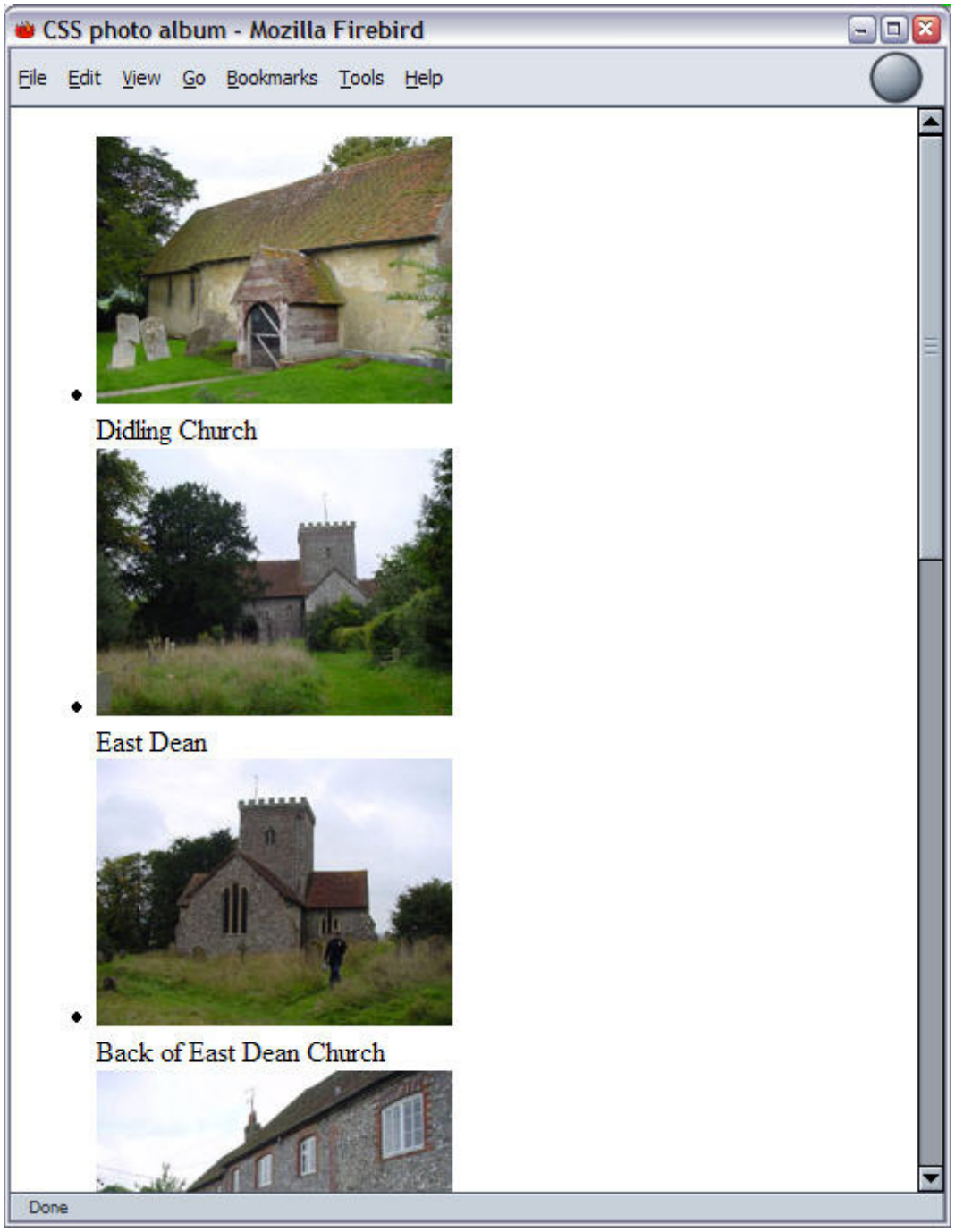
Creating the document

Structurally, what we really have in a photo album is a *list* of photos, so we will create our album as a structured html list.

```
<!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>
<title>CSS photo album</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>
```

```
</head>

<body>
<ul>
 <li><img src="didling_church_thumb.jpg" width="200" height="150"
alt="Didling Church, West Sussex" /><br />
  Didling Church
 </li>
 <li><img src="eastdean_church01_thumb.jpg" width="200" height="150"
alt="East Dean Church, West Sussex" /><br />
  East Dean
 </li>
 <li><img src="eastdean_church02_thumb.jpg" width="200" height="150"
alt="East Dean Church" /><br />
  Back of East Dean Church </li>
 <li><img src="eastdean_street_thumb.jpg" alt="East Dean" width="200"
height="150" /><br />
  East Dean
 </li>
<li><img src="elsted_church_thumb.jpg" alt="Elsted Church, West Sussex"
width="200" height="150" /><br />
  Elsted Church
 </li>
 <li><img src="northchapel_church_thumb.jpg" alt="Northchapel Church,
West Sussex" width="200" height="150" /><br />
  Northchapel Church
 </li>
 <li><img src="pulborough_church_thumb.jpg" alt="Pulborough Church. West
Sussex" width="200" height="150" /><br />
  Pulborough Church
 </li>
 <li><img src="singleton_church_thumb.jpg" alt="Singleton Church, West
Sussex" width="200" height="150" /><br />
  Singleton Church
 </li>
</ul>
</body>
</html>
```

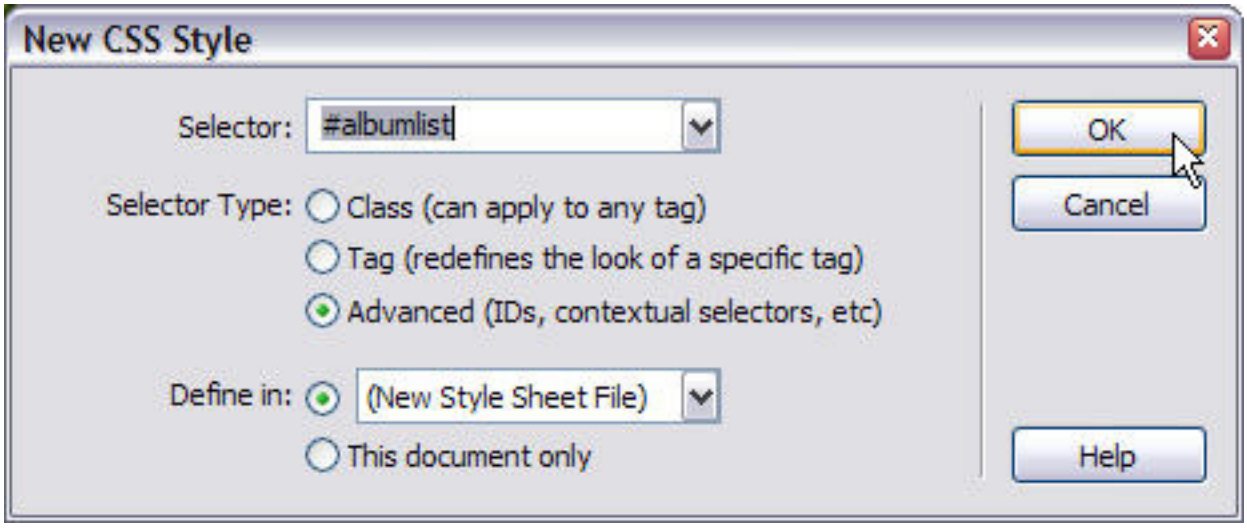
The photographs will display in the browser when unformatted as a list of items.



The photographs displayed as an unstyled list

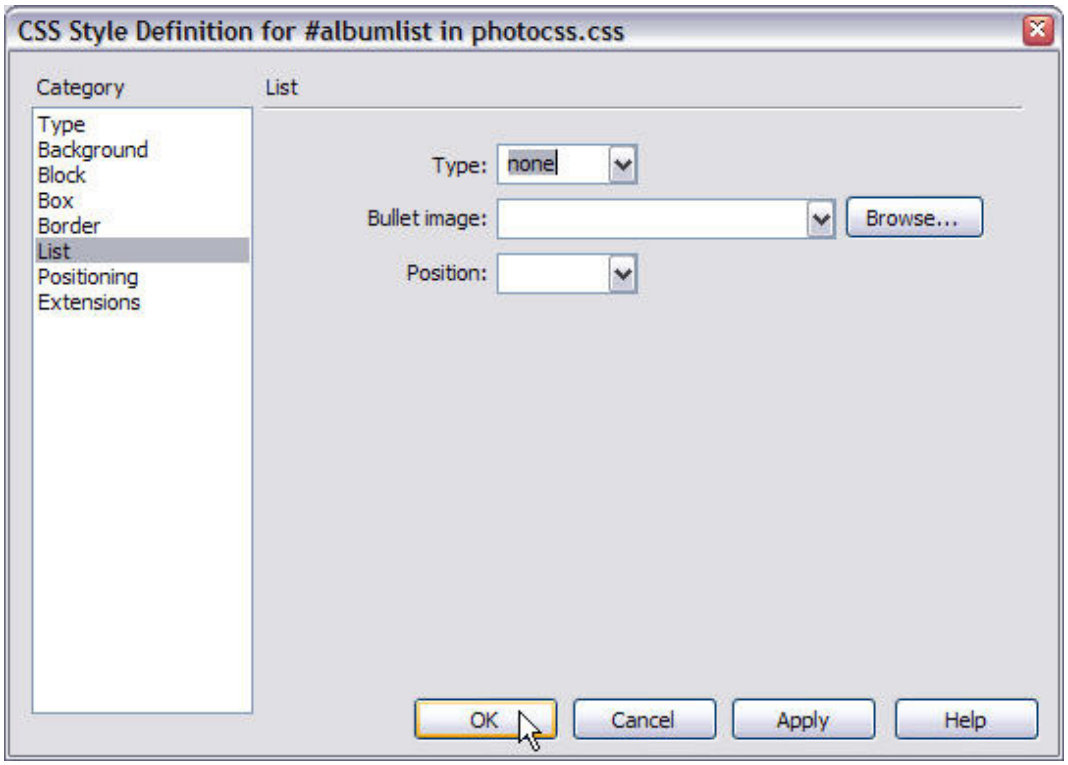
Once you have your photographs in the document in a structured format, you can begin to style them using CSS.

Create a New CSS Style, select the radio button for 'Advanced' and create it for an id '**#albumlist**'.



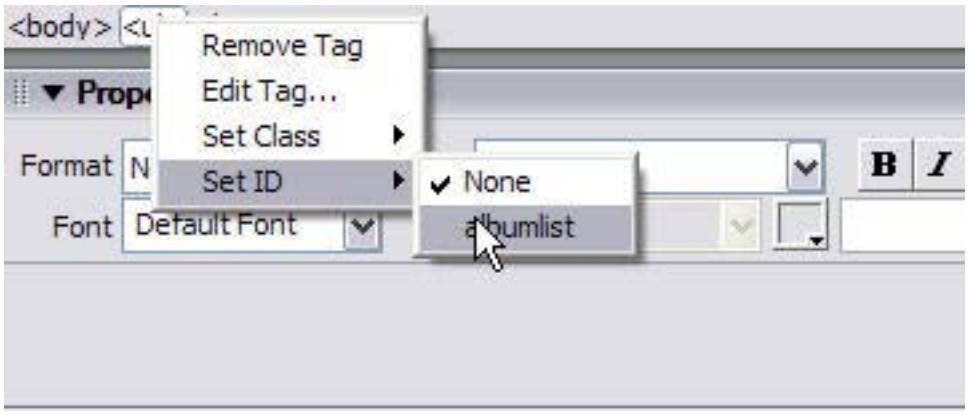
Creating styles for the id "albumlist"

In the CSS Style Definition Dialog, select the list category and set list type to 'none'. This removes the bullets on a list with this id.



Setting type to none

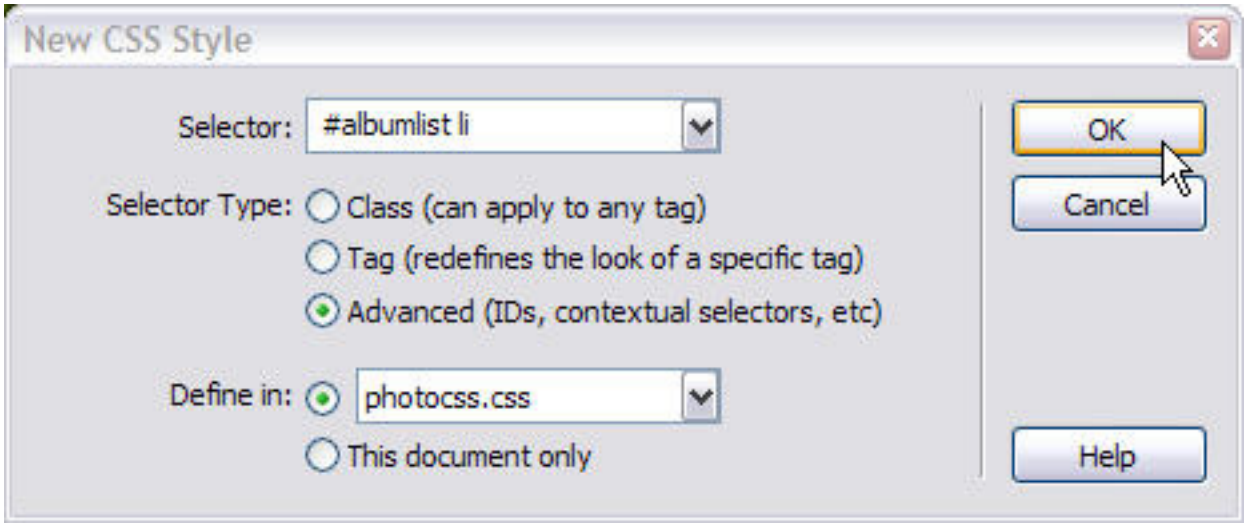
Click **OK** to close this dialog and create the id, then apply this id to the **ul** tag of the list of photographs.



Applying the class albumlist to the ul tag

After applying the class the bullets should disappear from the list.

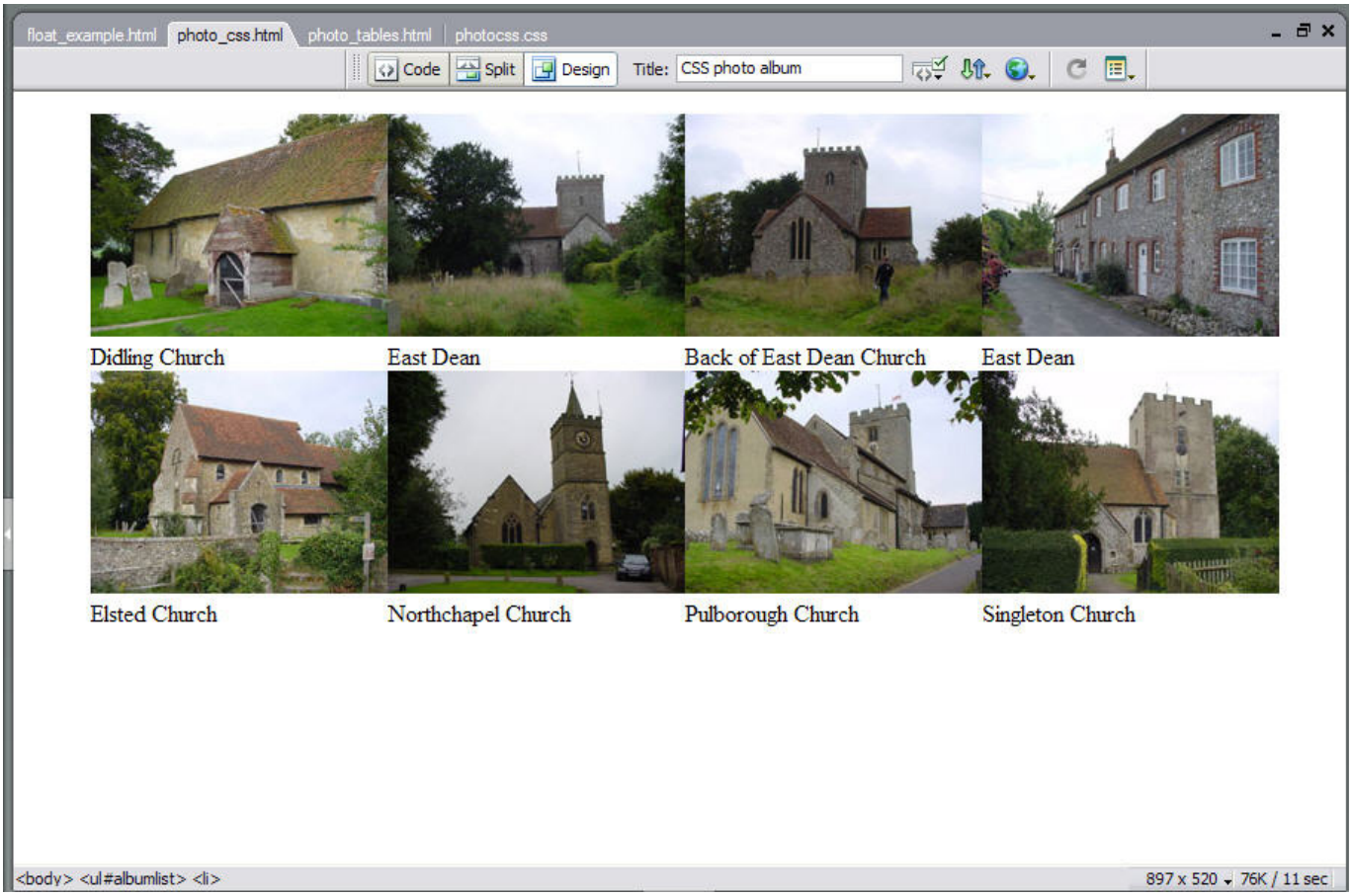
Next, create another new CSS style, this time define it for **#albumlist li** – that is, every **li** tag within the id **#albumlist**.



Creating a class for albumlist li

After creating this class, go to the Box category of the dialog and set float to left.

Click OK and the photos should move up one next to another in the Design View:

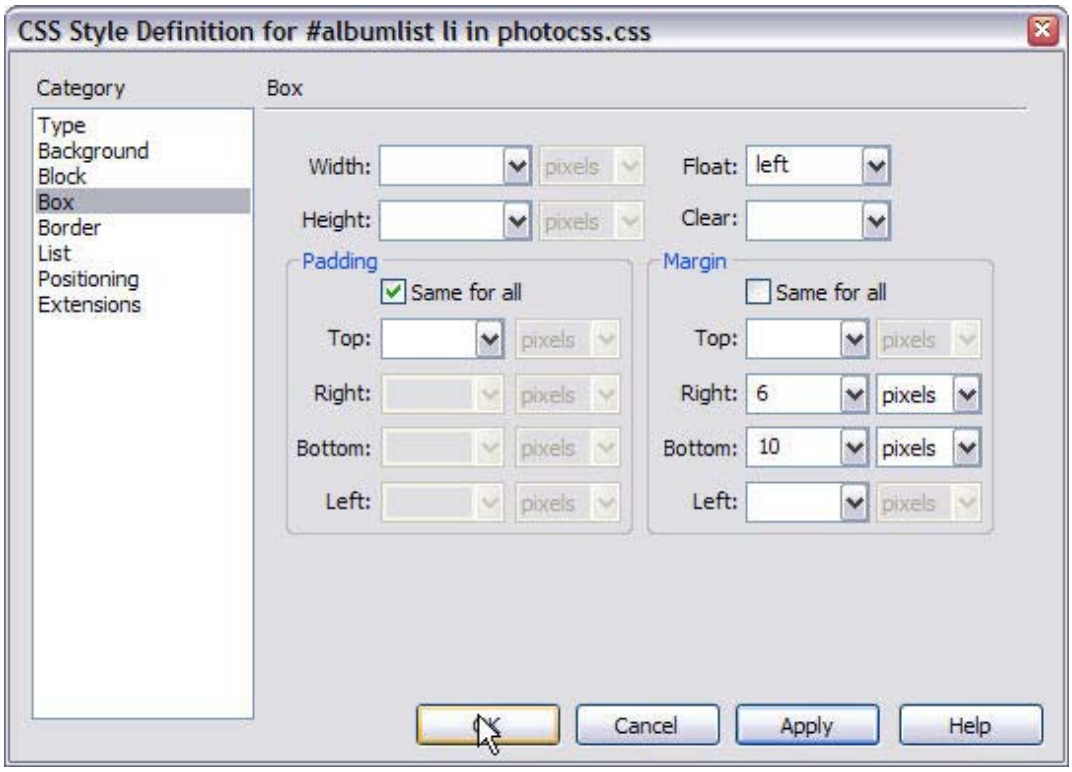


After floating the li's left

If you preview this in your browser, you will see that if you make the window wider and smaller the photos will wrap according to how much space they have.

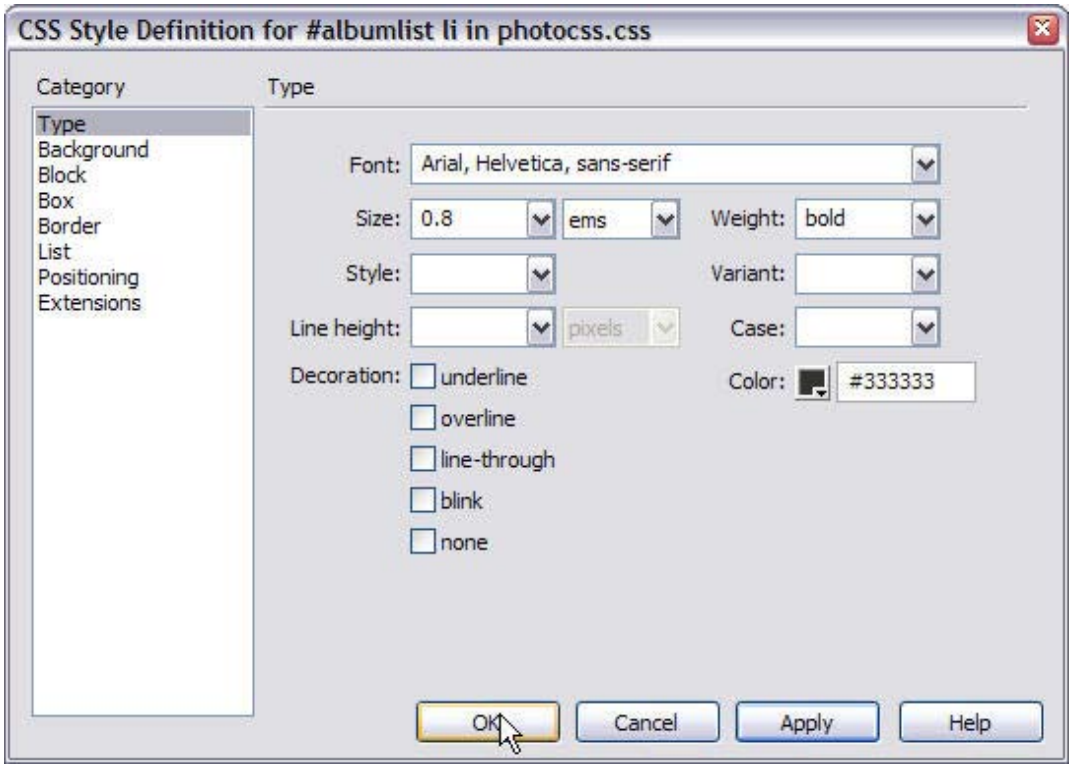
Creating space

Now that you have your basic grid layout, you will probably want to space out the thumbnails. To do so, edit the CSS style for **#albumlist li**. Add a margin to the bottom and right to give the images more space to their right and below them.



Adding padding to the class

To style the captions that are underneath the photographs, you simply style the type for this class.

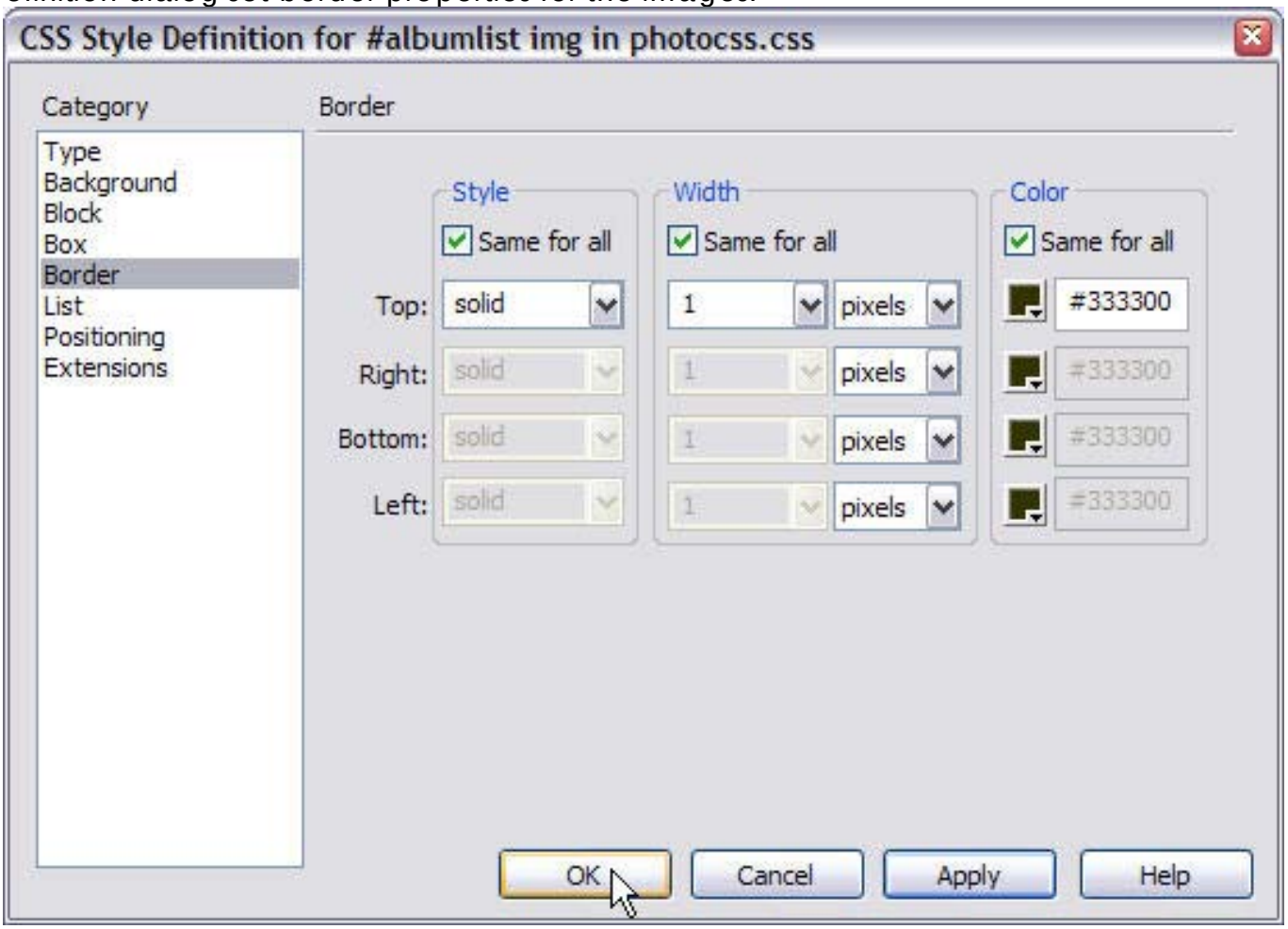


Styling the type of the image captions

Adding a border to the image

To add a border to the images, we can define a class for all img tags within #albumlist.

Create a new CSS style, for #albumlist img. In the **Border** category of the CSS Style Definition dialog set border properties for the images.



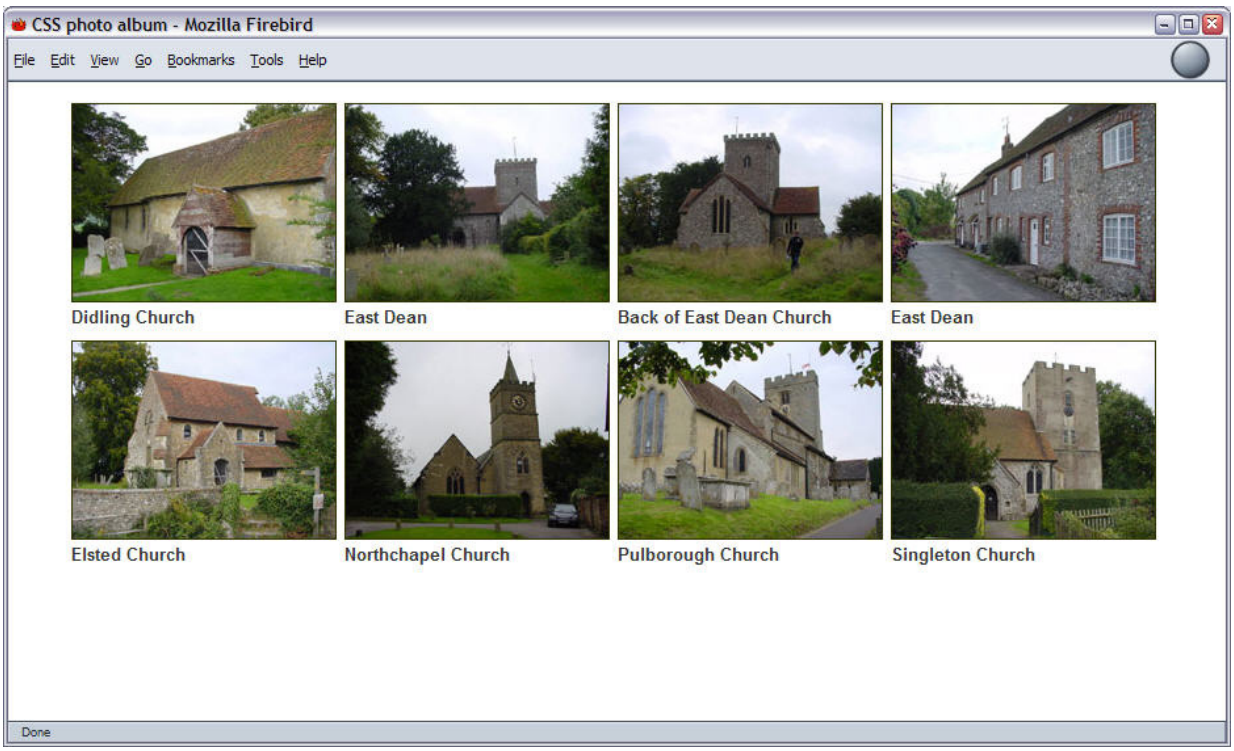
Setting border properties for the images

You can experiment with the CSS for this album layout from this point, adding background colors or other effects to the individual **li** tags, but if you look at the CSS that has been created for this so far you will see that this is a very efficient way to create the grid effect necessary.

```
#albumlist {
     list-style-type: none;
}

#albumlist li {
     float: left;
     margin-right: 6px;
```

```
     margin-bottom: 10px;
     font-family: Arial, Helvetica, sans-serif;
     font-size: 0.8em;
     font-weight: bold;
     color: #333333;
}
#albumlist img {
     border: 1px solid #333300;
}
```

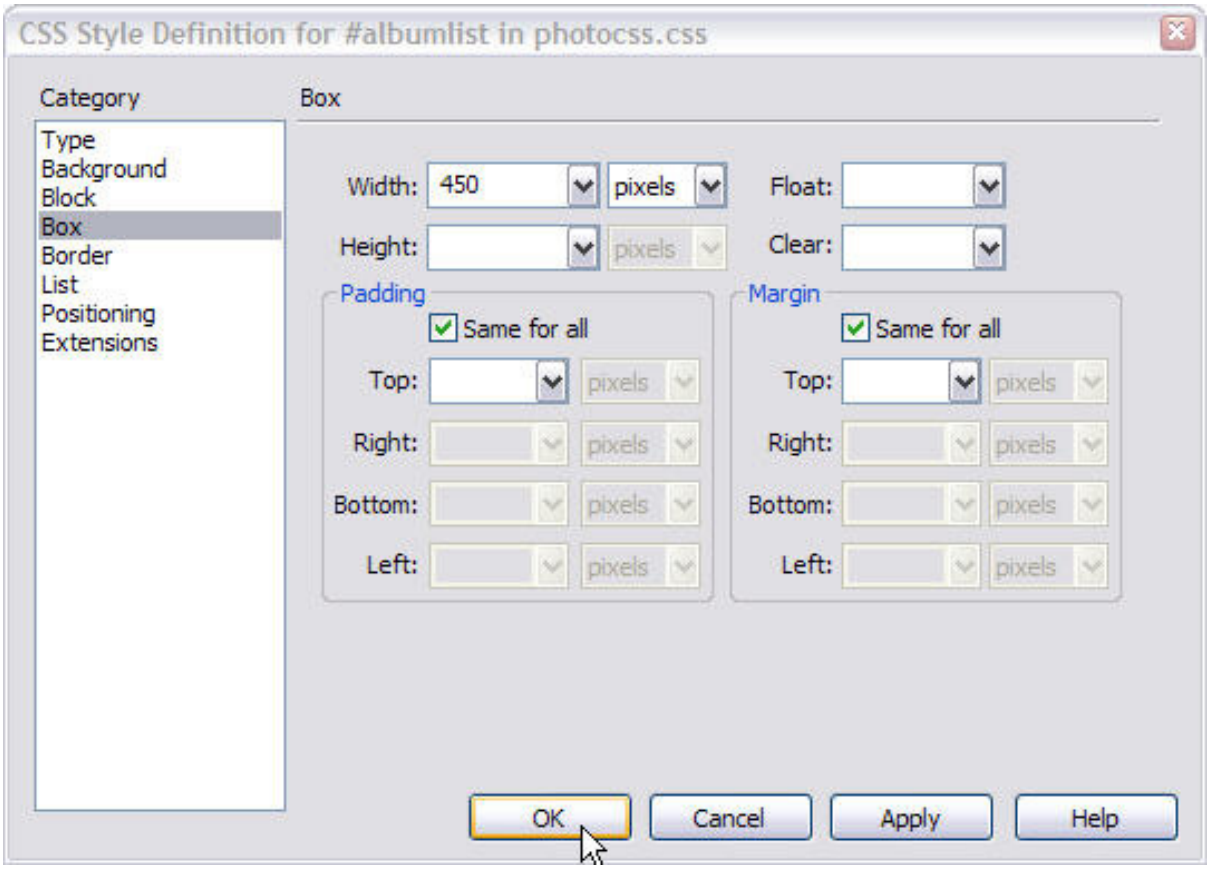
All we have added to the original document to create this effect is the application of the albumlist id to the ul tag.



The photo album as displayed in the browser

Setting the width of the layout

The fact that the images will wrap to the size of the screen may be a plus point of this method for some sites, however if you have a fixed width layout you may want to set the width of the album grid to a certain list. To do so simply set the width on #albumlist.

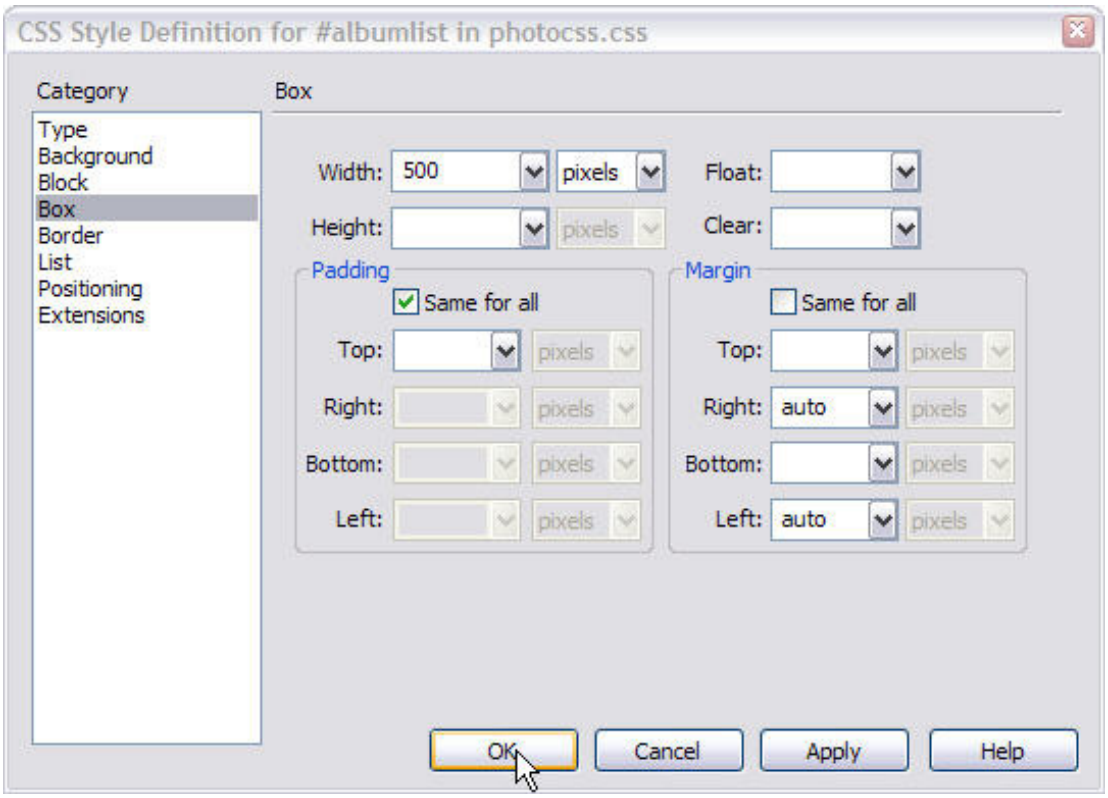


Setting the width of #albumlist

This will then force the photographs to stay within their container, at the width you have set. If the photographs are contained within another div that has a set width they should also honor that width setting and not expand outside of the bounds of the container.

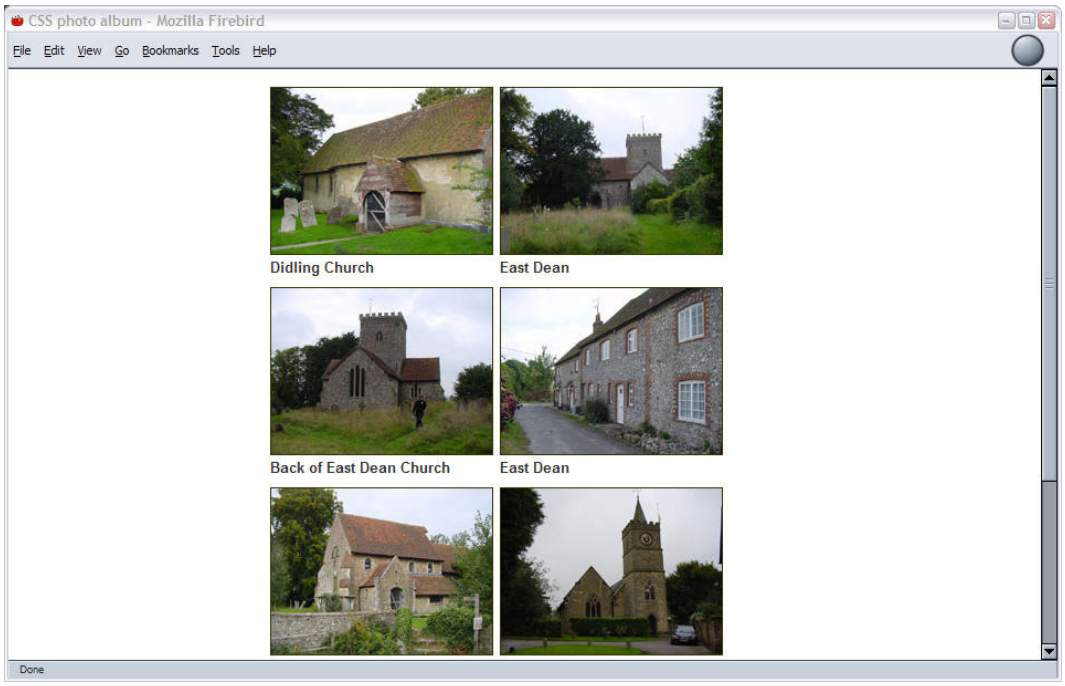
Centering the layout

You can center the layout by editing the CSS for #albumlist and setting **margin-right** to **auto** and **margin-left** to **auto** in the Box Category.



Centering the layout

If you have set the album to a fixed width of 450 pixels and centered it on the page, it will display like the image below in the browser.



Fixed width, centered photo album

You could lay out all kinds of page elements using this type of model. If you are not laying out pictures, but rather blocks of text, you will need to experiment with setting heights and widths on the list items in order that they remain with constant dimensions. The **float** property is very useful, and learning how to use it will make your CSS layouts far more interesting and flexible.

11. Centering Designs with CSS

One of the more frustrating aspects of CSS is centering our fixed designs effectively on the page. I know this because it's a question people ask me quite often—and I've lost count of the amount of times that I've seen this question asked on newsgroups and forums.

While CSS offers a logical means to center designs, the problem is with browser implementation. No surprises there! It's more than a bit frustrating because with table-based layouts this was a no-brainer. We'd simply use the **align="center"** attribute to center the containing table, and the entire layout would then be centered.

So how do we achieve the same effect in CSS? The good news is that we can. The bad news is that to center our content effectively, we have to employ a workaround in order to support multiple browsers.

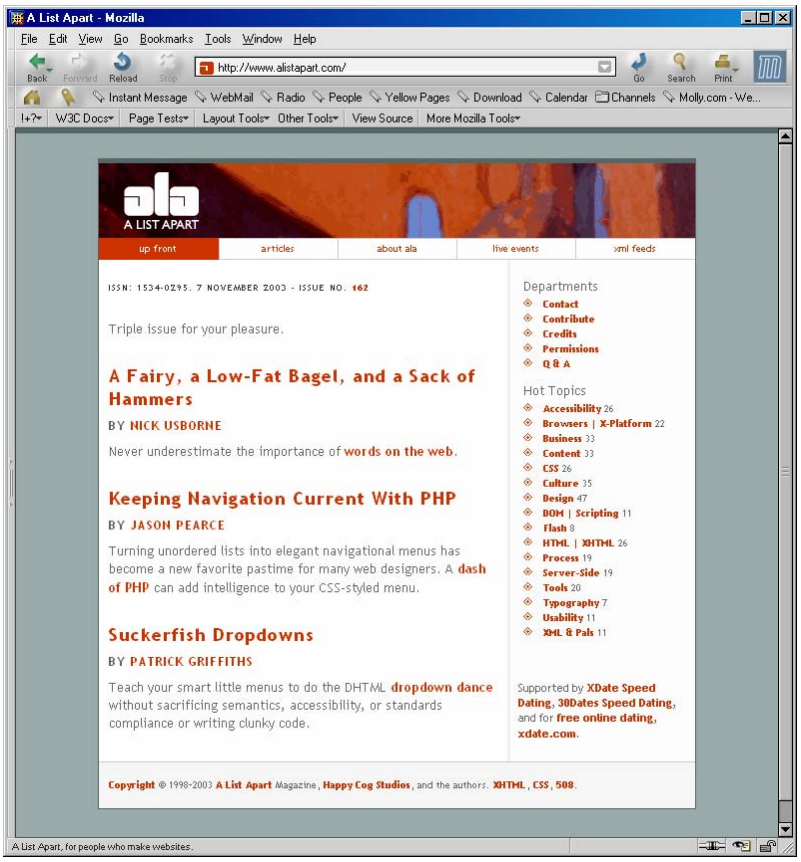
The Issue in Detail

Web design is as subject to the whims of the times as is fashion. At one point, even our table-based designs were fixed and aligned to the left. Then, as higher screen resolutions became commonplace, having our designs flush left with lots and lots of white space to the right became bothersome.

So, we began to center our fixed designs, balancing the white space more effectively.

Other camps began to strongly advocate "fluid" or "liquid" designs—those designs that flowed to fill the browser window. This technique became commonplace, but it, too, has two major flaws. First, liquid design means less control of the overall design, and second, managing text line lengths in a fluid design is difficult if not darned near impossible without sacrificing some other aspect of the design.

Now, even with CSS layouts, there are still folks thinking about fixed versus fluid designs. The new design at A List Apart, which is CSS-based, uses a fixed, centered design:



Simon Willison, a long time advocate of Web standards and a new member of the Web Standards Project, wrote on his weblog this very week "In recent months, I've been seriously reconsidering my preference for liquid layouts." You can check out the commentary and the responses at http://simon.incutio.com/archive/2003/11/09/. Interesting that this topic is being revisited at this stage of the game!

So, for those of you interested in designing fixed-width designs and having them centered, but using CSS to do it, this article is for you.

Centering the Right Way

The correct way to horizontally center a box (or "Layer" as Dreamweaver refers to positioned boxes) is to set the left and right margins of a given box to a width of "auto". This is going to be true of any box within your design that you'd like to center, not just the containing box we're using to center the design itself.

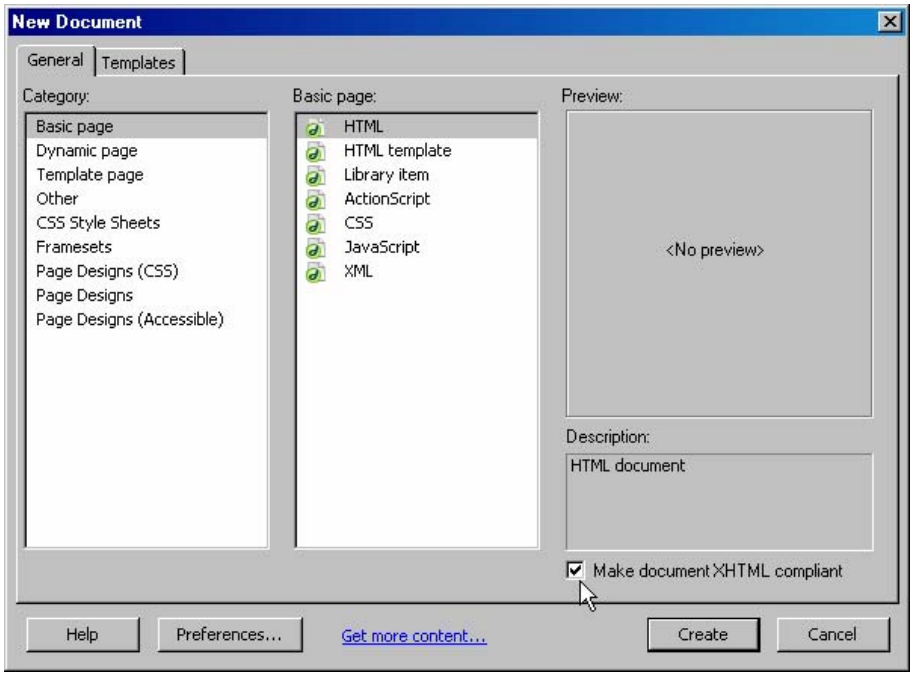
```
#mainbox {
     margin-left: auto;
     margin-right: auto;
}
```

By doing this, any box will neatly center, and in complying browsers, it does. In fact, let's go ahead and set up a box, then set if to use these properties and see what happens.

I'm using Dreamweaver MX 2004 for this, and I recommend you do, too. The reason is due to several markup fixes that the 2004 version offers, as described in just a bit. If you don't have MX 2004, you can download a trial version from the Macromedia web site. If you want to use MX instead, don't worry, I've provided details about how to manage the differences.

With Dreamweaver open and at the ready, follow these steps:

1. In Dreamweaver select File > New.
2. Be sure Basic page is highlighted, as is HTML.
3. Check the Make document XHTML compliant checkbox.



4. Click Create. Your XHTML file will be created.
5. Go ahead and save the document locally by selecting **File > Save As** and naming the file. I called mine **centering-auto.html**.
6. With the document open in Split view, go to the Code view panel and inside the body tags, type in the following code:

```
<div id="mainbox">

</div>
```

7. Go ahead and save your file.

If you're using Dreamweaver MX, and not 2004, it's possible that the XML prolog was added to the document, and that the namespace is not included in the opening <html> tag. You want to remove the prolog as it conflicts with the DOCTYPE switch in IE 6, which, as you'll see in just a few minutes, can interfere with proper interpretation of the Box Model, improperly displaying your box. So, look for a line of markup at the very top of your document that looks like this:

<?xml version="1.0" encoding="UTF-8" ?>

Simply highlight and delete it. Then replace the <html> tag with:

<html xmlns="http://www.w3.org/1999/xhtml">.

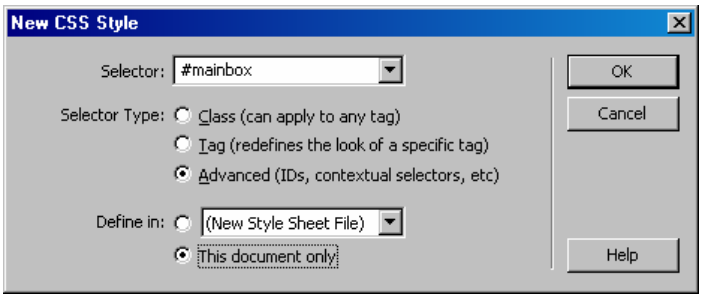
Once you've done that, you can save the file. All this fuss reveals exactly how understanding the languages with which we work and implementing them both as authors and software developers creating products for web design should pay attention to specifications—our lives will be so much easier as a result.

The basic markup we just created provides the structure for the division we're going to use as our main content box.

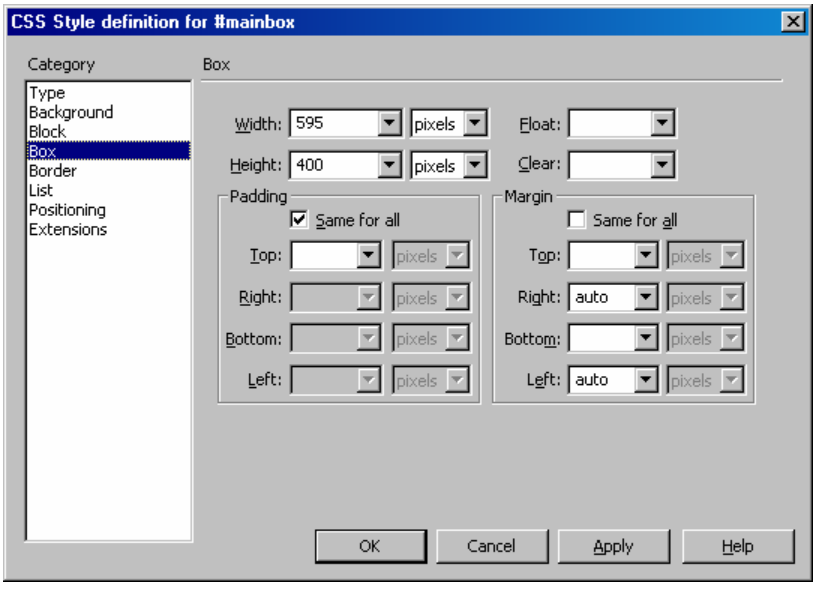
Note that I'm using an ID, meaning that I cannot apply this ID anywhere else in the document. IDs are purposely meant to uniquely identify one specific element within a page. This allows them to be accessed via scripting and IDs are frequently used for layout boxes because we only want one type of box per page such as a box for our header, footer, content, navigation and so on.

Now we'll style the box. To do so, follow these steps:

1. If your CSS Styles within the Design panel isn't open, select **Window > CSS Styles** to open the CSS Styles tab.
2. In the CSS Styles tab, click the New CSS Style Button. The new CSS Styles dialog appears.
3. In the Selector textbox, type in the ID name **#mainbox**.
4. Under Selector Type, click the Advanced radio button.
5. Choose This document only for the Define in option.

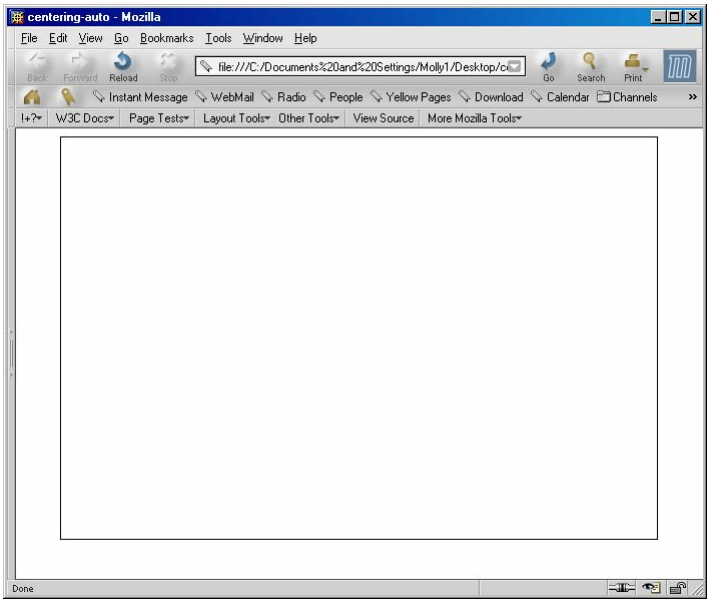


6. Click OK. The CSS Style Definition dialog appears.
7. Under Category, highlight Box. This brings up the properties available for the box.
8. In the Width textbox, enter a width of **595**.
9. In the Height textbox, enter a height of **400**. These are arbitrary numbers just for the purposes of this exercise. When you're creating your containing box, you'll follow the width your design requires, and you'll probably leave the height out as the content will determine the height.
10. Under Margin, uncheck the Same for all checkbox. This will make all the Margin options active.
11. In the textbox next to Right, choose auto from the drop-down menu.
12. In the textbox next to Left, choose auto from the drop-down.



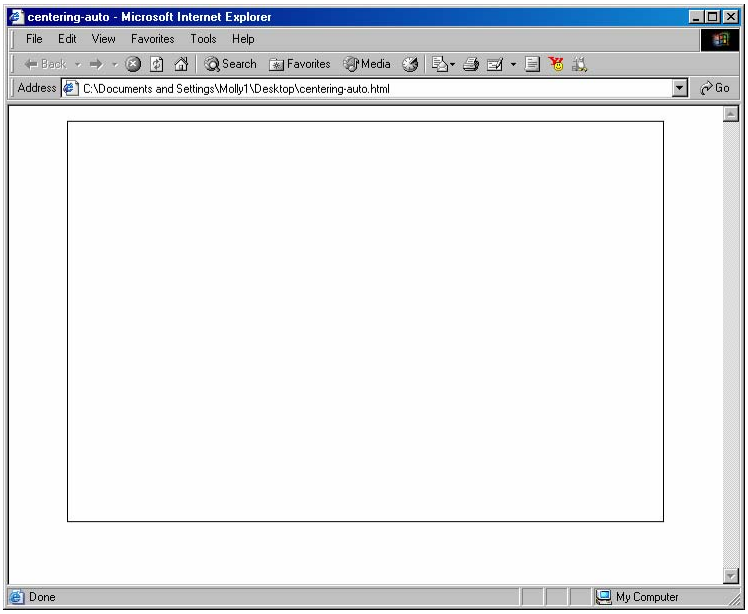
13. Move to the Border Category, and let's give the box a 1 pixel black solid border so we can see it as we work with it. Under Style, choose Solid from the drop-down menu, Under Width, enter 1 into the textbox, and under Color, choose black.
14. Click OK.

Your box is centered! Or is it? Well, if you look in Mozilla, it is:



Centred in Mozilla

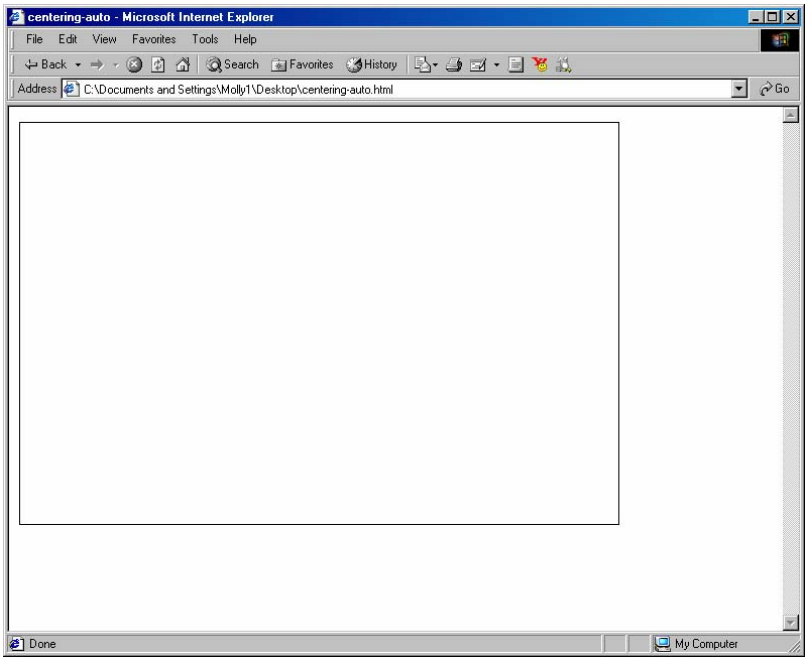
And if you look in IE 6 it is:



Centered in IE6

WARNING: If you take out the DOCTYPE declaration, the box in IE 6 will be left-aligned because without the DOCTYPE declaration in place, the browser operates in "quirks" mode rather than "standards" mode—something you can learn more about in the chapter "DOCTYPE SWITCHING IN MX" This is one reason I'm using Dreamweaver MX 2004—the DOCTYPEs in 2004 are all happily corrected and the basic templates do not generate the XML prolog—two nice improvements that the new version offers.

Opera 7 has no trouble properly rendering the centered box, either. But if you look in IE 5.0 (as shown here) or Netscape 4.x or many other browsers, the box is not centered:



CSS Centering Doesn't Work in IE 5.0

So, the conclusion here is that the correct way to center your containing box may not provide you with the backward compatibility you require.

Centering the Wrong Way

So, as the sorry state of browser affairs may disallow us to use the correct CSS to manage our containing boxes, we have to look at workarounds or hacks. The most common one in use is to apply the text-align property to the document body, setting the alignment to a value of "center". This technique will work in a wider range of browsers, and will even work in cases where you don't have a proper DOCTYPE (but of course you should).

The workaround is controversial for a number of reasons, at least to hardcore markup junkies:

- We're applying a property meant for text, not for positioning, so we're not using the language correctly, which is why this is considered a hack.

- We're going to have to write extra CSS to override **the text-align: center** property, because we're going to be using the text-align: center property on the body element. So, any text within the body is now going to be—guess what—aligned to the center.

To apply the text-align hack for centering, follow these steps in Dreamweaver:

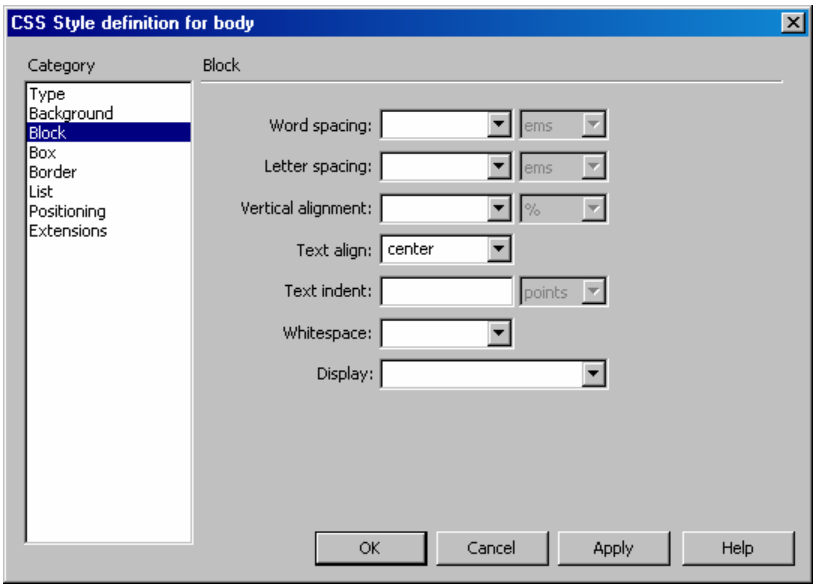
1. Select File > New. Choose a Basic HTML page and check the Make document XHTML compliant checkbox.
2. Click Create.
3. Add the same markup we used to create the box in the last exercise within the body tags:

```
<div id="mainbox">

</div>
```

4. Save your file, I named this one **centering-hack.html**.

With your XHTML document at the ready, let's go ahead and add the box and the hack. To do so, follow these steps:

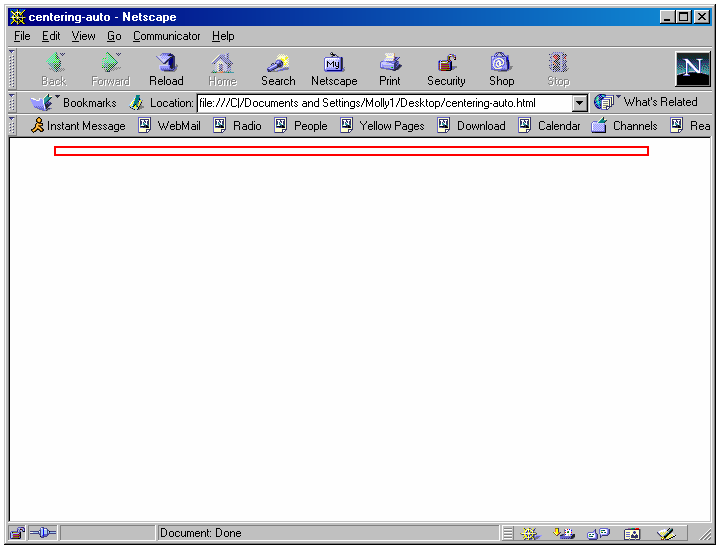
1. From the CSS Styles tab within the Design panel, click the New CSS Style button.
2. In the New CSS Style dialog, add **#mainbox** to the Selector field.
3. Click Advanced for Selector Type.
4. Click This document only for the Define in option.
5. Click OK. The CSS Style Definition dialog appears.
6. Now go to the Box Category, and set up your main box with a width of **595** and a height of **400.** Leave the Padding and Margin options as is.
7. Add a border using the Border Category. Let's have some fun this time, I made my border dashed, 2 pixels wide, and red.
8. Click OK.
9. Click the New CSS Style button again.
10. In the New CSS Style dialog, add **body** to the Selector field.



11. Click Tag for Selector Type.
12. Click This document only for the Define in option.
13. Click OK.
14. In the CSS Style Definition dialog, highlight the block category.
15. For the Text align option, choose **center** from the drop-down menu.
16. Click OK.

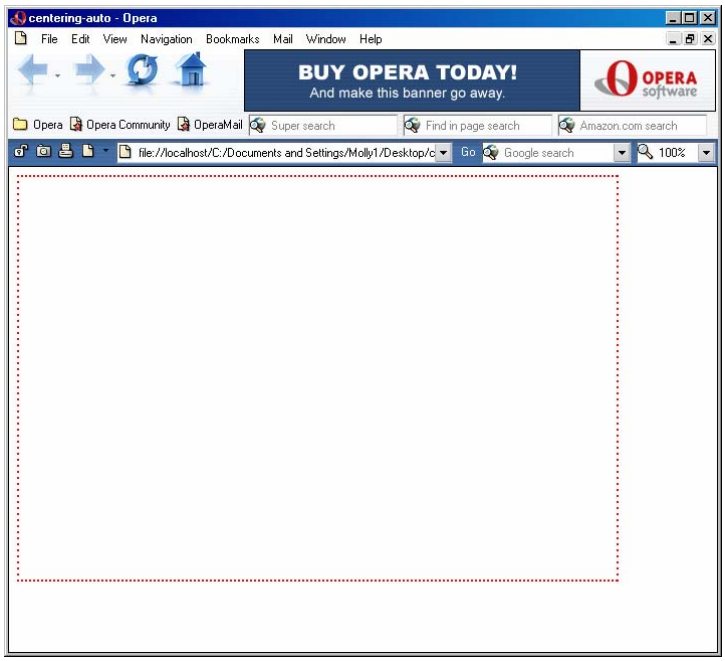
Your box is now centered! Or is it?

In IE 6 it is. In IE 6 even without the DOCTYPE declaration it is. In Netscape 4.x it is (even though the height and border styles aren't supported):



Centering Hack in Mozilla doesn't work.

But in Mozilla and Opera (shown here), the box isn't centered. That's not a fault of those browsers by the way, it's because *they're rendering the CSS properly*. We're using a hack, remember?



Centering Hack in Opera doesn't work.

So now what?

One Right and One Wrong Equals Compatibility

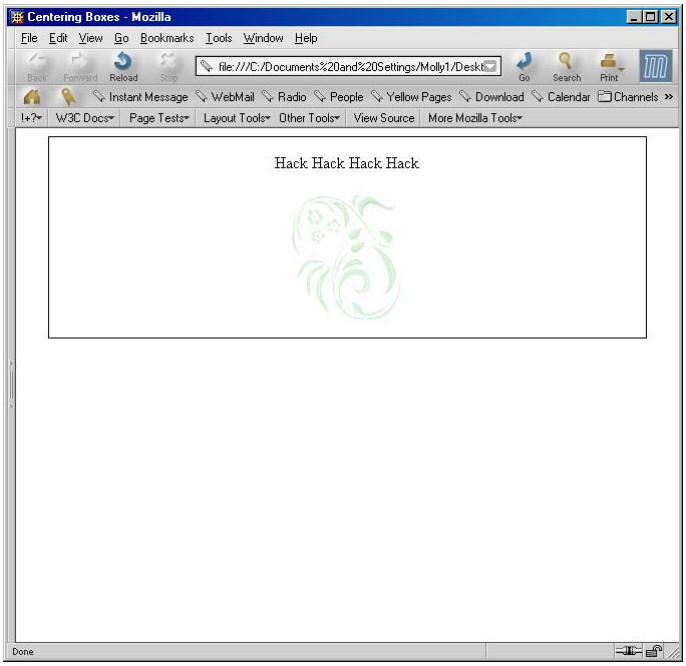
To make your containing box as cross-browser compatible as possible, you have to use both the right way *and* the hack. To do this, simply go ahead and copy the style for **body** from the centering-hack.html document and paste it into the centering-auto.html document, saving that document as centering-combined.html.

Here's what the markup for the workaround looks like:

```
<!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>
<title>Centering Boxes: The Right Way</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>

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

```
#mainbox {
     width: 595px;
     height: 200px;
     border: 1px solid black;
     margin-right: auto;
     margin-left: auto;
}

body {
     text-align: center;
}
</style>

</head>

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

But now, of course, everything in the document is going to be centered. Here, I've added some content without styling it:



Everything gets centered.

As a result, you'll have to be very careful to now create styles for your paragraphs, headers, and other content to ensure everything is aligned properly.

The Good News

Fortunately, even though using a hack is an ugly thing by nature, this hack isn't terribly harmful and we're able to get the effect we're after—a design that is neatly centered on the page. What's more, the compatibility factor at this point is very high, we're able to display this in all kinds of browsers, including Netscape 4.x, which is pretty cool.

12. Styling forms with CSS

In this chapter, we will look at the variety of ways in which we can make attractive forms for our sites and web applications by using CSS. Forms are an important but unglamorous part of many sites and for web application design, much of the client-side development can be taken up with creating usable forms. CSS makes it easier to create attractive forms quickly, because once you have developed a default look for forms then you can reproduce that for every form on the site.

Styling form elements – what can we change?

Using CSS , we can change many things about the appearance of our forms and the elements within them - either by redefining the html tag that controls these elements, or by applying custom classes to them.

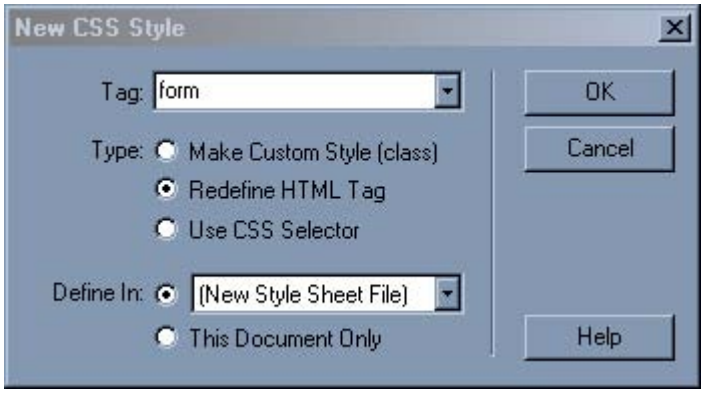
The form tag

In a new document in Dreamweaver, insert a form using the Insert toolbar. By default, a form doesn't render in any visible way in the web browser – in the Design View in Dreamweaver it displays with a red dashed line so that you can see where the form is on your page.

Creating a style that will apply to all forms

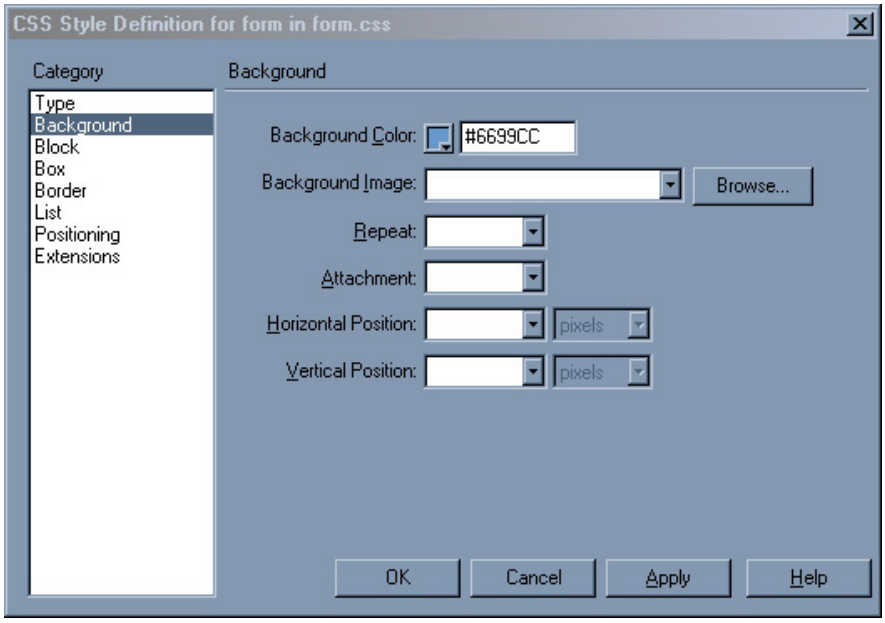
If you want to create a style that will apply to all forms on any pages that have this style sheet linked to it, create a new CSS Style (in the CSS panel click the **New CSS Style**) and choose to 'Redefine HTML Tag'.

Select '**form'** in the selection box for Tag. If this is a new document then define this in a new stylesheet file – in a site in progress you can choose to add them to the existing style sheet for the site.



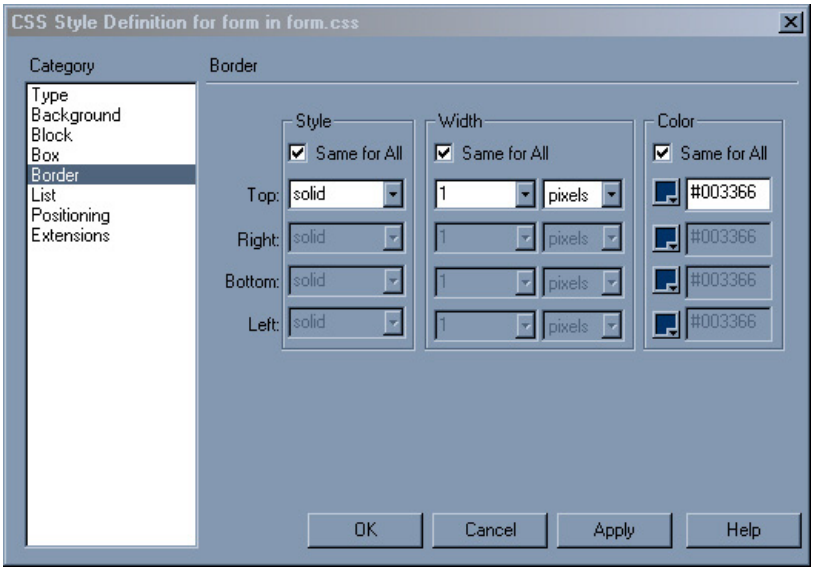
Redefining the form tag

Click **OK** and in the CSS Style Definition dialog set a background color for the form.



Setting the background color

And a border.



Setting the border

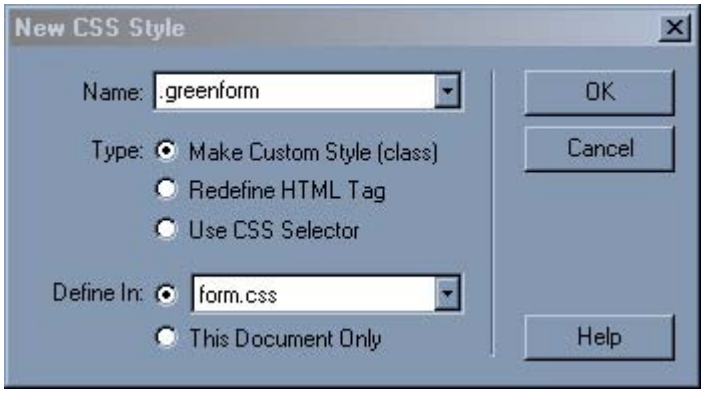
Click **OK**. In Dreamweaver you should see your form change to reflect the styles for the form.

To see anything other than a thin line in the web browser you will need to add some text to the form before previewing it.

The form in a web browser

Creating a style that will apply only to certain forms

If you want to create several different form styles for your site – perhaps one style for a login form, another for a longer, complex questionnaire and yet another for your contact form then you will need to use custom classes.
Using the same page and form create a new CSS Style, this time select to **Make Custom Style (Class)** and name it '.**greenform**'.



Make Custom Style

Click OK and then set a green background and darker green border for this class. Click OK out of the CSS Style Definition Dialog and return to the Design View of Dreamweaver. Your form will still be blue as it is still taking the general form definition. To change it to use the special form definition '.**greenform**' select your form. In the CSS Styles Panel switch the radio buttons at the top of the panel to 'Apply Styles'.

Then, with your form still selected click on the class name '**greenform**'. You should see your form take on the colors of this class in the Design View Window.

Applying a Style using the CSS Styles Panel

The CSS we have created

If you double click on the CSS style sheet name in the Site Panel it will open up in Dreamweaver and you can see the CSS that we have just created:

```
form {
     background-color: #6699CC;
     border: 1px solid #003366;
}
.greenform {
     background-color: #009966;
     border: 1px solid #003300;
}
```

form (note no period before the word) is the rules that apply to all forms on the site

.greenform (note the period that shows it is a class) is the special green form style that we have created.

Switch to Code View and you can see how this relates to our form. The form currently has the class greenform applied to it:

```
<form action="" method="post" name="form1" class="greenform" id="form1">
  <p>This is my form</p>
</form>
```

To return to the generic form style you can simply remove that class:

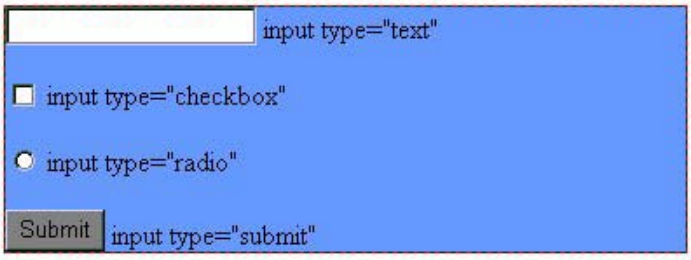
```
<form action="" method="post" name="form1" id="form1">
  <p>This is my form</p>
</form>
```

The input tag

Many form elements – text, radiobuttons, checkboxes, buttons – are all types of the input tag. Therefore if you simply redefine the input tag you will affect all of these different elements.

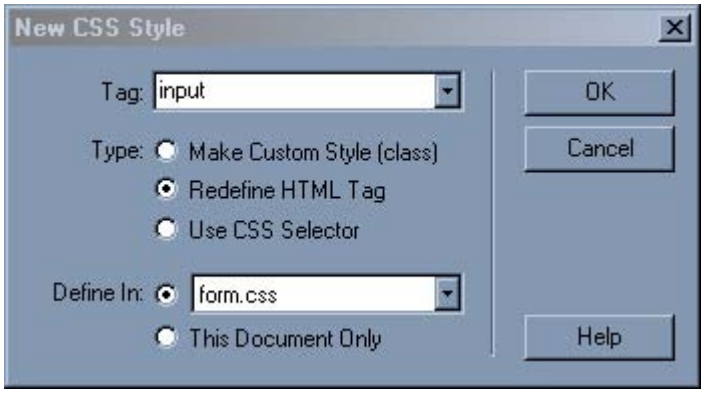
Redefining the input tag

As an example of what happens when you redefine the input tag, take the following form elements inserted into our generic form style.



Form with various elements in Dreamweaver MX

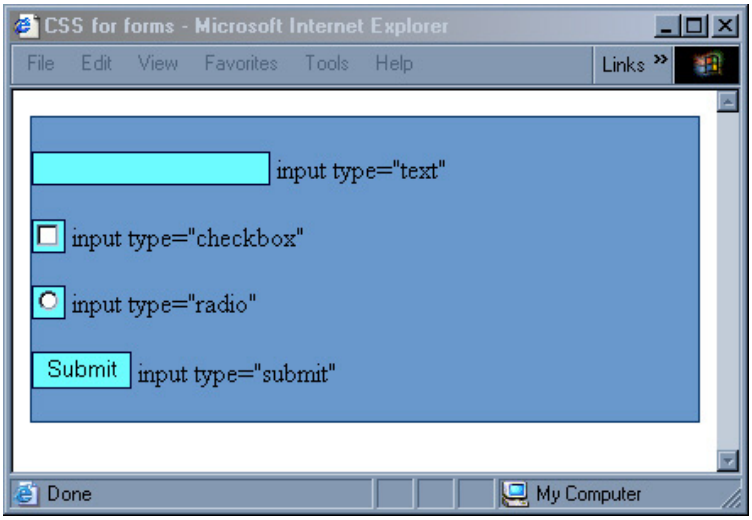
Create a New CSS Style and select to redefine the input tag.



Redefining the input tag

Give the input tag a background color (a different one to the background color of the form) and a border, then click OK.

These changes will not show up in Dreamweaver - as it does not render styles on form elements - so you will need to preview in a browser to see the results.



Form with styled input tag

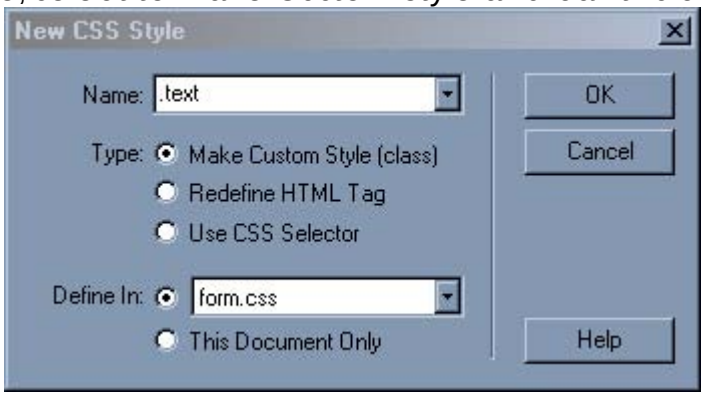
As you can see, styling the input tag will apply the same style to all of these elements – if you only have a very simple form on your site then this may be all you need, however it is likely that you would rather have some more control over the individual elements and so, once again, to do this custom classes are needed.

Creating classes for the input tag

When I start work on a new site, I usually create my generic form classes right at the start. That way, I can always apply them to the new forms that I create as I create them and then if I decide later in the development that I want the form to look slightly different I only need to change the style sheet and not revisit each form. At this point I will normally create classes for text, smalltext and buttons.

Text input

Create a New CSS Style, select to Make Custom Style and call this class '**.text**'.

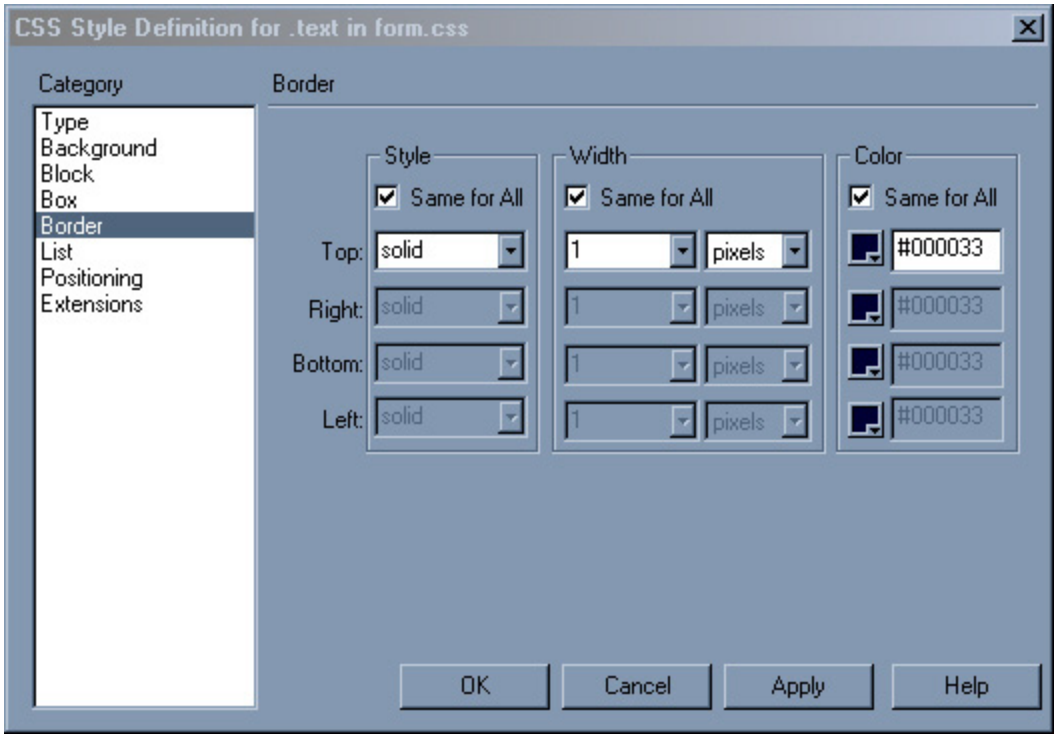


Creating a text input class

Click OK.

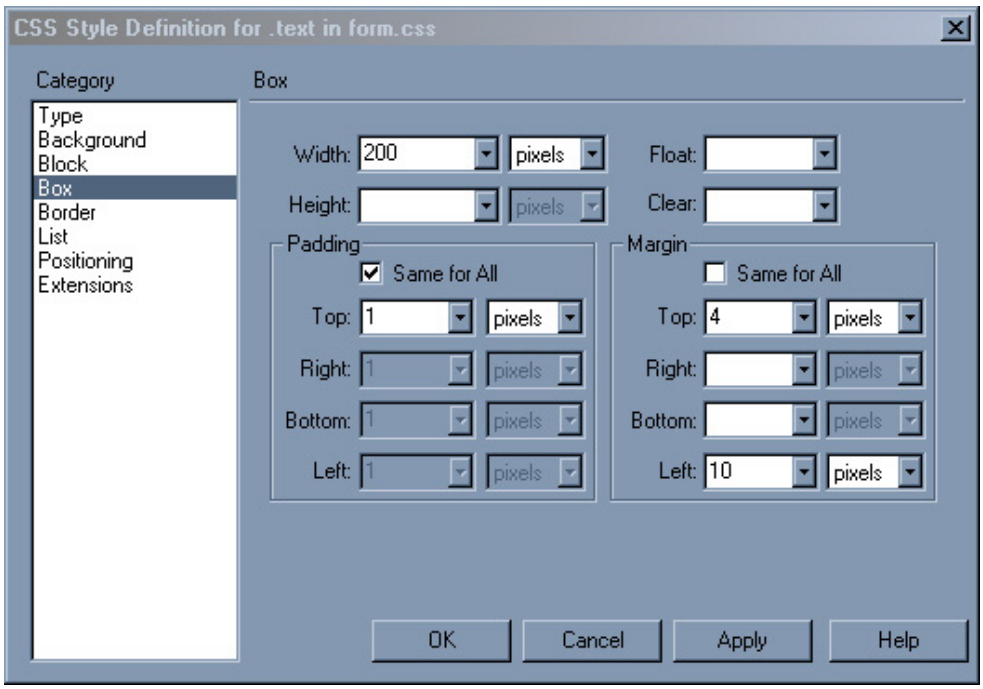
You can change most things about text fields:

- give them a border;



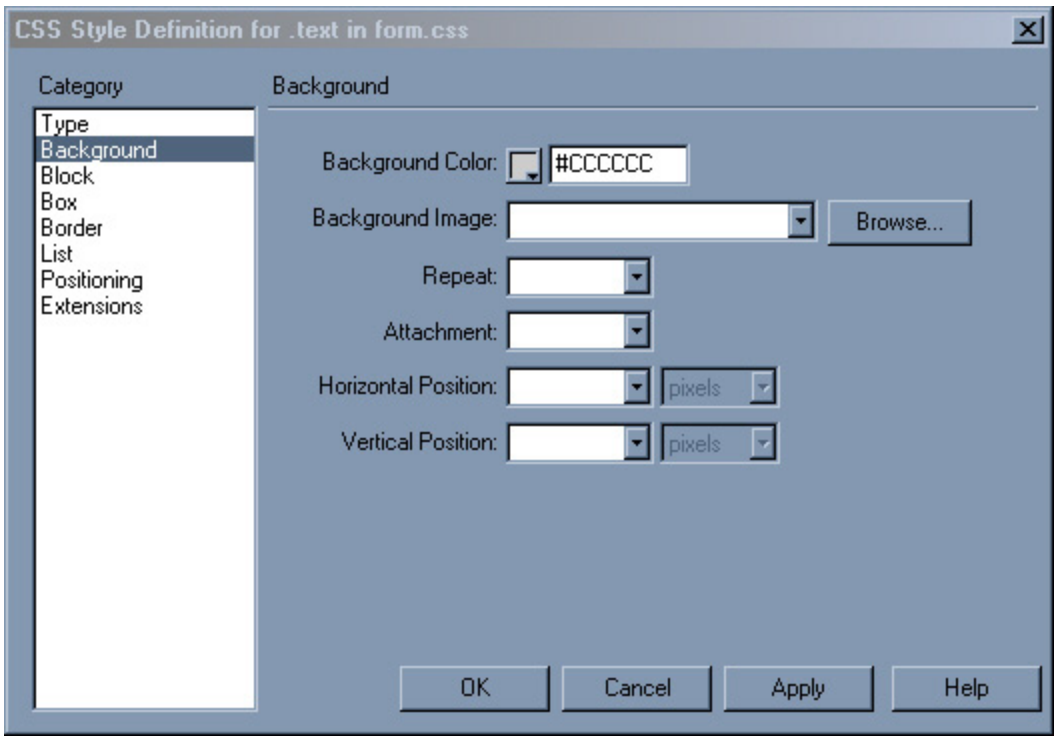
Setting the border

- set the width, the padding (space inside) and margin (space outside the field);

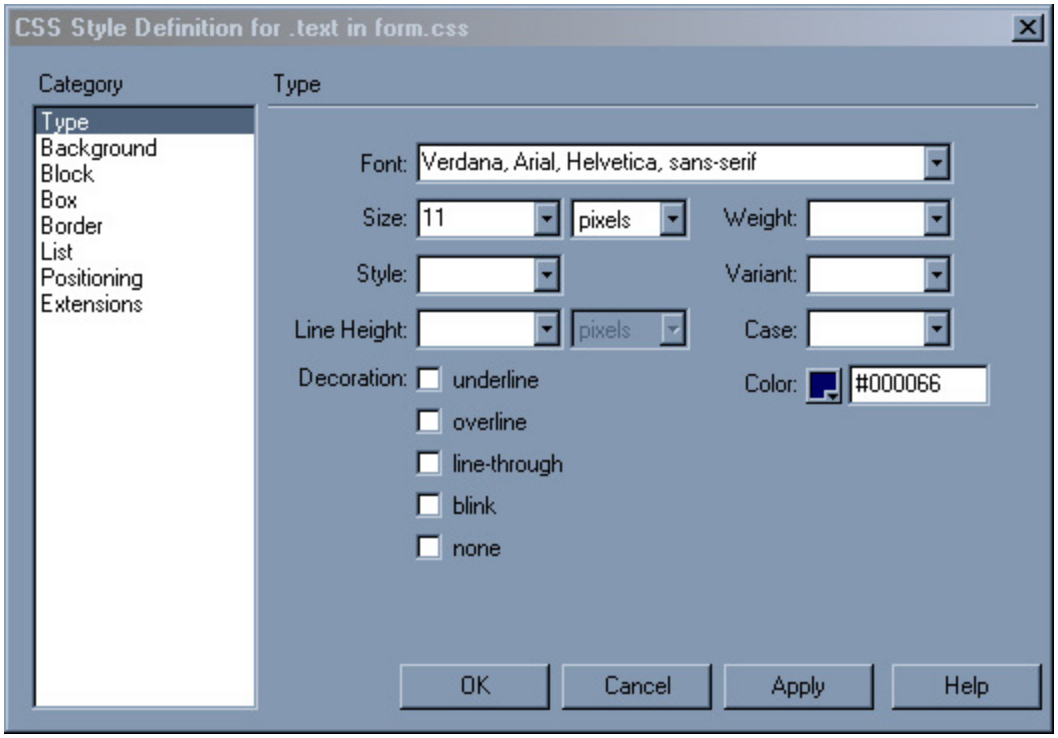


Setting width, padding and margins in the Box category

- change the background color;



Setting the background color change the color, font and size of the text that is entered when someone completes the field.

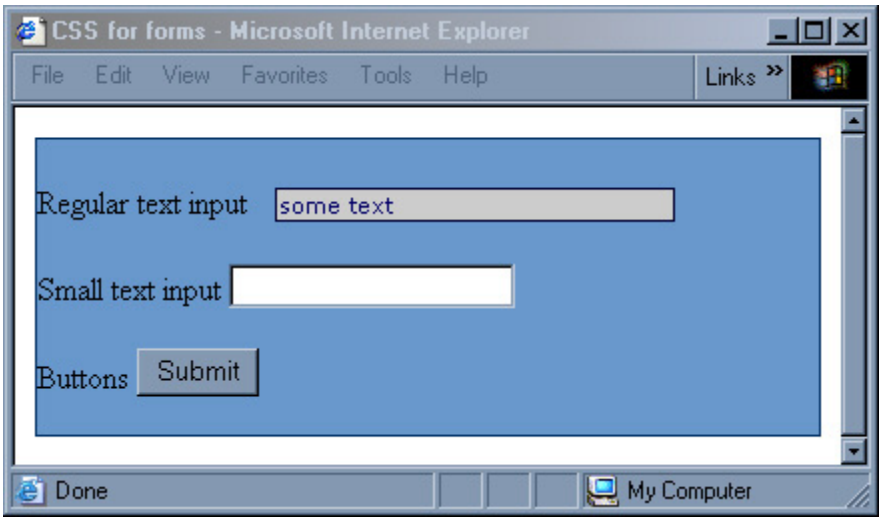


Setting the type

From a usability point of view you should always bear in mind that people will need to be able to complete the fields – tiny boxes with minute text may look nice but can be difficult to complete, particularly where you are asking people to add a lot of information.

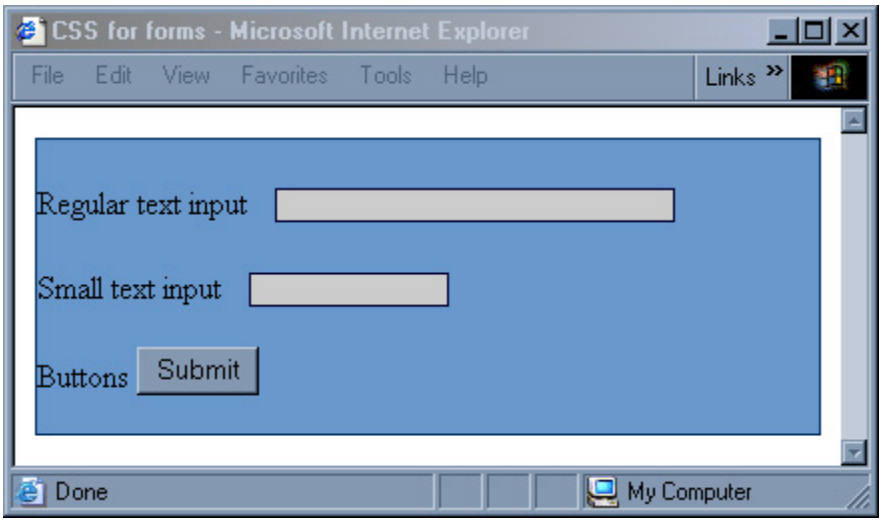
After setting the styles for this form, click OK. You will then need to apply the class to your default text field.

Dreamweaver will display the width changes but to see all of your changes preview in the browser.



Text field with a class applied

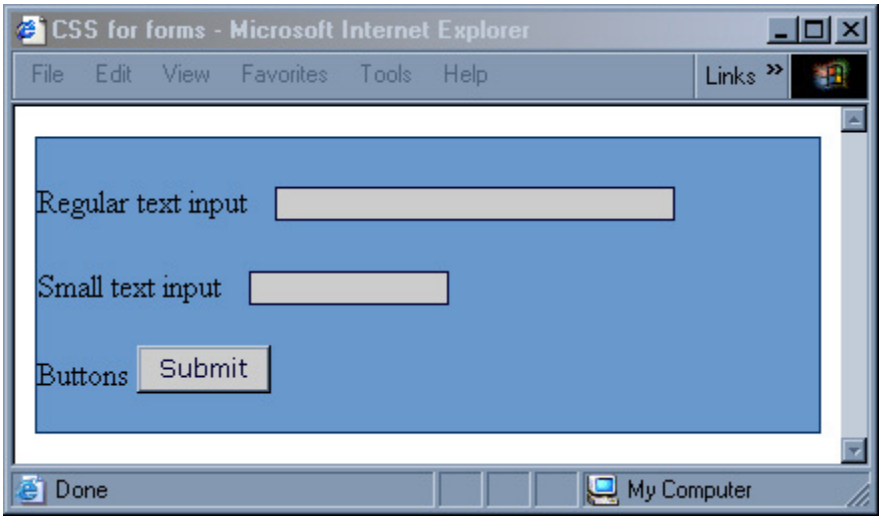
The small text input would be used for anywhere you don't want a great big field – for instance where you are asking people to type in a short password or their Zip Code. For this field I would use the same styles apart from the width, which I would set to half of the value of the original field – in this case 100 pixels. This class I call '.smalltext' and here it is applied to a field in the form:



The form now with classes for text and smalltext

Buttons

You can change all of the elements of your buttons as well, when doing this however, make sure that your users are still aware that this is the button! It is possible to make a button look just like normal text by setting borders to 0, and the text to look like the body text of your page, however people are used to seeing the default button 'look' so be very careful when changing the button style too extensively. I have simply created a class called '**.button**' to make the text slightly smaller and given the button a background color and color to fit in with the rest of my form.



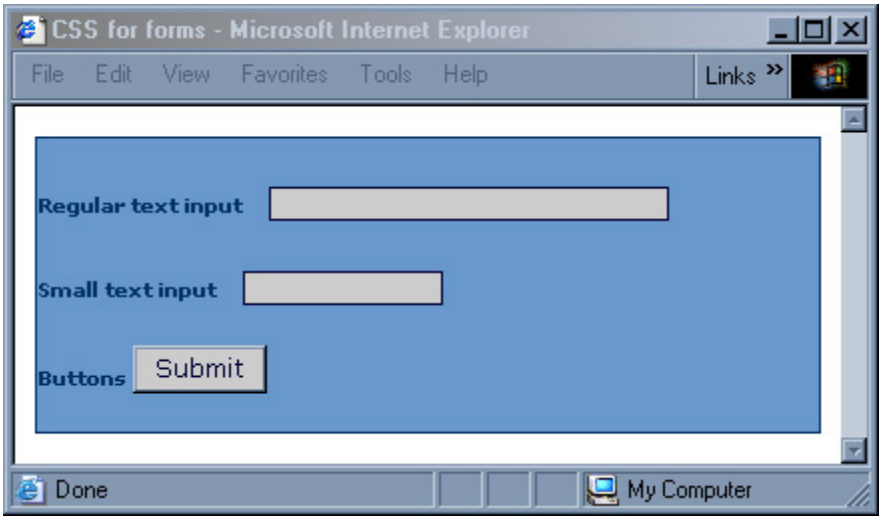
The form with the button class applied

Styling the label tag

As I have used the label tag for the labels to the left of these form elements, I can style this tag by redefining it.

Redefining the label tag

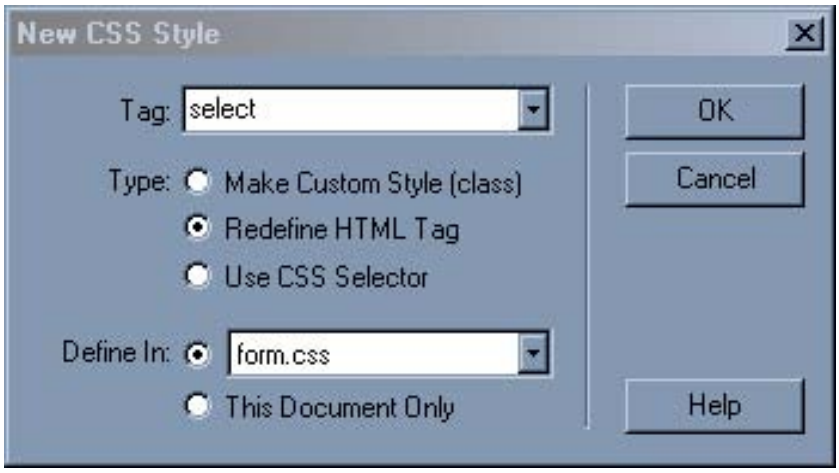
I have simply set some styles for the type of any text that is marked up with this tag, and my form now looks like this:



The form after redefining the label tag

Select menus

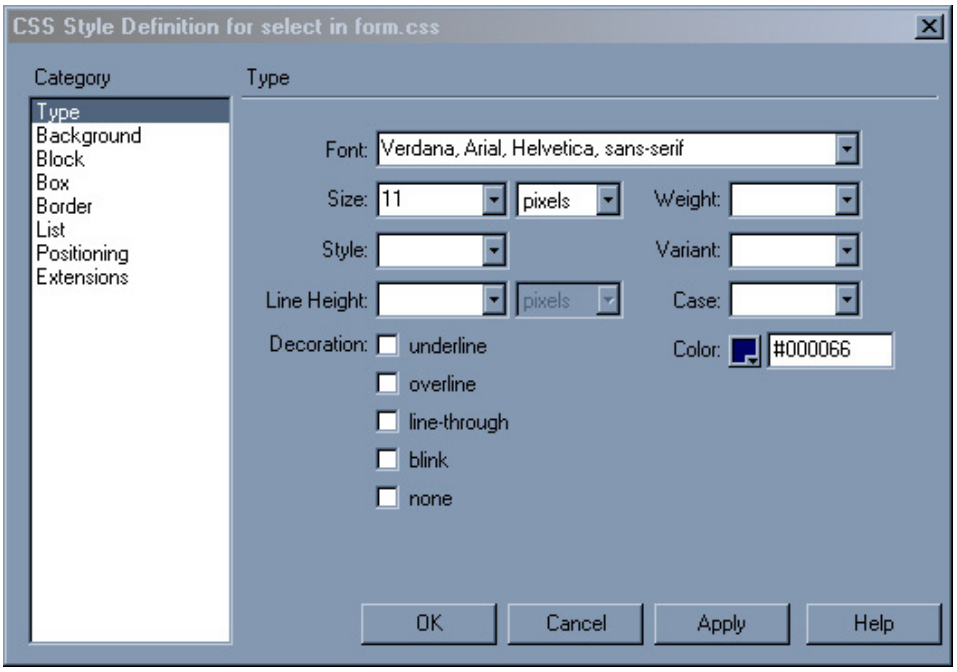
You can also change the text and background color of select menus. Once again you can achieve this by redefining the select tag or by creating a custom class. As there are fewer things that you can alter about a select menu then simply redefining the tag is appropriate in a lot of cases.



Redefining the select tag

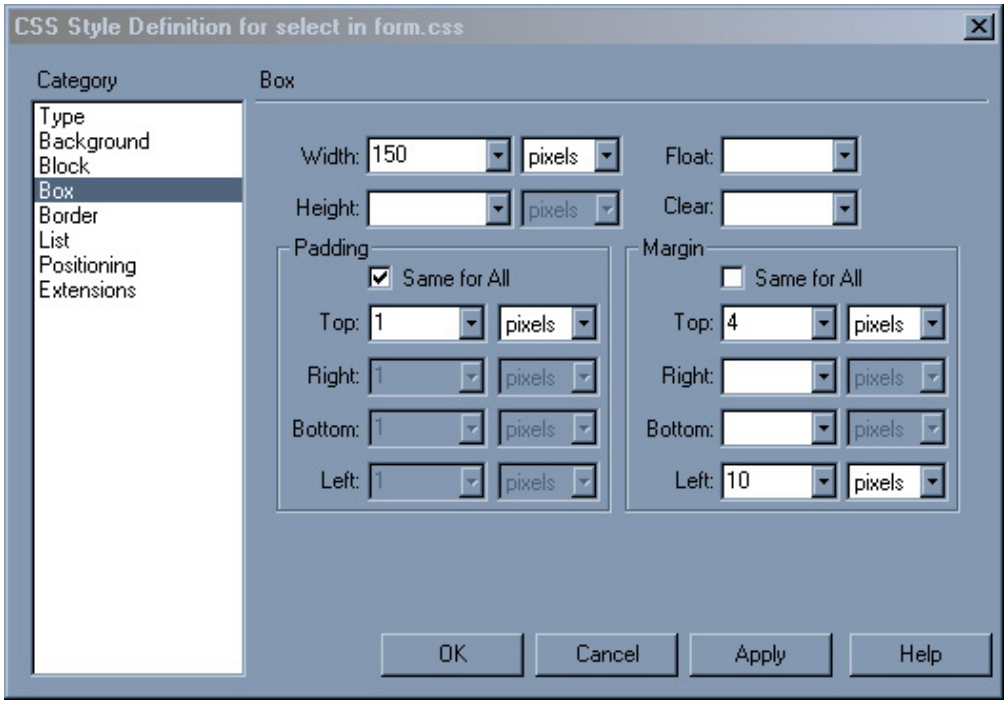
For the select tag, changes that will display in browsers include:

- Changing the type;



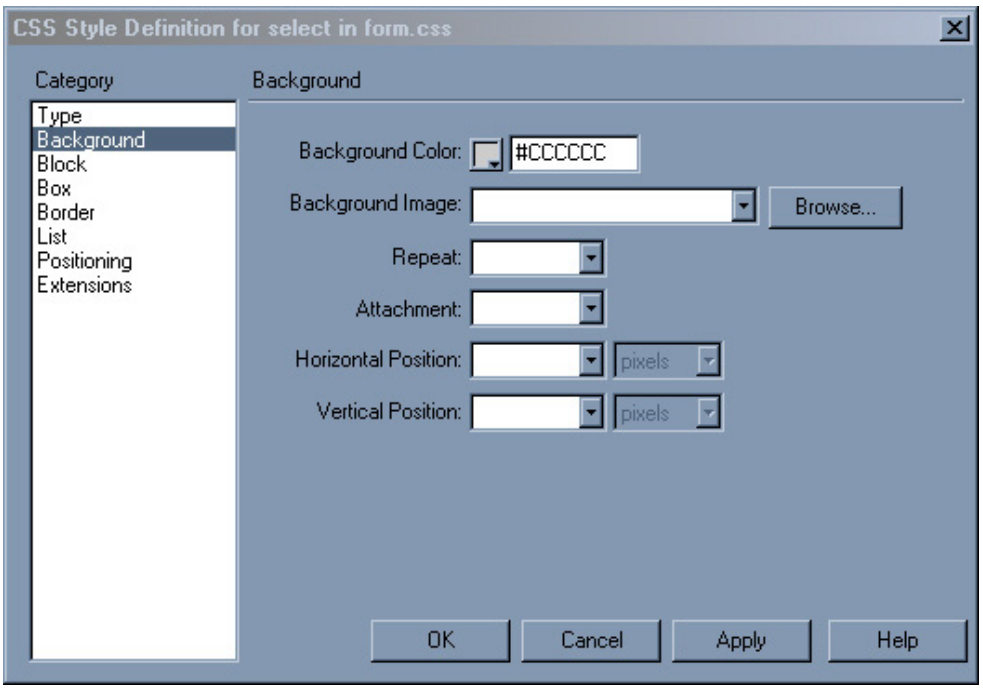
Setting the type for the select tag

- Setting a width, padding and margin;

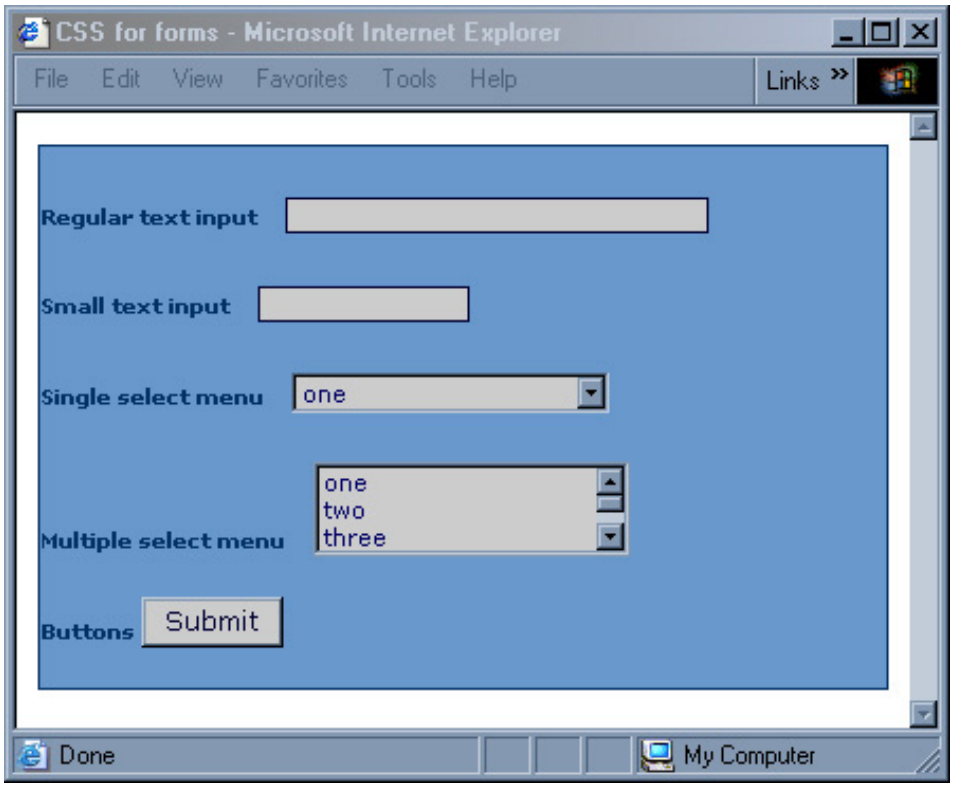


Setting width, padding and margin for the input tag

- Setting a background color



Setting the background color for the input tag

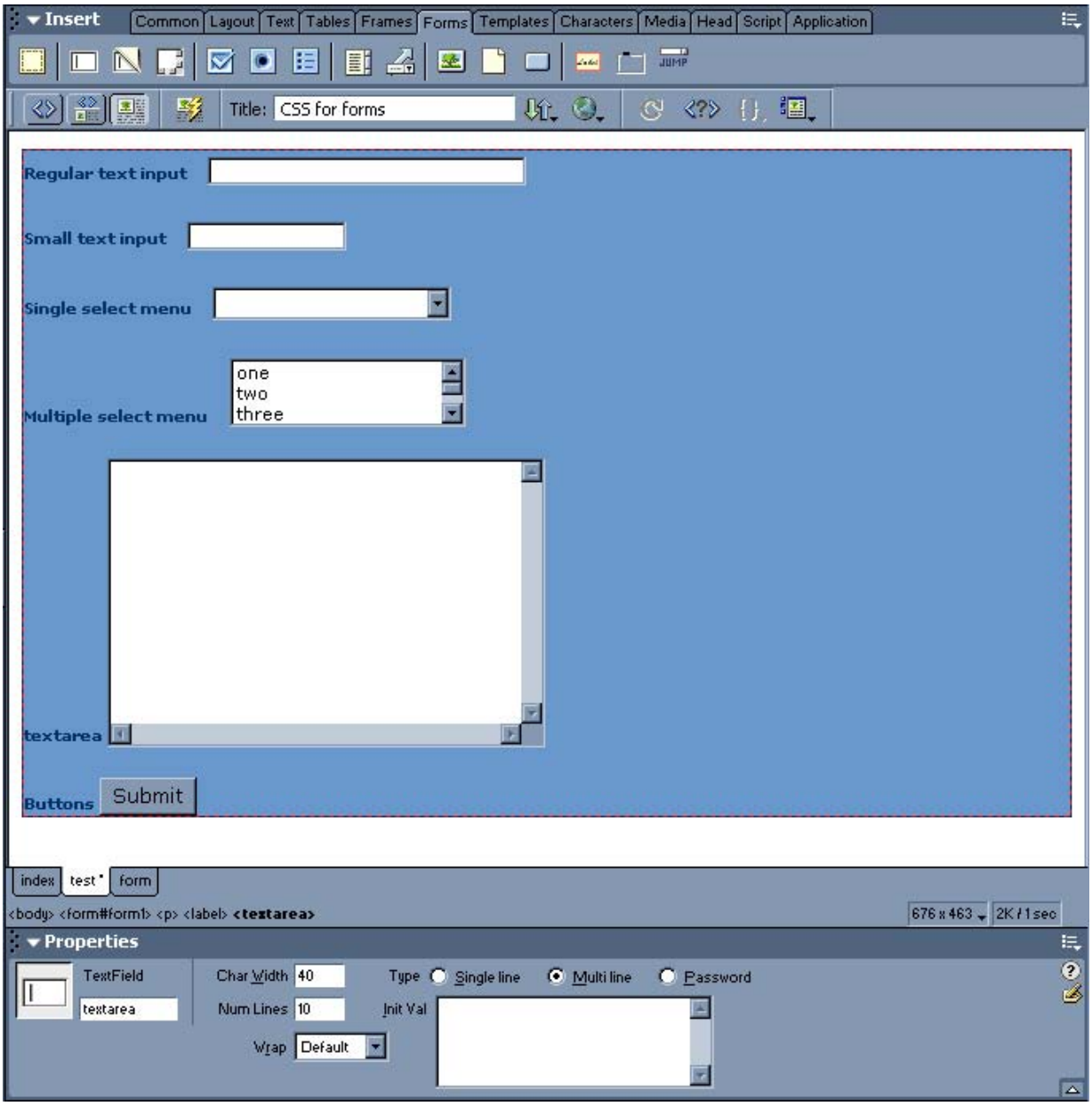


The form after adding select elements and redefining the select tag

You can, of course, create custom classes for these elements in the same way that we created the classes for the input tag.

Textarea

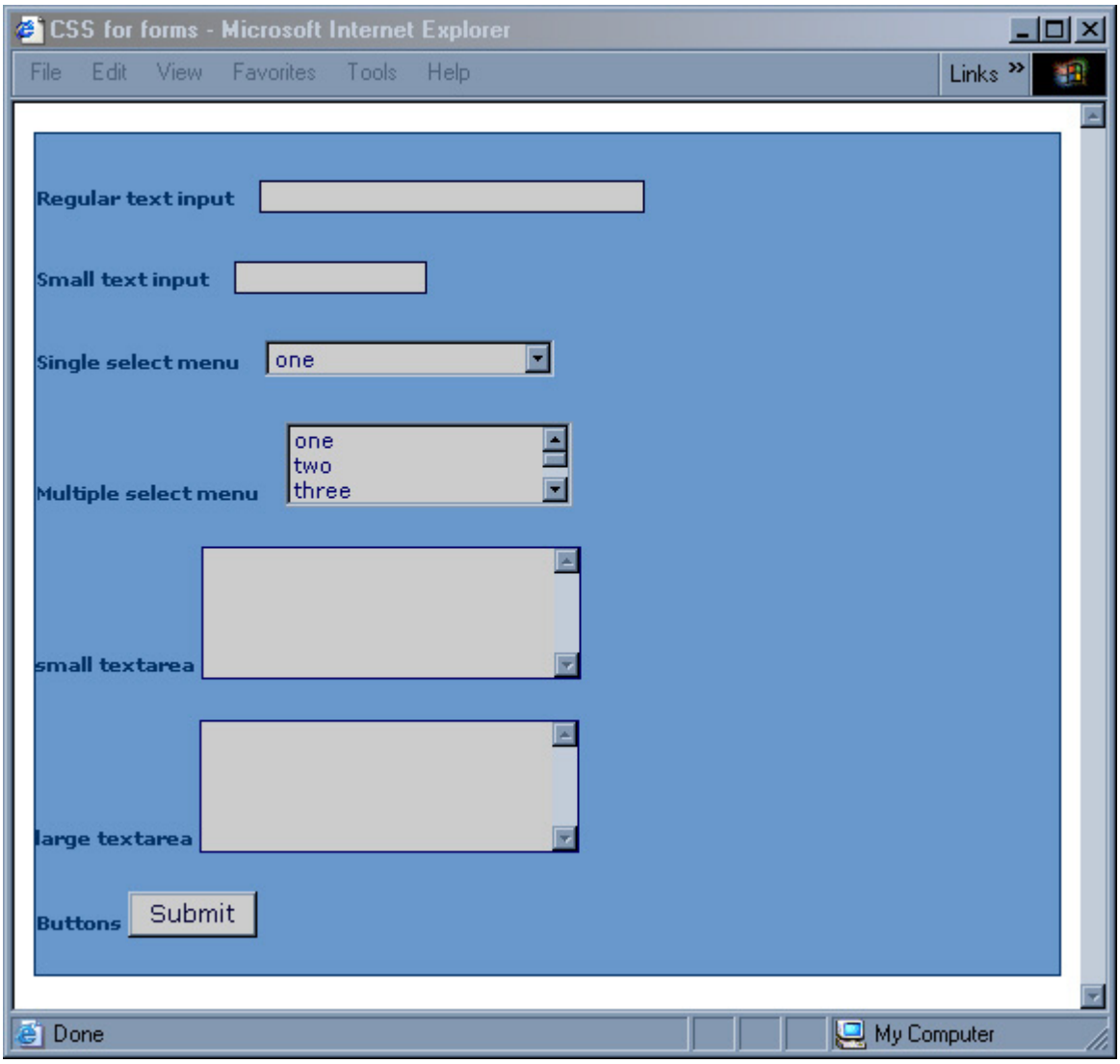
Textareas – large text input elements can also be styled in much the same way, in addition to the changes that you can make to a regular text field which uses the input tag, you can set the height of a textarea. Although we can change the height and width of a textarea using CSS, valid (x)html requires that you also set the size of it in the html using cols and rows. You set the size in Dreamweaver by selecting the element and then setting the properties in the Property Inspector.



Setting Char Width (cols) and Num Lines (rows) of a textarea in Dreamweaver

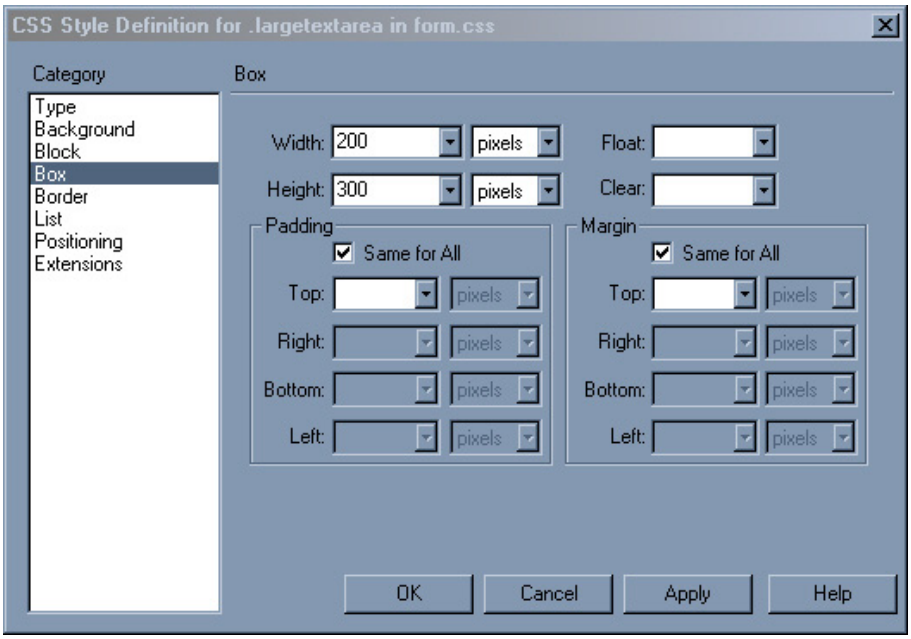
You can then redefine the textarea element and/or create custom classes for it. You will often find that you need different sizes of textarea. If all of them are going to look the same but you need two different sizes, you can combine redefining textarea with creating custom classes to save you having to enter all of the color and type information for each field.

First, redefine textarea – set the background color, color and type just as we did for text.



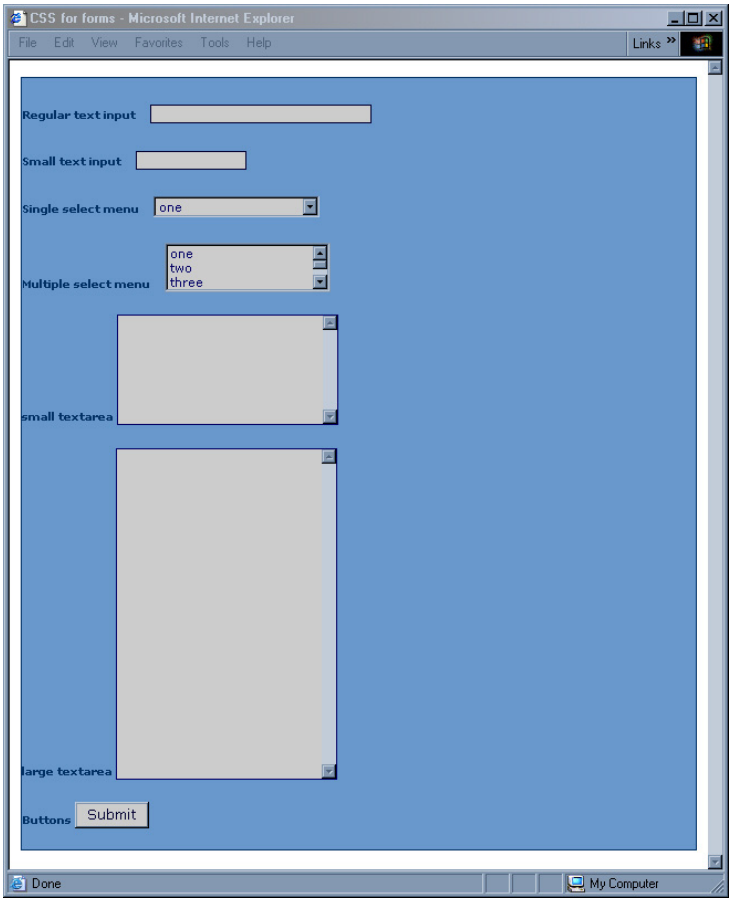
The form after redefining textarea

Now create two custom classes – one named '.**smalltextarea**' the other '.**largetextarea**'. Give them different heights and/or widths. I have set both to be 200px wide but give the small a height of 100px and the large a height of 300px.



Setting the height and width for .largetextarea

Then apply these classes to your textarea elements:



Form with the two textarea classes applied

What about old browsers?

Form elements styled with CSS are admittedly a problem when it comes to everyone's favorite old browser – Netscape 4.*. Even with a site laid out using tables, if you use form elements styled with CSS then you need to be aware that Netscape 4 doesn't support this styling and may display the form in such a way as to make the form unusable.

The simplest way around this problem is to add your forms stylesheet using the **@import** method that I describe in the 'old browsers' chapter. If the rest of your CSS works fine for Netscape 4, then you could simply have a separate style sheet that is just for the form elements. Then attach it to your page using the @import method. To do this in Dreamweaver, select the Attach Style Sheet button on the CSS Panel and instead of selecting the default **Link** radio button on the Link External Style Sheet dialog, select **import**.

Attaching a style sheet using the @import method

As the styles would not show up for Netscape 4 anyway, using this method will ensure that users of this browser will be able to complete the form as the elements will simply display in their default manner.

13. Using Design-Time Stylesheets to Create a Print Stylesheet

Despite the promised utopia of a paperless office, you are likely to find that users will print out your web pages for reference, or to read away from the computer. As we know, what looks good on the web does not necessarily look good in print and if your site uses many graphics, the user is going to be using up much unnecessary printer ink in getting their copy, as your navigation buttons aren't of much interest once the application is printed!

Many sites link to 'printer friendly' versions of their pages. These versions usually are separate versions of the document, created either by hand (which means you have to maintain 2 versions of the document) or by a script, and the printable document will be formatted for print and contain no navigation or irrelevant graphics. While this method works well, you probably don't have the additional development time of creating the new pages or writing the script to create the printable page, and you need to have a link on each page that launches this special version.

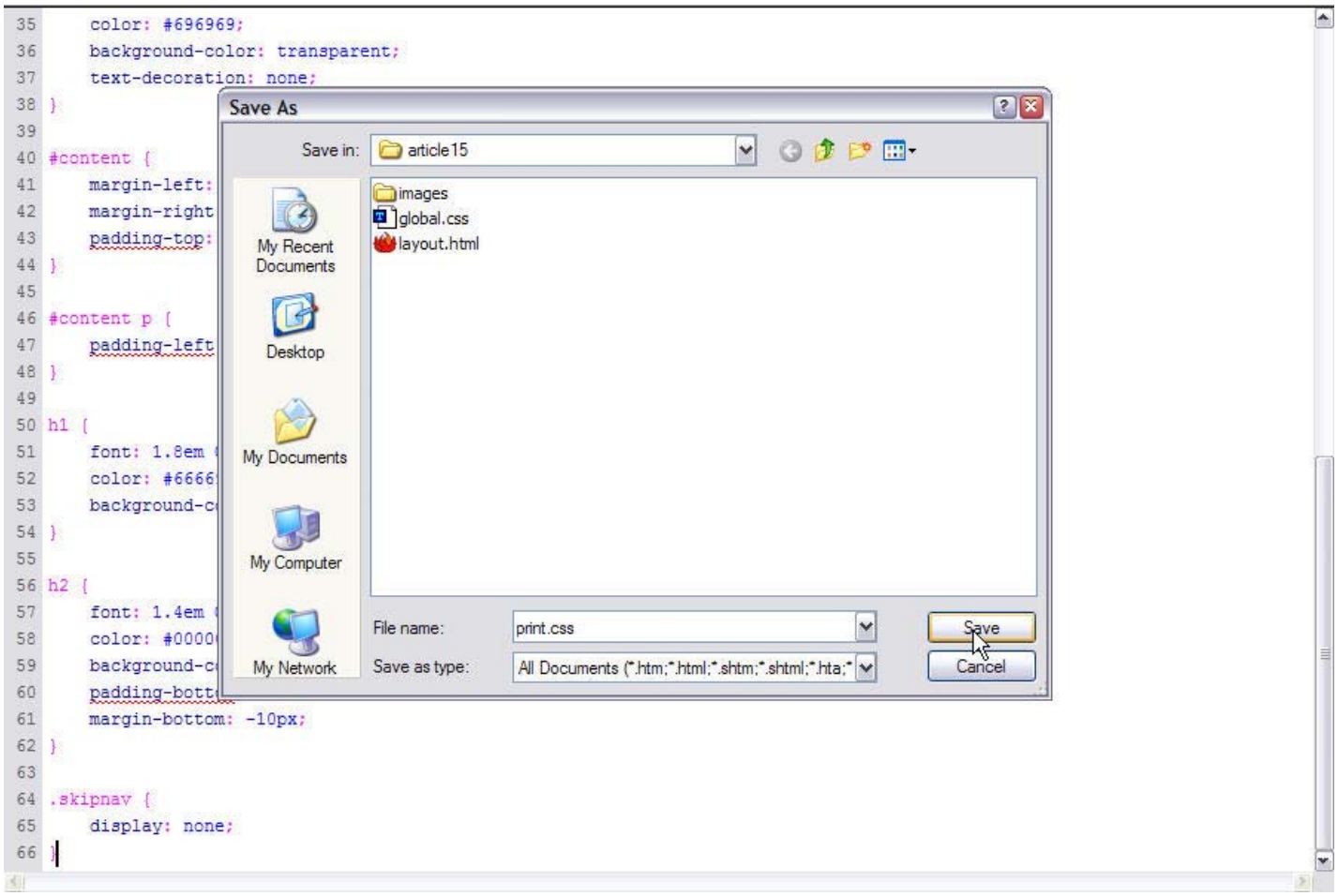
A print stylesheet gets around all of these problems. The print stylesheet comes into play when the user prints any document to which it is linked. You can define any element on your page differently in the print stylesheet and these are the styles that will be used when the document is printed. You can use the print stylesheet to hide areas of the page, such as navigation, graphics-intensive headers or unnecessary pictures; you can change the font styles, colors and sizes in order that the page is readable when printed; you can ensure that the contrast between colors works well even if the document is printed in black and white, and you can even add to the document areas that will only display on printing – such as page related information, to make it obvious where the document came from.

You can create a print stylesheet for an existing web site just as easily as you can for a new site, and many of the changes can be made without editing your document at all, other than to add the link to the new stylesheet file.

You can either follow this chapter using your own existing site – or use the files that I have using which are included as a code download.

Getting Started

When I create a print stylesheet, I usually do it as one of the final stages of development of the site. Once I have created a design I simply save a copy of my existing stylesheet as 'print.css'. That way I have a document that contains all of the styles defined for the web and I can simply alter them for print.



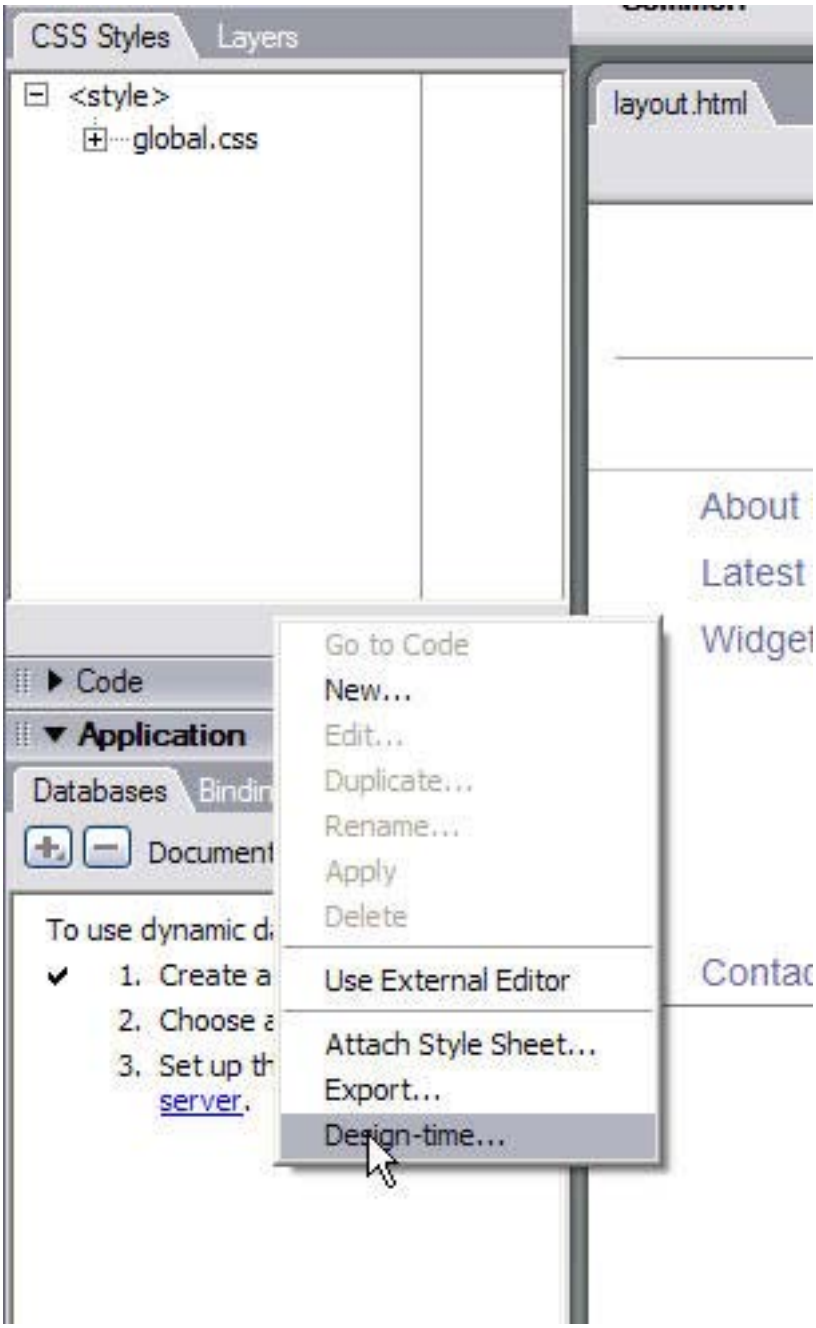
Saving the existing stylesheet as print.css

Using Design-time Stylesheets while creating a stylesheet

When working with your print stylesheet it will be helpful for you to be able to view it in Dreamweaver as you work. You can do this by using a Design Time Stylesheet. A Design Time Stylesheet is applied by Dreamweaver just so that you can see the effect of this stylesheet, it does not affect the way that the stylesheets are displayed once you upload your site to the web. In this case, our stylesheet will only display when the document is printed so by using it as a Design Time Stylesheet we can see how it will look as we work on it.

Adding the Design Time Stylesheet

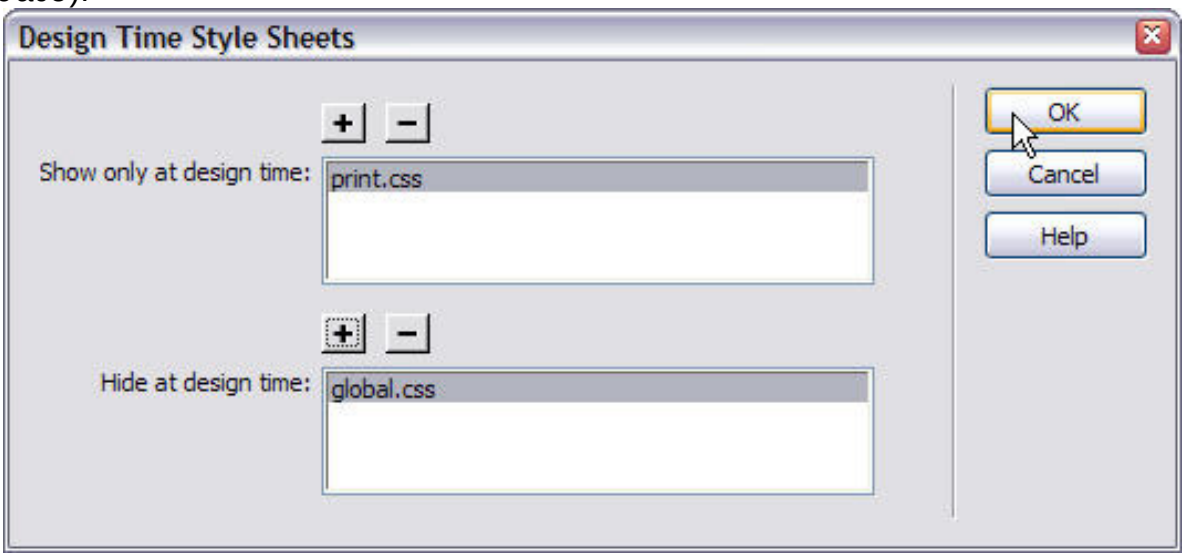
To add the stylesheet, right-click on the CSS Styles Panel and select 'Design time ...'



Select 'Design-time ...'

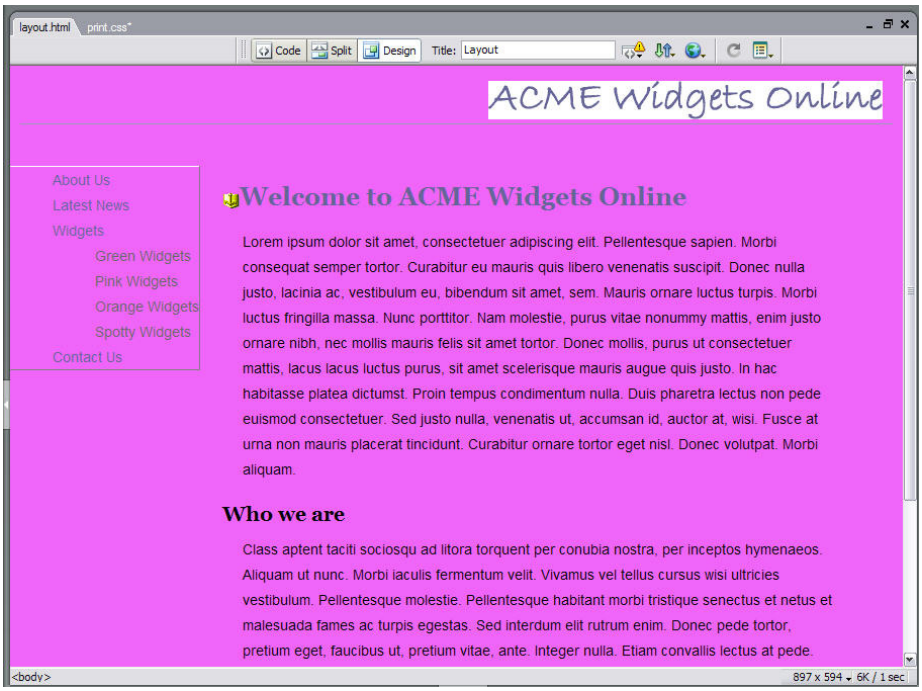
In the dialog that opens, click the cross above 'Show only at design time' and select your print stylesheet.

Repeat the process for 'Hide at design time' and select the real main stylesheet 'global.css' (in my case).



Design Time Style Sheets

Click OK and Dreamweaver should now be previewing using your print.css stylesheet – if you want to be sure then turn the background in print.css some putrid color; if you see the nasty color, then it has worked.:



What a lovely pink ...

Printing only relevant areas of the page

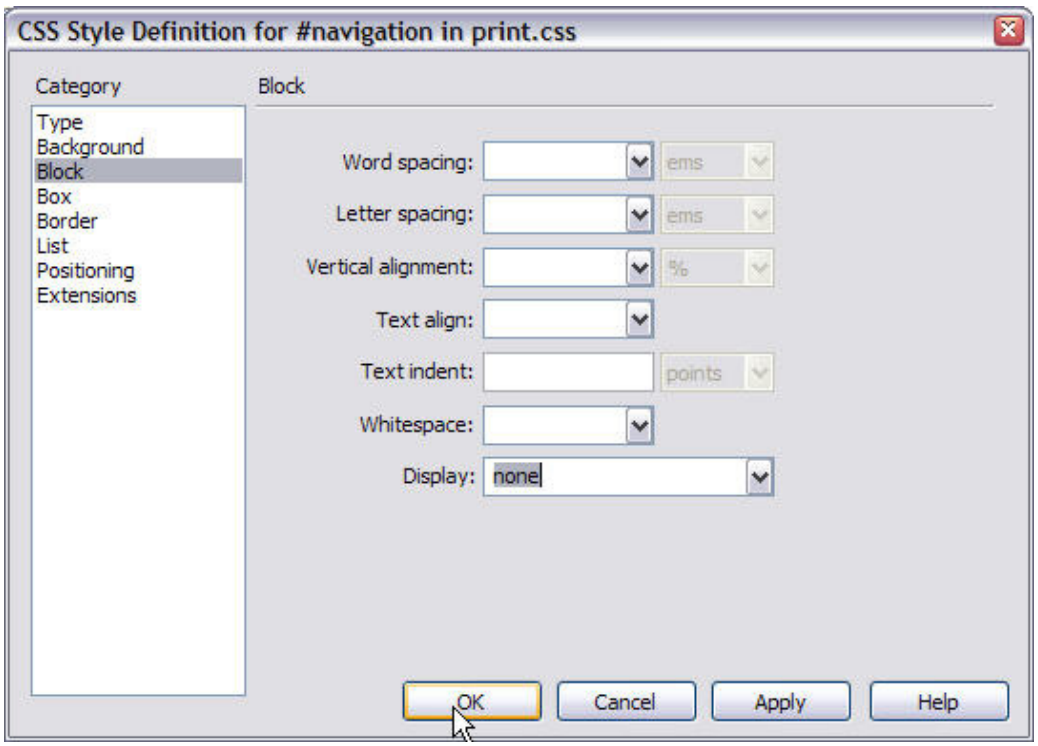
The first step in creating our print stylesheet is to remove any page elements that aren't necessary in the printed document. In our case this is the navigation. The navigation in my layout is already contained within a div named 'navigation' however, if the page element that you want to hide for print doesn't already have an id, give it one. Even if your layout is tables based rather than all CSS, you can give a table or even a table cell an id, in order to hide it.

```
@import url("global.css");
-->
</style>
</head>

<body>
<div id="banner"><img src="images/banner.gif" alt="ACME Widgets Online" width="390" height="38" /></div>
<div id="navigation">
  <ul class="main">
    <li class="skipnav"><a href="#content">Skip Navigation </a></li>
    <li><a href="#">About Us</a></li>
    <li><a href="#">Latest News</a></li>
    <li><a href="#">Widgets</a>
      <ul class="sub">
        <li><a href="#">Green Widgets</a></li>
        <li><a href="#">Pink Widgets</a></li>
        <li><a href="#">Orange Widgets</a></li>
        <li><a href="#">Spotty Widgets </a>   </li>
      </ul>
    </li>
    <li><a href="#">Contact Us</a></li>
  </ul>
</div>
```

My navigation div in Code View

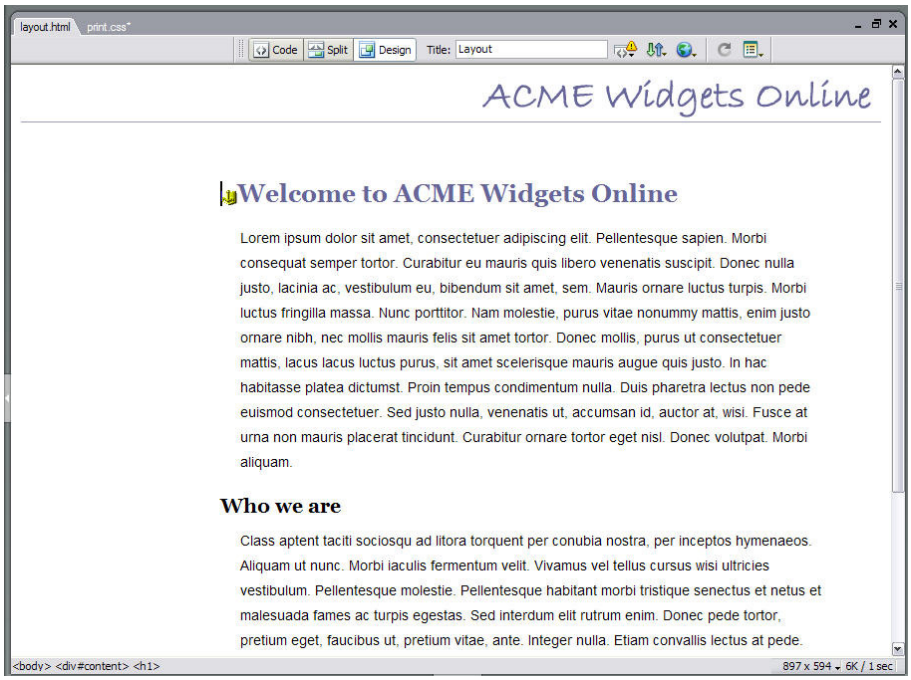
If you already have a named div for the area that you want to hide, edit that style in the print.css stylesheet in Dreamweaver.

In the Block category set the dropdown next to 'display' to 'none'.



Set Display: none

If you wrapped your area in a new div then create a New CSS Class using that name, and then set Display to none just as we did when editing the CSS Style above.Click OK and the area should disappear from the page in Design View.



The navigation is now hidden

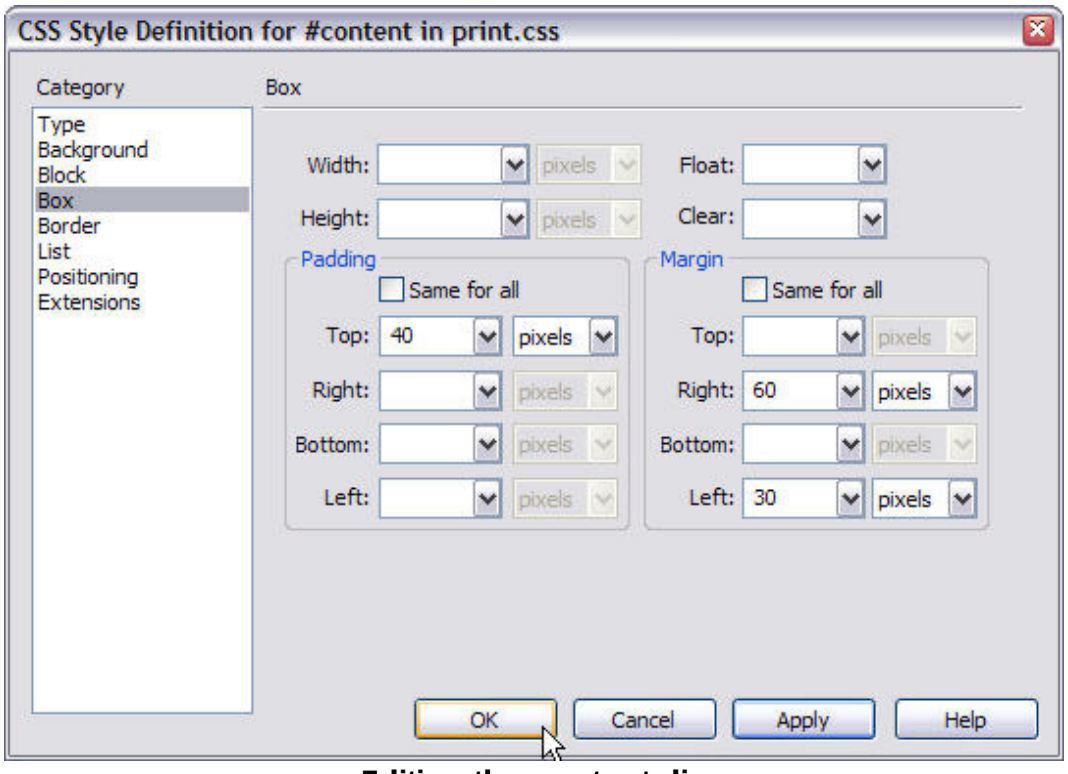
Making content stretch to fill the page

Now that we have hidden our navigation we have a gap where it once sat on the left hand side of the page. If we have large gaps on the page then the printed content could run onto more sheets of paper than it would do if it ran nicely up to close to the edges of the paper.

To eliminate the gap we need to make the content stretch into that space. If you are using a tables layout, depending on how you have structured the tables this may have happened anyway (as a result of the display:none) otherwise, if you have set the widths of the cells with CSS, the following method for CSS layouts will work just as well. If you have set the widths of cells using html attributes of the td tag you will need to convert these to CSS before this method will work.

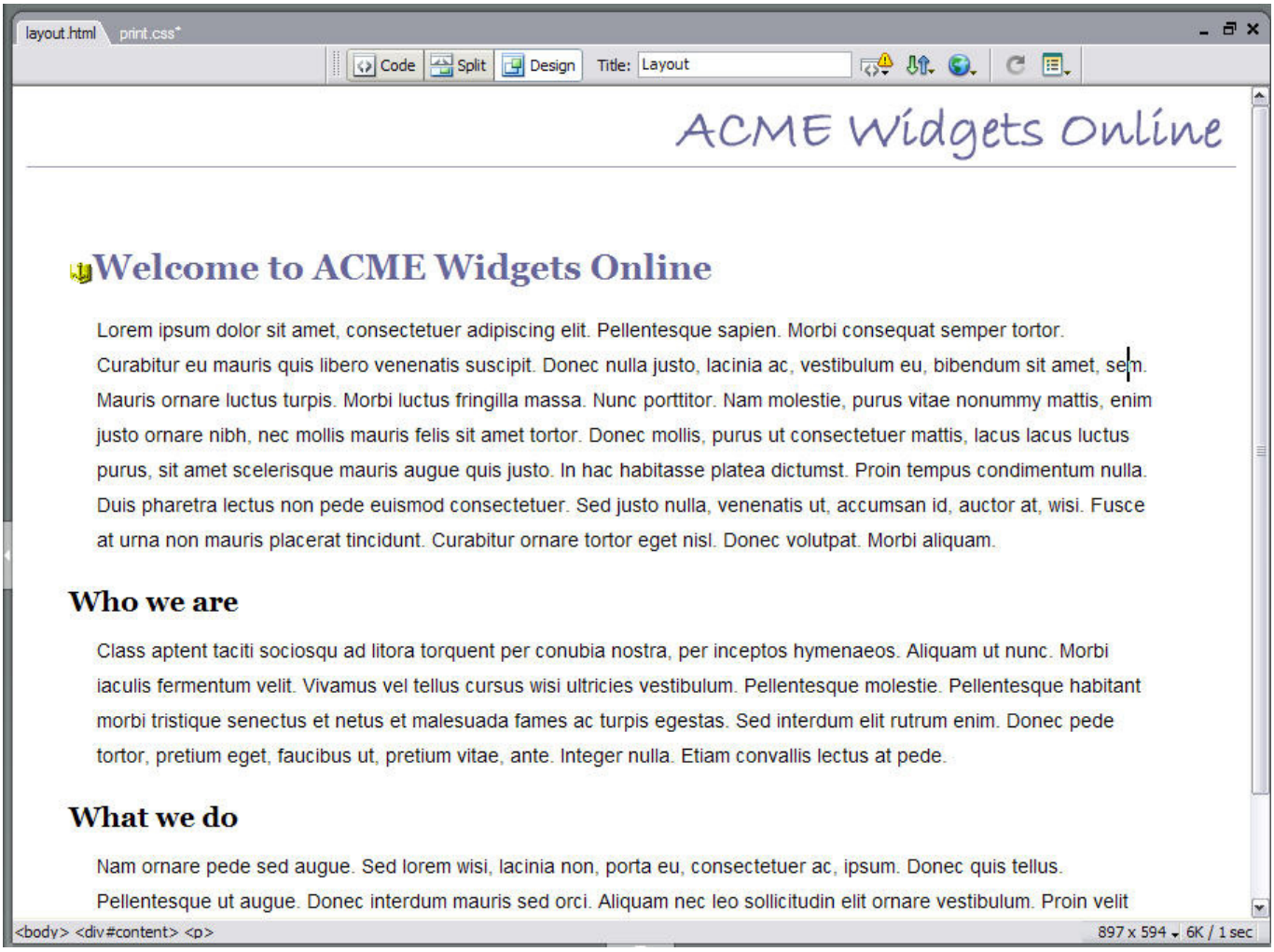
In my layout I need to edit the id 'content'.

In the Box category I have set a left margin of 200 pixels, which is what creates the gap. By changing that to 30 pixels I can effectively remove that gap.



Editing the content div

Click OK to see the content resize to fill the page area.

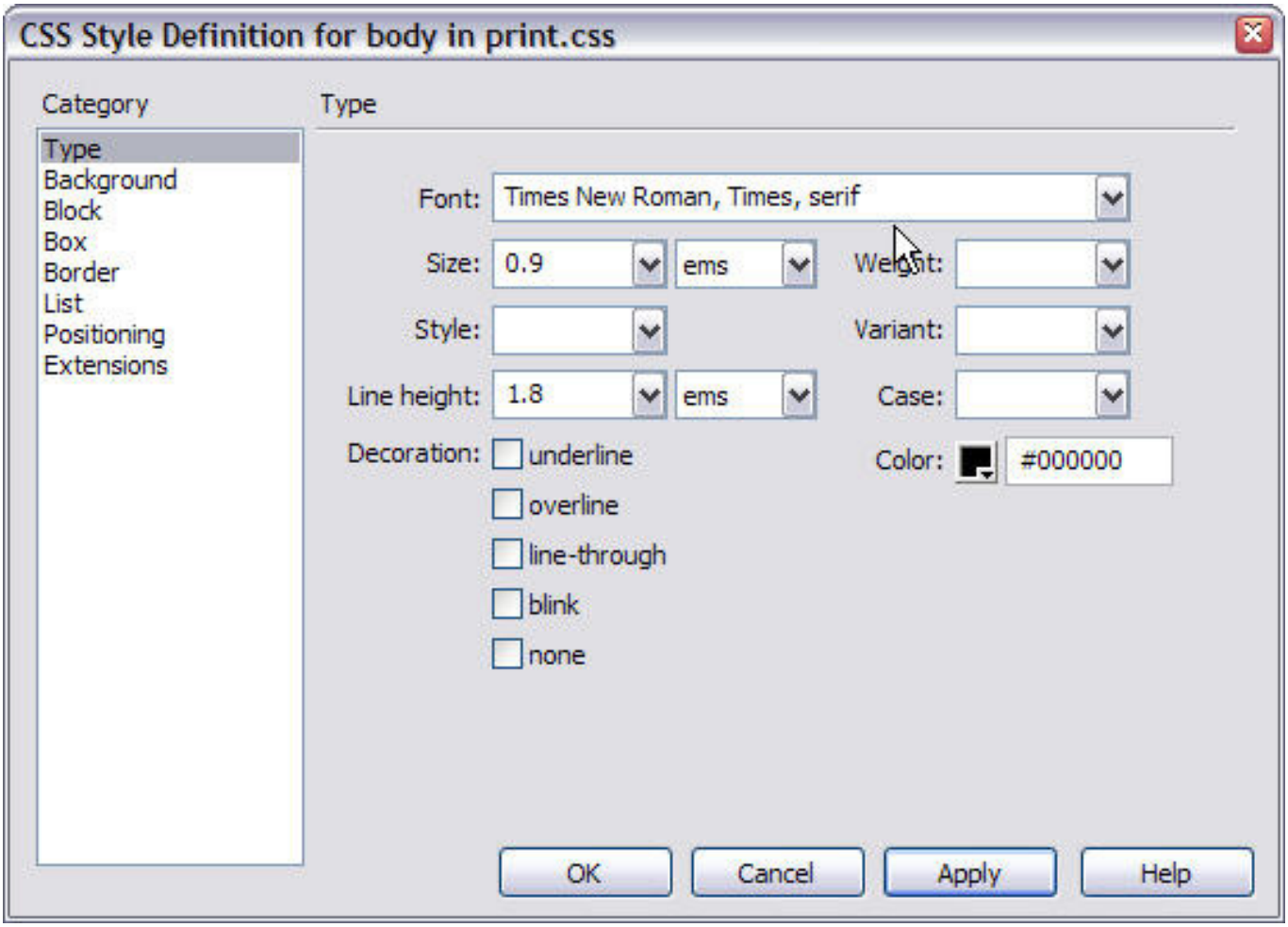


The layout with the navigation space removed

Using a different font style for print

Serif font styles, such as Time New Roman are easier to be read in print than sans-serif styles such as Verdana or Arial, which were designed to be read on the screen. You can change your font by editing it in the print.css stylesheet.

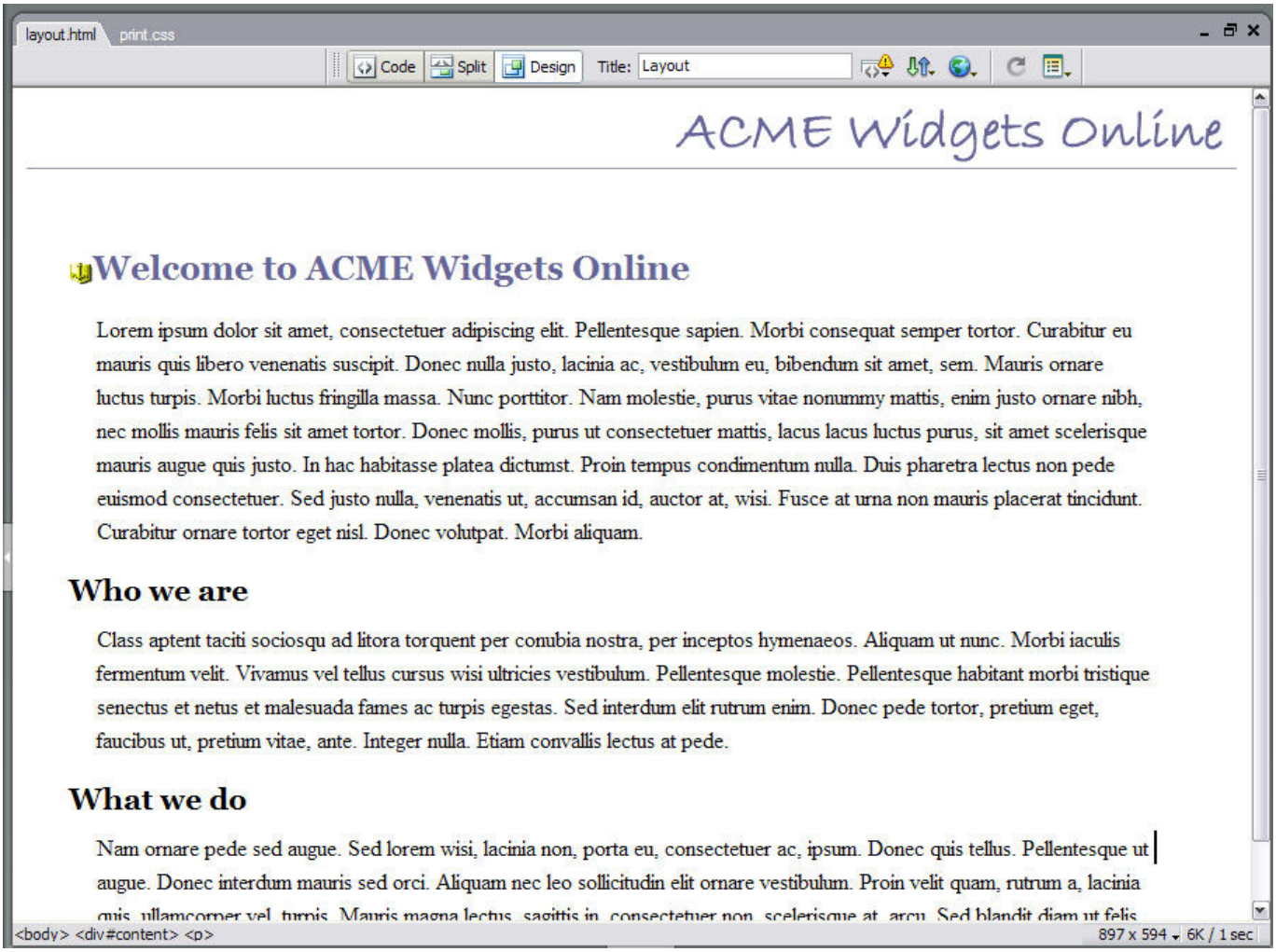
I have defined my fonts for the body tag and so edit the CSS Styles for body in Dreamweaver:



Changing the font style in the print stylesheet

Currently my font size is set in ems which are great for screen, but a better font sizing for print is points, which have a real world sizing – there are 72 points in an inch. So by setting your font in points you can know exactly how it will print out for the user. Points should not be used for screen stylesheets as the onscreen rendering of points is erratic between browsers and operating systems.

I have changed my font sizes to 12 points with the line-height set to 20 points. Don't forget to change your headings, and any other text elements too.

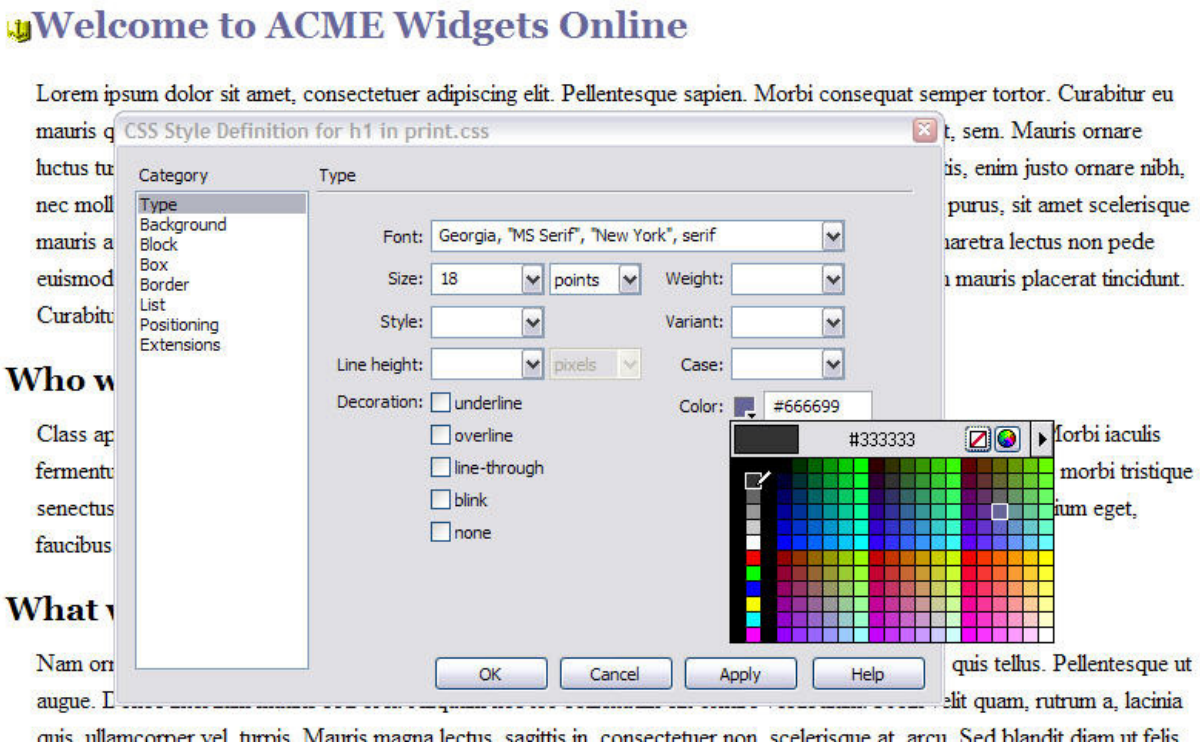


The layout after changing the font size and style

Converting to grayscale

Many users will only have a black and white printer, and if they just want the text, will probably prefer to print out in black and white. Coloured ink is expensive, too! If your site uses colored text, or light text on a dark background then you should change this in your stylesheet for maximum readability and ease of printing (your dark blue background with white text may look lovely online but no-one wants to print out pages and pages of deep blue!)

Simply change the color values in the print stylesheet to standard black on white; for some variation you could use shades of grey for headings or other elements as long as they are not too light.



Converting colored elements to grayscale

Display page information on printed versions

As well as hiding elements, you can display additional elements using the print stylesheet – a boxout containing the page's URL might be of use if someone is taking a copy for research purposes or to pass onto someone else.

When adding information we will display and format this information using the print stylesheet and hide the information using the site's main stylesheet (global.css).

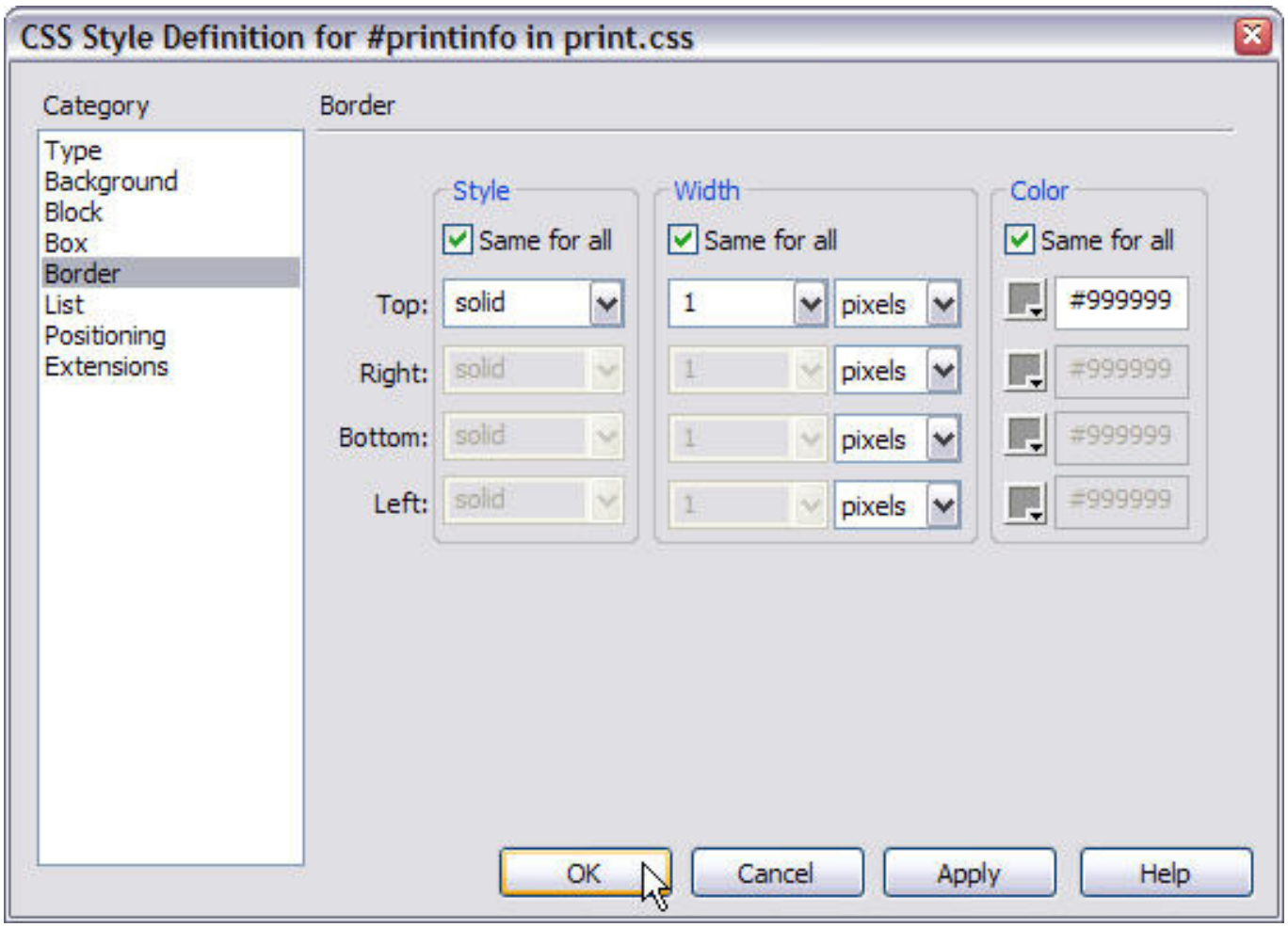
In your document add a div with an id of 'printinfo' containing the URL of this page.

```
<div id="printinfo">
     <p>Printed from: http://www.mysite.com/mypage.html</p>
</div>
```

```
31 <div id="content">
32     <div id="printinfo">
33     <p>Printed from: http://www.mysite.com/mypage.html</p>
34     </div>
35   <h1><a name="content" id="content"></a>Welcome to ACME Widgets Online </h1>
```

The div 'printinfo' in Code View

You can then style this div any way you wish by adding a CSS Style for #printinfo into print.css



Adding styles to print.css for printinfo

Now add styles for #printinfo to global.css (the main stylesheet for the site). We want to hide this area when the page is viewed in a web browser. To do so set display: none. As global.css is currently hidden at Design Time, Dreamweaver won't let you edit it through the panel – the quickest thing to do is to add the code directly to the stylesheet.

Open up global.css in Dreamweaver and add:

```
#printinfo {
     display: none;
}
```

to the bottom of the stylesheet.

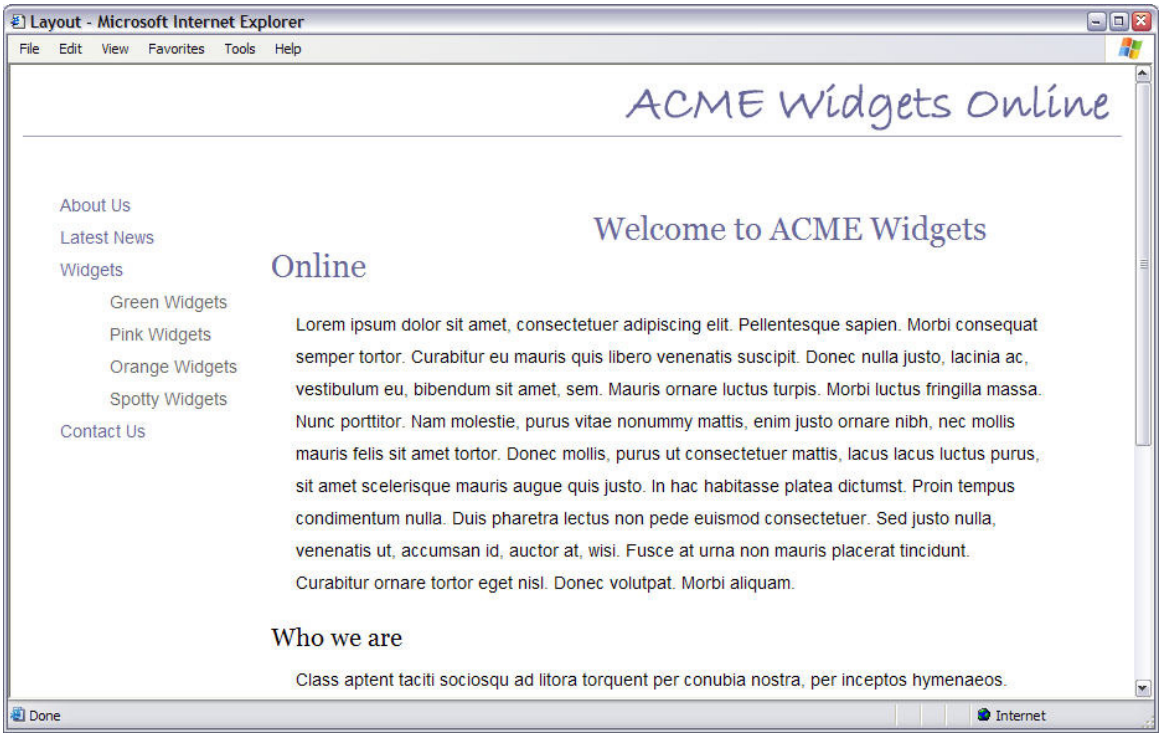
Attach the stylesheet to the document

Now that you have created your stylesheet you simply need to link it to your document in such a way that it understands that this stylesheet is only for print. *After* your existing link to your main stylesheet add the following:

<link rel="stylesheet" type="text/css" media="print" href="print.css" />

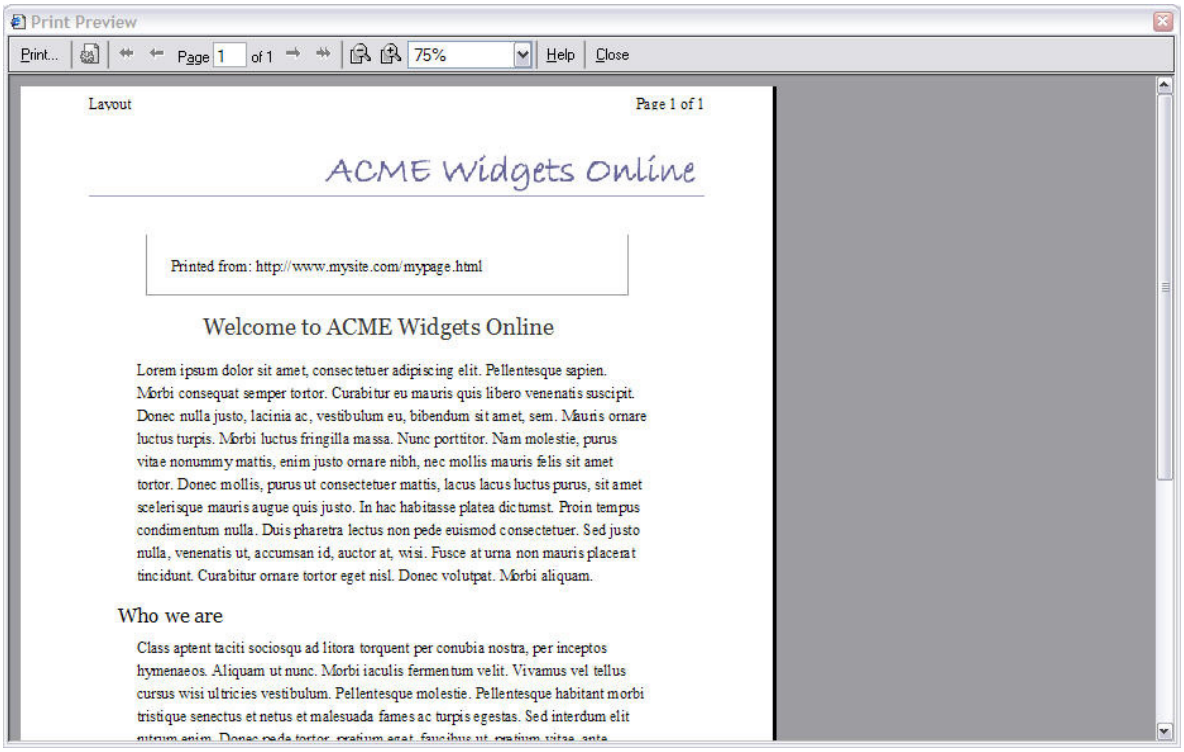
This is just a normal link to a stylesheet file with one important addition, the **media attribute** which tells the browser that this stylesheet is only for use when the document is printed. There are other media attributes – to link stylesheets for a range of media including Braille devices, speaking browsers and PDAs, however support for these other media types is limited. The print media type is well supported across current browsers.

After adding this line and saving the document you can now preview in a web browser. The layout should look exactly the same when viewed in this way.



The layout in a web browser

To view the document as printed you can obviously print it out, but to save paper you can also view it in print preview, which will show the document as it will look using the print stylesheet.



The layout in print preview

You can edit your stylesheet and tweak the print layout until you feel that your document is printing out in an attractive and user friendly manner, presenting a good image of the company or organization who is providing this information. You can take this technique further and even display special 'printer friendly' versions of logos and other graphics (ready optimized for grayscale) by inserting these images as background images using CSS and then displaying the print version when the page is printed. For web sites that do provide a lot of information then spending some time creating an attractive stylesheet for print can be a real enhancement to the site, and something that people will find very useful.

A good example of a well implemented print stylesheet can be found at A List Apart, have a look at the following URL in Print Preview and you can see many of the techniques that we have discussed in use. (http://www.alistapart.com/articles/javascriptreplacement/)

Summary

In this chapter we have learned how to create a print stylesheet for a web site. Even on an existing site this is a simple way to add practical and useful functionality, and the creation of a print stylesheet is not a time consuming job. There are more options for arranging the

page when using a CSS layout, however most of the techniques described here will work well on a well constructed tables site, where CSS has been used for as much of the formatting as is possible.

Appendix A: CSS and Old Browsers

If you have been designing for the web for any length of time you are probably well used to testing your work in a variety of different browsers and understand the need to make your sure that your site doesn't crash or render in an unreadable manner in any of the browsers in use. You may also have been put off using anything more than the most basic of CSS because of potential problems in older browsers, and when we talk about old browsers we usually mean the browser we all love to hate, Netscape 4.*.

The debate about whether one should care about Netscape 4 at all continues to rage, it does depend on your target audience – I have sites that never see a version 4 browsers and yet other people quote figures as high as 10%. It's a question of looking at your server logs.

When I talk about accessibility I don't simply mean that visually impaired users can use the site, I mean that ALL users can use the site easily and that includes those who can't or won't upgrade their browser. Browsers older than version 4, or any other browser that doesn't have support for CSS are not going to be a problem to us – they will ignore the CSS, so if you have structured your content logically they will display the content without any problems. Netscape 4 is a problem because it has some support of CSS, and some very bad support of certain CSS that can cause the browser to crash completely or render sections of your page unusable or unreadable.

If you need to test for Netscape 4, you can download it from http://wp.netscape.com/download/archive.html. Stress relievers can be found here or here.

Hiding styles from Netscape 4

The easiest and most graceful way to give Netscape 4.* something it can cope with whilst still being able to use CSS fully in your designs is to use a method that 'hides' the newer styles from it. This method does not require JavaScript to be enabled in your browsers – although Netscape 4 actually requires JavaScript to display CSS. Anyone who uses Netscape 4.* with JavaScript turned off will be the same experience as your users who have text-only browsers or devices, as long as your page is structured logically.

The way that Dreamweaver attaches your style sheet by default is to use the link tag.

```
<link href="global.css" rel="stylesheet" type="text/css" />
```

The **<link>** tag is the most used method of attaching external stylesheets, as it is supported by all CSS supporting browsers. However, there is another method of attaching a stylesheet that is **not** recognized by version Netscape 4 browsers, and we can use this lack

of support to our advantage. This method uses **@import** to attach the style sheet to the page.
To attach a style sheet using @import in Dreamweaver MX, select link style sheet from the CSS Panel and in the dialogue that follows browse for your style sheet as usual but select the import radio button instead of using the default link radio button.

Adding an external style sheet using @import

Attaching your style sheet using this method will give you the following mark-up in the head of your document.

```
<style type="text/css">
<!--
@import url("global.css");
-->
</style>
```

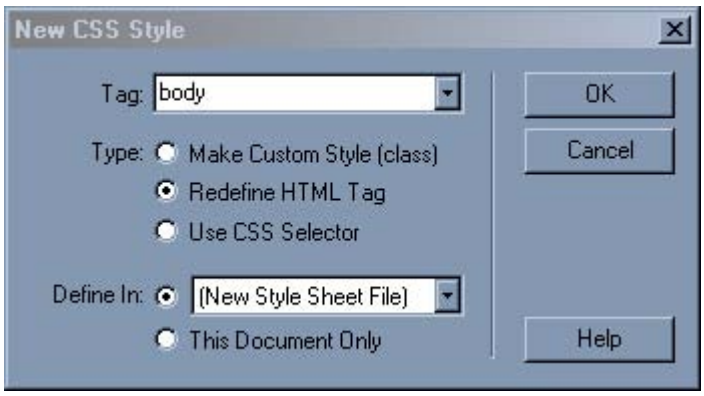
To test this out, set a very silly background color for the body in the CSS file – I find #FF33CC to be a marvellous color for testing things, you won't miss it! When you preview the page in an up to date browser such as IE6 or Netscape 7, and also in Netscape 4.* you should see your background color display in all its garish glory in the newer browsers but the background should remain as default in Netscape 4. This is because Netscape 4.* doesn't support this method of attaching a style sheet and so doesn't see the style that you have defined.

If you know that you have a very low proportion of Netscape 4 users and just want to ensure that your site doesn't crash the browser of those rare individuals, you can leave it at that, - attach your style sheets using @import and as long as your content is structured sensibly your Netscape 4 users will be able to read it in the same way that anyone using a text only browser can.

However, if you want to give your Netscape 4 users a little more than a default blank page you can use a linked style sheet for them and your imported one for the newer browsers, here is how to do that.

Staying with your test page select New CSS Style in Dreamweaver, in the dialogue select to redefine the Body tag but instead of selecting your existing (imported) style sheet to

create the style in, select New Style Sheet File.



Redefining the body tag of our new style sheet

Click Ok and save this new style sheet as old.css, then set the background-color of the body tag a different garish color to that which is set in the other style sheet – I recommend #666633. Click OK. Switch into Code View, you should find that Dreamweaver has used the link method to attach old.css but has put it after the imported style sheet.

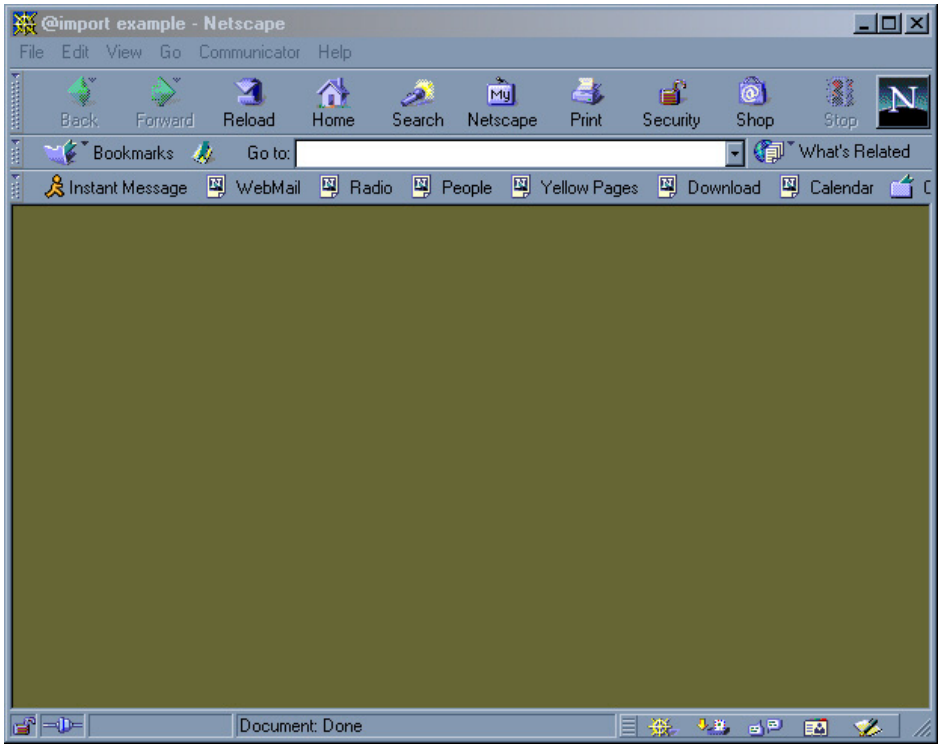
```
<style type="text/css">
<!--
@import url("global.css");
-->
</style>
<link href="old.css" rel="stylesheet" type="text/css" />
```

Where we want the link is before that import section, so cut and paste it to there:

```
<link href="old.css" rel="stylesheet" type="text/css" />
<style type="text/css">
<!--
@import url("global.css");
-->
</style>
```

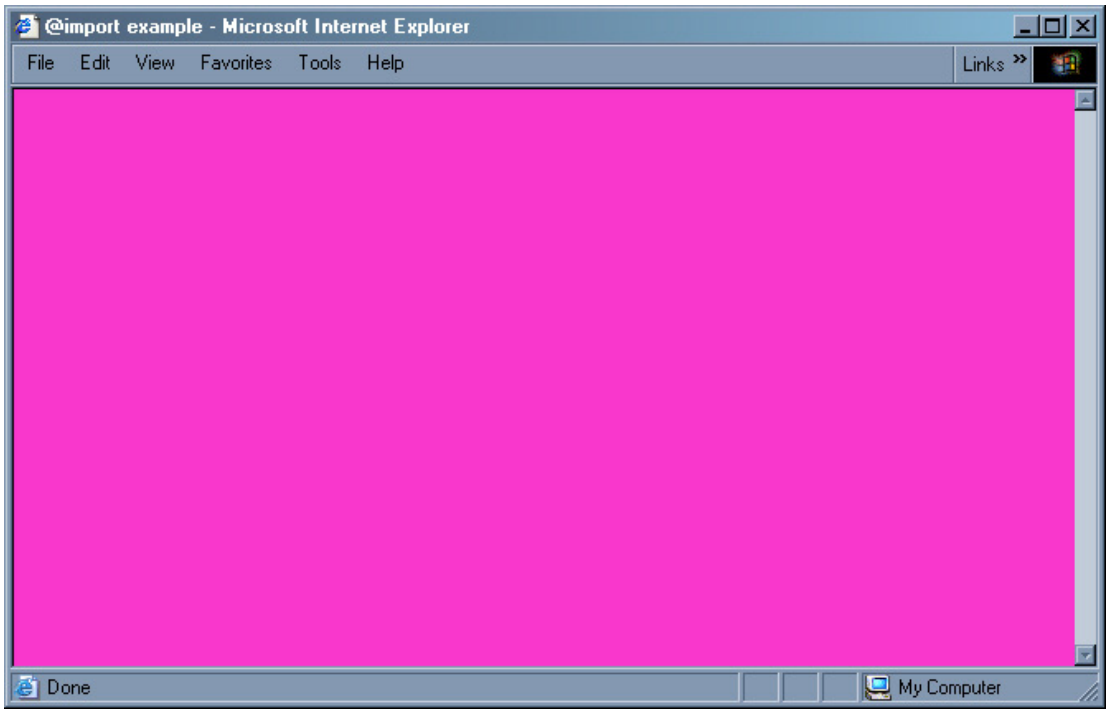
The reason **we need them in this order** is because what we are doing with the imported style sheet is overwriting the values in the linked style sheet. Newer browsers see both methods, so they will first come to the linked sheet and take notice of what is set there and then see the imported sheet. If there are the same classes and tags in the imported sheet they will take precedence as they come nearer to the content in the document – in the same way that if you have an external style sheet and inline styles the inline styles will take precedence.

If you save your page and preview in Netscape 4 you should see the background color set in the old.css style sheet as shown below.



The background from the linked style sheet displays in NS4.*

Previewing in a newer browser you will see the background color from the imported style sheet.



Beautiful Pink won't work in Netscape 4.

which will not work in Netscape 4.*. This browser does support some positioning, and so if you are careful you can lay out your page in an acceptable manner for Netscape 4 even if it does not have all of the style of your layout for more capable browsers.

I have created a simple layout that uses positioning that Netscape 4.* has no ability to render – using absolute positioning from the right of the document in order to create a liquid layout with a right hand menu. The mark-up for this layout looks like this:

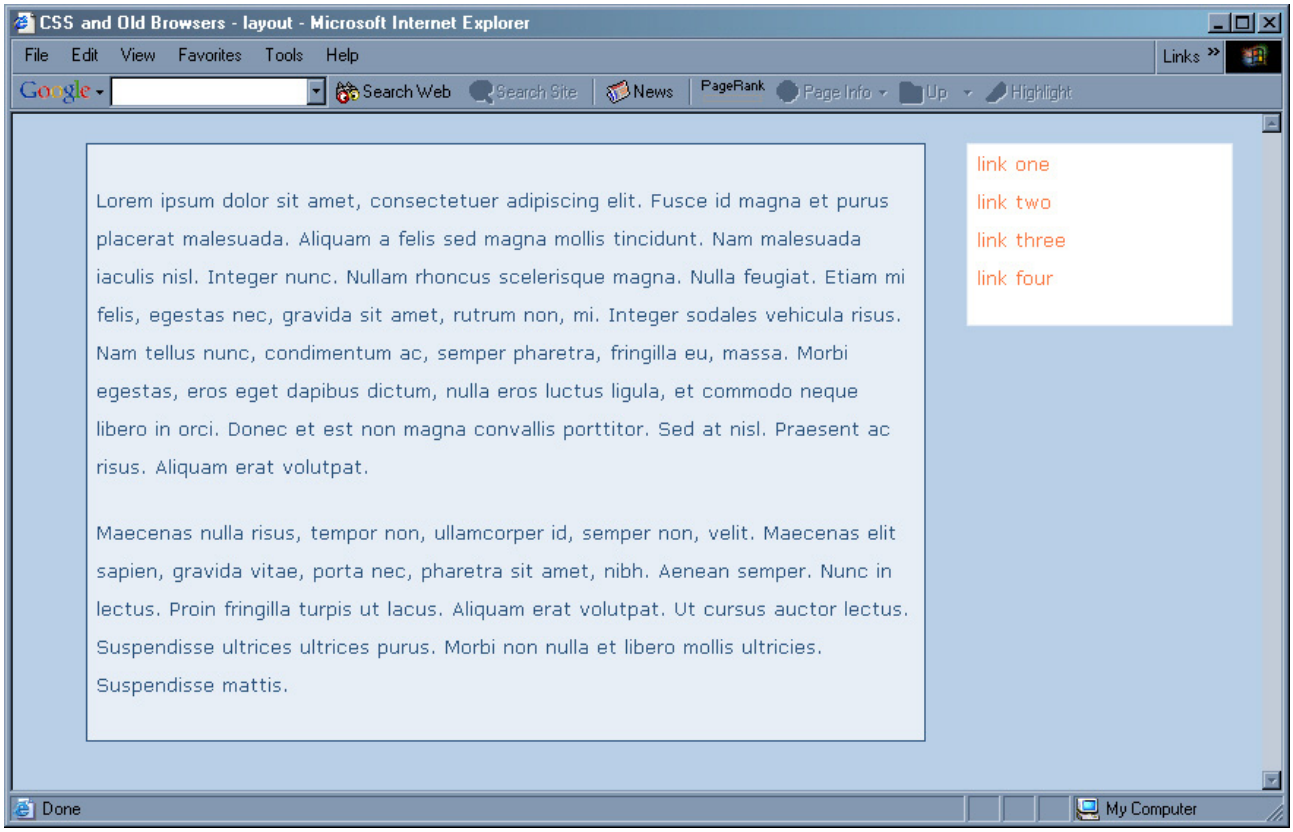
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>CSS and Old Browsers - layout</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>


<link href="global.css" rel="stylesheet" type="text/css" />
</head>

<body>
<div id="nav">
  <ul>
    <li><a href="#">link one</a></li>
    <li><a href="#">link two</a></li>
    <li><a href="#">link three</a></li>
    <li><a href="#">link four</a></li>
  </ul>
</div>

<div id="content"><p>Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Fusce
  id magna et purus placerat malesuada. Aliquam a felis sed magna mollis
tincidunt.
  Nam malesuada iaculis nisl. Integer nunc. Nullam rhoncus scelerisque
magna.
  Nulla feugiat. Etiam mi felis, egestas nec, gravida sit amet, rutrum
non, mi.
  Integer sodales vehicula risus. Nam tellus nunc, condimentum ac, semper
pharetra,
  fringilla eu, massa. Morbi egestas, eros eget dapibus dictum, nulla
eros luctus
  ligula, et commodo neque libero in orci. Donec et est non magna
convallis porttitor.
  Sed at nisl. Praesent ac risus. Aliquam erat volutpat.</p>
  <p>Maecenas nulla risus, tempor non, ullamcorper id, semper non, velit.
Maecenas
```

```
    elit sapien, gravida vitae, porta nec, pharetra sit amet, nibh.
Aenean semper.
    Nunc in lectus. Proin fringilla turpis ut lacus. Aliquam erat
volutpat. Ut
    cursus auctor lectus. Suspendisse ultrices ultrices purus. Morbi non
nulla
    et libero mollis ultricies. Suspendisse mattis.</p>
</div>
</body>
</html>
```

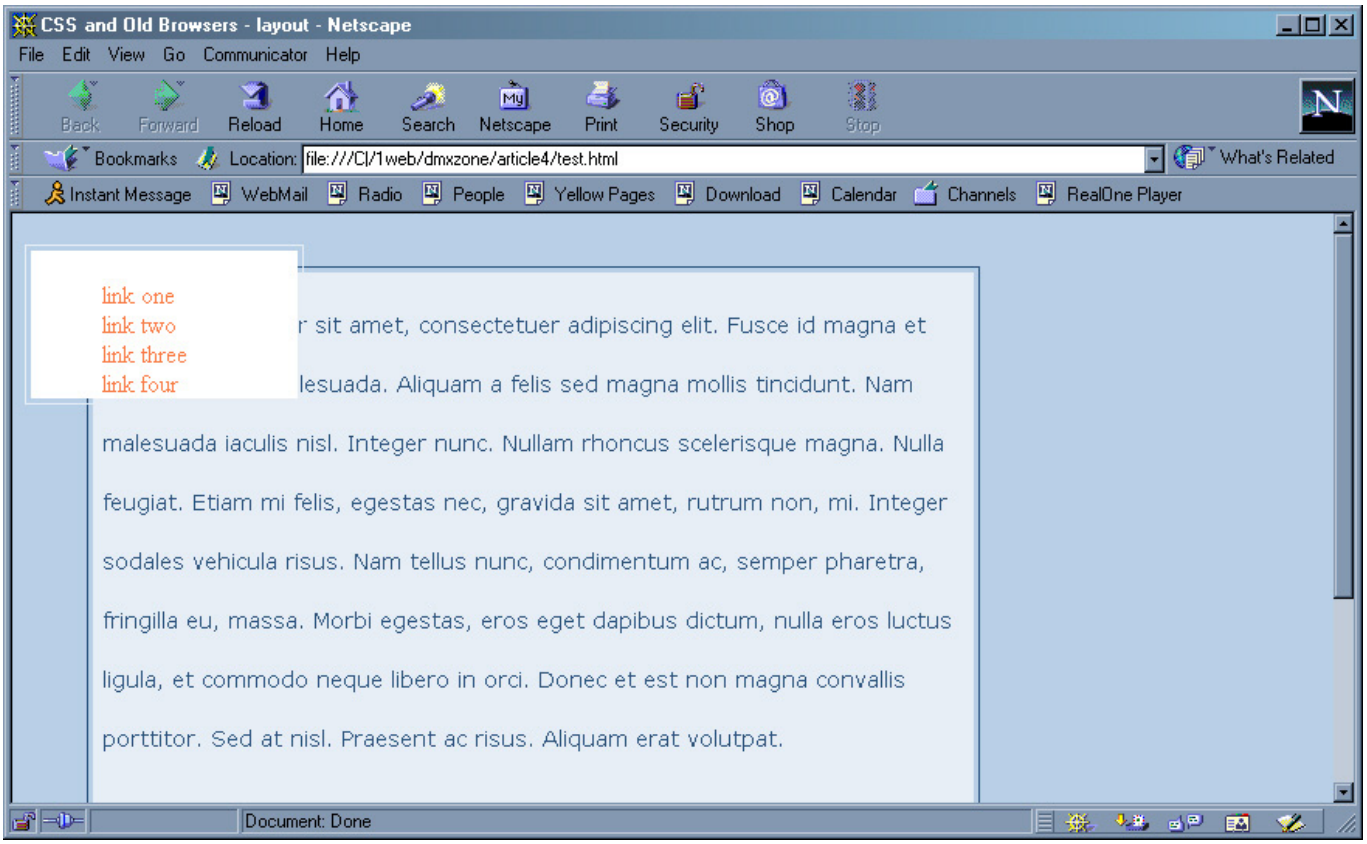
And here is the CSS - which I have saved in an external style sheet and attached to the page by the link method.

```
body {
     background-color: #B7D0E7;
     color: #000000;
     padding-top: 0px;
     margin-top: 20px;
}

p, td, li {
     font: 0.8em/2em Verdana, Geneva, Arial, Helvetica, sans-serif;
}

#content {
     margin-right: 220px;
     margin-left: 40px;
     background-color: #E6EEF6;
     color: #29547D;
     padding: 6px;
border: 1px solid #29547D;
}

#nav {
     position: absolute;
     top: 20px;
     right: 20px;
     width: 180px;
     background-color: #ffffff;
     color: #000000;
     border: 1px solid #E6EEF6;
}

#nav li {
     list-style-type: none;

}
```

```
#nav ul {
     padding-left: 6px;
     margin-left: 0px;
}

#nav a:link, a:visited, a:active {
     background-color: transparent;
     color: #FF7844;
     text-decoration: none;
}

#nav a:hover {
     background-color: transparent;
     color: #191970;
}
```



Our document displayed in IE6 and other up to date browsers



Our document as displayed in Netscape 4.75

As you can see Netscape 4.75 doesn't do too bad a job of rendering this layout, but because it doesn't support the right positioning it will dump the menu on top of the content.

We can use our @import method to fix this problem without having to rethink the layout for users of those browsers that support it.

First, open up your style sheet and save it as 'old.css'.

Switch to Code View and change the link to the style sheet to link to old.css as opposed to global.css.

Now reattach global.css using the @import method as we did earlier. You should end up with this in the head of your document – be sure that the import section comes after the link:

```
<link href="old.css" rel="stylesheet" type="text/css" />
<style type="text/css">
<!--
@import url("global.css");
-->
</style>
```

Now open up your old.css in the Code View of Dreamweaver MX.

There are two ways in which I have dealt with this situation in the past – one is to make the layout fixed width for Netscape 4 as then we can make the content block a set width and position the navigation using absolute positioning from the left, the other (and this is how I will approach it here) is to swap the menu over, so that it appears on the left of the content for Netscape 4 users.

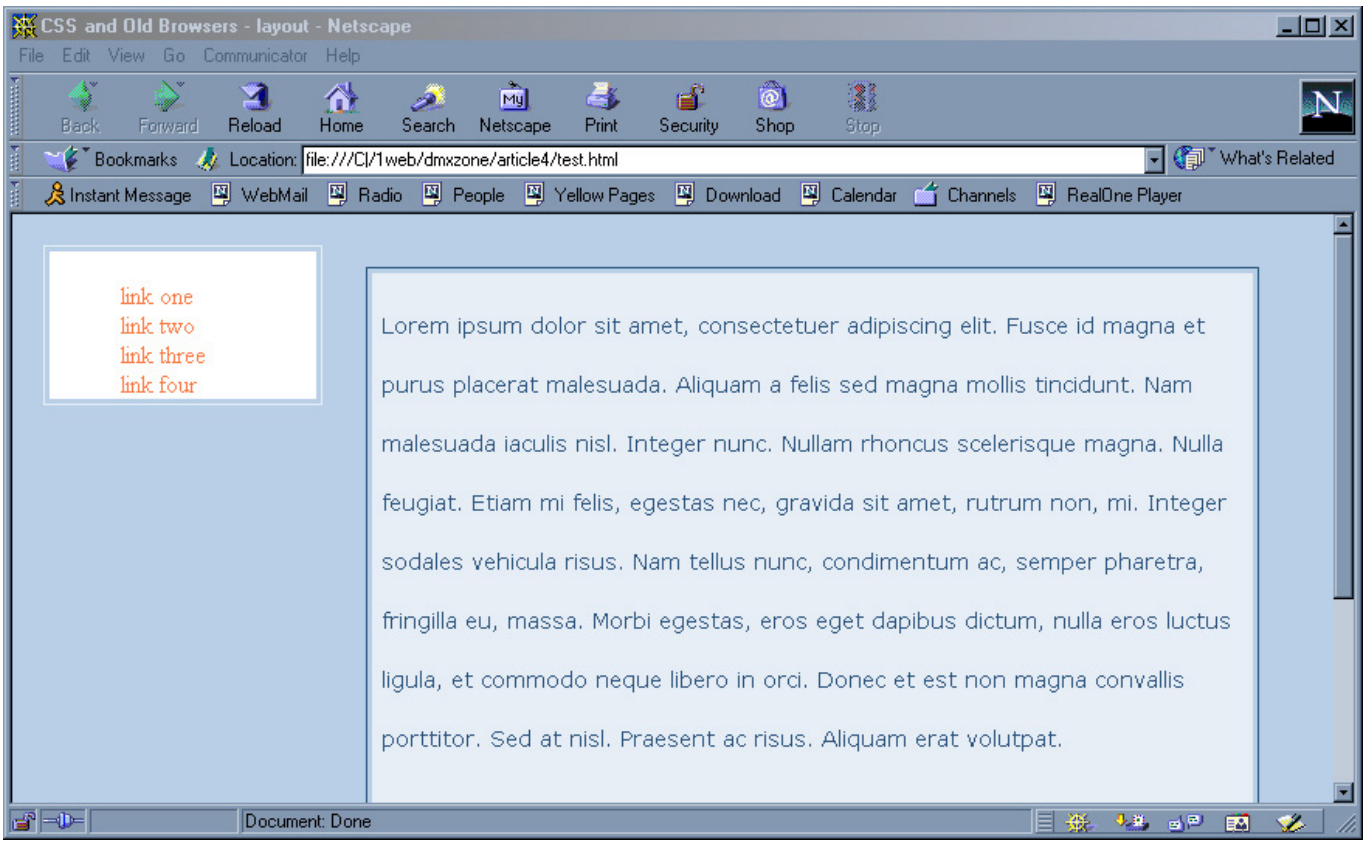
To swap the menu to the left of the content, edit the section (in old.css) for #content. Set margin-right to 40px and margin-left to 220px.

```
#content {
     margin-right: 40px;
     margin-left: 220px;
     background-color: #E6EEF6;
     color: #29547D;
     padding: 6px;
     border: 1px solid #29547D;
}
```

Now edit the section for #nav changing right: 20px to left: 20px.

```
#nav {
     position: absolute;
top: 20px;
     left: 20px;
     width: 180px;
     background-color: #ffffff;
     color: #000000;
     border: 1px solid #E6EEF6;
}
```

Now preview your document in Netscape 4.*



Our document in Netscape 4.75 after changing old.css

If you have previewed the document in a newer browser already, you will be muttering about how I don't know what I'm talking about as it will appear broken, with the menu jumping over to the left. There is a reason for this. If you remember when we did our test page earlier, I explained that what we are doing is over writing the values set in the linked style sheet with different values for those browsers that see the imported style sheet. This means that any value in the linked style sheet must have another value over writing it in the imported sheet – and we have just added a left positioning value to our linked style sheet that does not appear in the imported one.

All you need to do to fix that is to open up global.css, and in the section for #nav add the following line - left: auto; making the section look like so:

```
#nav {
     position: absolute;
     left: auto;
top: 20px;
     right: 20px;
     width: 180px;
     background-color: #ffffff;
     color: #000000;
```

```
     border: 1px solid #E6EEF6;
}
```

Preview your page now and it should have returned to its prior state.

You can continue to tweak the values in the old.css style sheet to make the viewing experience of the Netscape 4.* user better – how much you feel you need to do this really depends on how likely you are to get users using this browser. As you can see from this example, however, it really isn't difficult to provide them with something that will be usable and even relatively attractive and so even if you expect to get very few users you may as well provide something – even if it is just neatly laid out text on a plain background.

The 'Netscape Resize Fix'

Netscape 4 has a bug that causes all of your areas positioned with CSS to pile on top of each other or totally disappear when the browser window is resized. You can try this with our layout by loading it in Netscape 4 and resizing the window. Dreamweaver MX has a fix for this – if you have ever worked with Dreamweaver 'layers' you will have noticed that it adds a block of JavaScript into the head of the document when you add a layer – if you are using an external style sheet then you will need to add this yourself.

You can add this within Dreamweaver by selecting 'Add Netscape Resize Fix' from the commands menu. If you are going to be creating many pages then it makes sense to put this command into an external JavaScript file to save it being on every page. After adding this, try resizing your window and you will see the page reload and the elements return to where they should be.

Even if you are not using CSS positioning, the technique we have looked at today can be very useful – Netscape 4.* has well documented problems with styles on form elements, adding borders and background colors to form elements can at best leave your form looking peculiar and at worst make it totally unusable. Importing the styles for your form will ensure that all users can complete your forms. Netscape 4 also renders font sixes differently in comparison to newer browsers, so a comfortable text size in IE6 might look strange in Netscape 4 – again, using the import method you can tweak your sizing separately for each browser, giving a good experience for all of your users.

Appendix B: DOCTYPE switching in Dreamweaver MX, and the Box Model

About DTDs and DOCTYPEs

A Document Type Definition or "DTD" is a definition for the language and language version in use. I like to think of a DTD as a long laundry list of elements, attributes, and other syntax and structure rules inherent to the particular language version that the DTD helps to describe.

General features of a DTD include:

- A DTD is machine-readable for parsing
- A DTD is also human-readable and understandable
- A DTD is an ASCII (text) document
- DTDs express syntax and structure
- A DTD is declared with the DOCTYPE declaration

DTDs have been the means by which all HTML and XHTML languages and versions have been defined. If you'd like to scare yourself and see what a DTD looks like, visit the following link for the HTML 4.01 Strict DTD:

http://www.w3.org/TR/html401/sgml/dtd.html

A DOCTYPE declaration declares the language version for the document it represents. DOCTYPE declarations don't quite look like any other HTML you use--that's because they're not HTML. Rather, DOCTYPE declarations use SGML syntax. You may know that SGML is the parent language to HTML--which is where this bit of formality came from.

The sophisticated web author will adhere to the rules of a given DTD, and declare that adherence by including a DOCTYPE declaration at the top of a web document. Similarly, authoring tools will often insert a DOCTYPE declaration automatically onto a page.

If you use Dreamweaver or Homesite regularly for HTML, the following will look very familiar:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

This DOCTYPE declaration is, by default, inserted automatically by Dreamweaver into all documents generated by clicking on File > New > Basic HTML page.

Note: If you're using a lot of presentational markup and not using CSS or CSS positioning, this default DOCTYPE is a good choice. However, if you are using CSS and want more control, read on!

So, in the case of the default DOCTYPE, we can, just by looking at it, determine that the document in question:

- is an HTML document
- is available to the public
- resides at the W3C
- follows the HTML 4.01 STRICTDTD
- is in English

"Great!" you're no doubt thinking. "So my document is identified--big deal. Does this stuff *do* anything?"

Days of DOCTYPEs Past

Aside from declaring the document type to the author, in the past the DOCTYPE declaration was passive in terms of a web browser. No action was taken by a browser based on the existence--or lack of--of a DOCTYPE declaration in a document.

However, there is significant value to a DOCTYPE declaration in addition to simply declaring the document's type--and this is when validating a document. Validation tools take the document being validated and compare it to the DTD using the information declared within the DOCTYPE.

Since validation has been very low on the priority list over the past years, not many people paid attention to DOCTYPEs, DTDs, or even to validation tools for that matter. But this has changed significantly in recent times, for numerous reasons. One is general awareness that adhering to web standards and languages makes for more consistent workflow and results. Another, equally important but lesser known reason is that the DOCTYPE declaration is no longer a passive portion of your document.

In fact, how a DOCTYPE is formed, and that it even exists in your document now plays a very significant part in helping you gain that elusive control over your designs.

The Box Model Nightmare

Browsers use visual models in order to flow and format content. Aspects of presentational markup and CSS (especially positioning) rely on these models to produce a given display within a browser. Within browsers, most elements generate a box, and this is referred to as the "Box Model."

But the visual modeling technologies in browsers are different. Most frustrating to today's contemporary designer, the Box Model is significantly misinterpreted by Internet Explorer, which is by far the most common browser in use.

The Box Model problem is a perfect example of why designers need to know about the issues raised in this chapter. But this problem is only one of *many* issues caused by inconsistent browser technologies, but it's a big one and totally affects the way you gain--or lose--control over positioning consistently when using CSS.

The Box Model issue has been described very well by Tantek Çelik, the lead developer for Microsoft's Macintosh IE. I'll paraphrase from Çelik's explanation as to the box model problem, which has to do with the differences in the way user agents calculate borders and padding. Consider the following CSS:

```
#box1  {
     border: 10px solid red;
     padding: 20px;
     background: blue;
     width: 300px;
}
```

This CSS would create a box that has a 10-pixel border to each four sides, padding around the entire box to a measurement of 20 pixels, and a set width of 300 pixels. If you're calculating the box model properly, you would add border and padding measurements to the 300-pixel content area, not subtract from them, making the content area small.

To properly calculate the total width, including content, border, and padding:

10 pixels left border +
20 pixels left padding +
300 pixels content area +
20 pixels right padding +
10 pixels right border =

The box should be a total of 360 pixels wide. But misinterpretations of the box model place the border and padding inside the defined content width. So, if you define a box to have 300 pixels and then any borders and padding are subtracted from your content area, that area is unfairly minimized. A browser that improperly manages this will calculate the box as follows:

300 pixels content area –
20 pixels left padding –
10 pixels left border –
20 pixels right padding –
10 pixels right border =

The content area of the box is now 240 pixels wide, and the total width of the box is 300 pixels. This means you as a designer are now totally frustrated in trying to position and

present a given element consistently, and to do so you're going to likely have to rely on a complicated hack known as (of course) "The Box Model Hack."

Here's where deep breaths and repetitions of "OM" come in. Put the Xanax DOWN!

The Hopeful Solution

Çelik, studying the problem, recognized that no browser could afford to move ahead with more compliant and consistent technologies without allowing for reasonable backward compatibility.

The solution Çelik devised was to split the browser's capabilities into two modes: Quirks mode and Compliance (or "standards") mode. Quirks mode is the implementation of rendering engines in use that manage non-standard markup--essentially the same forgiving rendering that we've relied upon for years--forgiving of our trespasses as well as those of our tools, but of course incredibly inconsistent as a result.
Compliance mode, on the other hand, is a streamlined standards-compliant rendering engine, allowing for faster, more accurate, and more *controlled* rendering of your designs. In fact, you can overcome the entire Box Model fiasco by switching IE 6 into Compliance mode, which repairs the box model problems of days past.

So how do you tell a browser which mode to use?

By incorporating the correct DOCTYPE into your document, of course!

Not So Fast

Now you see the reasons as to why having two modes and using DOCTYPE Switching (the name of this interesting technology) makes sense to those designers seeking control and calm. And who doesn't seek control and calm? But there's another problem, and that's that browsers with DOCTYPE Switching technology rely on *specifically formed DOCTYPE declarations* in order for proper switching to occur.

DOCTYPE declarations can be written in any number of ways. The default DOCTYPE that Dreamweaver MX uses is okay (and MX 2004 is a whole lot better). There's nothing *wrong* with it in any technical sense - but there is something wrong with it when it comes to DOCTYPE Switching technology. There are some very specific DOCTYPEs that you must use in order to kick the browser in question into Compliance mode, and I've provided a link in the *RESOURCES* sidebar to help you define which ones should be used.

The mechanism of DOCTYPE switching is, at its core, fairly sensible and straightforward:

- Documents with older or Transitional DOCTYPEs, poorly formed DOCTYPES, or no DOCTYPE at all are displayed using Quirks mode, and will be interpreted with the legacy bugs and behaviors of version 4 browsers

- Documents with properly formed HTML Strict or XHTML DOCTYPEs are displayed using Compliance mode. This mode follows W3C specifications for HTML, CSS, and other layout languages as closely as possible

Of course, Netscape Navigator 4.x came long before DOCTYPE switching was even conceived, so it should be assumed to always be in quirks mode (and a buggy form of it at that). Opera 6 and earlier does not bother with DOCTYPE switching, and should be assumed to be in standards mode since Opera has been purposely developed with standards in mind. Note that it may still have bugs, but Opera's behavior is very close to the standards modes of other browsers.

Modifying DOCTYPEs in Dreamweaver MX

There are some very easy ways to modify DOCTYPEs in Dreamweaver MX. I describe two of them here.

The first is using a Macromedia Exchange Tool called *Insert HTML Doctypes, V2.0.6.* and was authored by Jerry Baker. Once installed, simply select Modify > Document DTD within Dreamweaver MX, and then select the DTD you'd like to modify the current DTD to. The DTDs within this tool are accurate.

If you'd like to have your default Dreamweaver MX HTML page be HTML 4.01 strict (rather than the transitional default), with a correct DOCTYPE for DOCTYPE Switching, you can make the change directly by modifying the Basic template.

Note: you do not need to modify any Dreamweaver MX XHTML DOCTYPEs as they all appear to be in order.

To make the change:

1. Select File > Open.

2. Locate the Macromedia Dreamweaver folder on your hard drive. You should see a subfolder titled Configuration.

3. Open the Configuration folder, and look for another subfolder titled DocumentTypes. Open this folder.

4. Look for another subfolder called NewDocuments. Open this folder. Look for the file default.html.

5. Open default.html. In Code view, highlight this line:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

6. Replace it with this DOCTYPE :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```

7. Select File > Save.

The default HTML page is now properly marked up, and you won't need to make this change again.

Making the Switch

While incredibly useful for authors, DOCTYPE switching might have remained no more than a curiosity had it only been implemented in IE5 for the Macintosh. Happily, it has since been adopted by a slew of contemporary browsers including all recent versions of Opera, Netscape, and the seemingly ubiquitous IE.

RESOURCES

About the Box Model

Box Lessons: CSS Workarounds for Browser deficiencies by Owen Briggs defines the problems and provides excellent resources,
http://www.thenoodleincident.com/tutorials/box_lesson/

About DOCTYPE Switching

Doctype switching and standards compliance: An overview by Matthias Gutfeldt. Provides technical details and resources about the switching technologies discussed in this chapter,
http://gutfeldt.ch/matthias/articles/doctypeswitch.html

Doctypes and their respective layout mode, by Matthias Gutfeldt. This table shows a variety of DOCTYPEs and which layout mode they'll invoke in a range of browsers,
http://gutfeldt.ch/matthias/articles/doctypeswitch/table.html

Validators

Go to http://validator.w3.org/ and see how well your pages validate!

Macromedia Exchange

For extended features on the Macromedia Exchange, including the *Insert HTML Doctypes* tool, see http://exchange.macromedia.com/

Where Now?

Now you've learned the foundations of CSS. You can

- Give legacy sites a hybrid layout by preserving tables, but use CSS for all other styles
- Strip away tables to get maximum separation of style and content.
- Make 2 column and 3 column sites using only CSS.
- Offer alternate stylesheets for aesthetic choices, or for accessibility reasons
- Deal successfully with dinosaur browsers, so that at least users receive all your content

So where now? The best thing to do is experiment. Experiment with the layouts used in the chapters. There are plenty of resources on the web offering free Style Sheets. Some of them are

- http://www.bluerobot.com/web/layouts/
- "boxes" by Owen Briggs
- "layouts" by Eric Costello
- much inspiration, and a great way to see what does what at the CSS Zen Garden

As you'll have realised by now – and will soon be cursing about - unfortunately not all browsers implement the whole CSS spec (or, not as you'd expect it). Here's some browser compatibility charts:

- Peter Paul Koch's Quirksmode
- Codebitch charts
- http://centricle.com/ref/css/filters/

There's times when you just can't get a regular CSS file to render properly across all your target browsers, and might need to turn a browser's bugs to your advantage with a CSS "hack". Here's a great compendium of CSS filters and hacks:
http://www.dithered.com/css_filters/index.html

As CSS is becoming primetime, tips and cool stuff is still being developed. At the time of writing, there are a number of great sites that look at new ways to use CSS. We suggest, for starters,

- A List Apart
- Zeldman.com
- StopDesign by Doug Bowman (who designed www.wired.com)

The "Accessibility Toolbar" is a plug-in for IE/ Win that shows structure of the web site you're visiting. There's a free tutorial on using it on DMXzone. Please note, that when looking at

other people's css, it's not legal to copy it and use it unless comments in the file specifically allow it.

In short: experiment and learn!

About DMXzone

History of DMXzone

DMXzone was founded in Feb 2001 by **George Petrov**. It was then called UDzone after the Macromedia product UltraDev that preceded Dreamweaver MX. By April 2001 we'd already been asked by Macromedia to speak at the Macromedia UCON 2001 conference in New York. Since then, we've grown to over 150,000 registered members of all levels and locations, who come together to share knowledge and learn from each other. We are an independent community and are in no way connected with Macromedia, the makers of Dreamweaver MX.

In May 2003, we launched our very successful Premium Tutorials track, publishing professionally written tutorials by a team of authors for an affordable price every day, as we ourselves were tired of shelling out lots of money for computer books full of redundancy and newbie's explanation. This premium track runs alongside the free content submitted by members.

What do we do

Membership of the community is free. You can view most content on the site without registering, but when you become a member you can add your own articles, tutorials, news items, extensions, opinion polls and participate in the forums. To purchase extensions or download free extensions, you need to become a member.

The DMXzone Team and Manager Team consists of professionals and volunteers who work hard to bring you the extensions that you are asking for, give you the support that you need when you have questions and to bring you the latest information pertaining to web development. We like to encourage our visitors to actively participate, that is why we organize competitions, run opinion polls, let you rate articles, extensions and tutorials and let you add your own articles.

CSS Web Sites with Dreamweaver MX

Everything you ever wanted to know about Web Standards, CSS and Dreamweaver but were afraid to ask

This book will teach you CSS in a practical, project-based manner. It's unique in that it's for people who know their way around Dreamweaver, showing you how to make CSS sites using Dreamweaver as your development tool.

You'll learn by doing. No previous CSS knowledge is assumed. We look at the basics of cutting down presentational HTML, then removing tables, move on to CSS for positioning, and then the techniques used to make some common layouts (two columns, three columns, and so on), using CSS to style boxes, borders, margins, lists, photo albums - always concentrating on the practice rather than the theory.

Both authors are established Dreamweaver experts, who are members of the Web Standards project and regular writers for DMXzone.com, a large independent community of Dreamweaver developers.

Rachel Andrew has worked in the industry as a webmaster, technical project manager and senior web and now runs her own company 'edgeofmyseat.com', providing complete web solutions and outsourced development. Rachel serves on The Dreamweaver Task Force for the Web Standards Project.

Molly E. Holzschlag works with a group of other dedicated Web developers and designers to promote W3C recommendations. She also teaches Webmaster courses for the University of Arizona, University of Phoenix, and Pima Community College. She wrote the very popular column, Integrated Design, for Web Techniques Magazine and spent a year as Executive Editor of WebReview.com.

CSS and Design

ISBN 90-77397-02-7