CSS in a Bottle, v.2

An introduction to Cascading Style Sheets (CSS), intended for employees of Automark Web Services
by David Stiller
(Last updated 10/14/2002)

Introduction

If this is your first foray into CSS, either through Automark's eyes or your own, this document will clarify definitions, dispel myths, and excite you into visually enhancing your HTML code in a powerful and economical manner. If this is your second go-around, this document will overhaul your familiar habits. The increased efficiency is worth the challenge.

CSS Defined

According to the W3C, "Cascading Style Sheets (CSS) is a simple mechanism for adding style (*e.g.* fonts, colors, spacing) to Web documents." As described by WebMonkey.com, CSS is "an elegant cousin to HTML," and for good reason. Written in simple text—just like HTML—CSS dramatically expands and improves your formatting control over web content. You are freed from the limited font sizes of HTML (the mere `<font size="1">` through `<font size="7">`). You can establish line spacing (leading), alignment, padding, borders, layers, colors, and more. CSS reduces download times by thinning out HTML and governs whole sites with a single file.

Basics

Style Attribute

At its most basic, you can add CSS to any given line of HTML by including the `style` attribute in a tag. This is effectively like using the `<font>` tag, but current W3C recommendations favor CSS. **NOTE:** Automark does not generally use the style attribute method for DCMS because it is not conducive to template-driven sites, but it's good to learn. (See Selector Rules below for attribute syntax.)

Style Attribute

```
<p>This text appears plain.<p>
<p style="color: #00FF00; font-weight: bold;">This text appears
  green and bold.<p>
```

Style Tag

To apply attributes to *all instances* of a given tag—which is more likely and more powerful—just convert the `style` attribute to a tag itself and place it inside the `<head>` tag of your HTML document, then specify which HTML tags you'd like to enhance by defining them with **selector rules**. You may specify as many selectors as you wish.

Style Tag

```
...
   <style>
      p {
         color: #00FF00;
         font-weight: bold;
      }
      b {
         color: #FF0000;
         font-style: italic;
      }
   </style>
</head>

<body>
<p>This text, and all P text, appears green and bold.  <b>This
  portion of the text appears red and italic, because of the
  properties set for B; it also appears bold because of the inherent
  styling of B.</b></p>
...
```

Selector Rules

A rule is comprised of several parts. In the case of selector rules, the first part is the **selector**, which represents the HTML tag to be enhanced. The selector is followed by a number of **declarations** surrounded by curly braces (`{ … }`), which are further comprised of two parts, **properties** and their **values**. These pairs are separated by a colon (`:`) and declarations are separated by a semi-colon (`;`).

Anatomy of a Selector Rule

```
p {  ← selector
  color: #FF0000;
  font-weight: bold;
}  ↑ declarations
     (properties: values;)
```

Declaration Properties

How many properties are there? In truth, many—but not all are available to all browsers. In general, Automark sticks to a "safety" group of properties which are valid in Internet Explorer and Netscape 4.x and 6+, with a few exceptions that work in Internet Explorer only, or in IE and Netscape 6+. These exceptions are allowed because they don't actually "break" anything in Netscape 4.x, but merely fail to display.

Our in-house tool of choice for CSS is TopStyle. This application features "intellisense" code hints and color formatting, and offers convenient segregation of properties by browser acceptance and usage category. TopStyle calls these groupings "style definitions" and Automark uses the "Netscape 4 and IE 4" style definition.

Pause for Inheritance

Before we forge ahead, let's consider the issue of inheritance. In the above Style Tag example, note that the properties of the `b` selector's declarations are passed to all `<b>` tags on the page—you could say the `<b>` tags *inherit* these properties. Note also that although the `b` selector does not specify font weight, all `<b>` tag content is still bold, because a normal `<b>` tag is bold intrinsically. Therefore:

- Inheritance passes all properties to its children that are not already specified otherwise.

If this example had included the declaration `font-weight: normal;` the bold would have been overridden.

Valuing and Extending HTML

Automark's current approach to CSS acknowledges the formatting inherent in HTML, then extends it. Where possible, use what's already there. The `<b>` tag, for instance, renders bold whatever text it surrounds. The `<h>` tags render text as bolded headings, and depending on the number value specified, headings of a respective font size.

```
HTML Bold
This text appears plain.
<b>This text appears bold.</b>

HTML Heading
<h1>Bold, largest heading.</h1>
<h2>Bold, slightly smaller.</h2>
<h3>Bold, smaller Still, and so on.</h3>
```

By tapping into these existing characteristics, we can reduce certain aspects of CSS coding. For example, let's define a header style from scratch, then see where we can consolidate. **NOTE:** This example uses a different kind of rule (see Class Rules below), but don't be thrown by it—obviously, there is no `<exampleHeader>` tag.

```
CSS Header Style from Scratch
.exampleHeader {
   font-family:  Arial;
   font-size:  20px;
   font-weight: bold;
}
```

We want a header in 20px Arial with a bold font weight, and that's what these declarations provide. If we had extended an `<h>` tag instead, we could have omitted the

`font-weight` property since it is built into the `<h>` tag innately. In this case we only save one line, but the combined result of many saved lines can make a difference. We have the additional benefit of "insurance" in case the CSS fails to load: since the `<h>` tag actually is a header, it will look like a header even without the style enhancements.

CSS Header Style that Extends HTML

```
h1 {
   font-family:  Arial;
   font-size:  20px;
}
```

The Next Step

Class Rules

Sometimes selector rules aren't enough. Think of a table with alternating row colors: by using selector rules, at best you could define the properties for `<th>` and `<td>` tags alone, which is simply not enough control (all the `<td>`s would look the same).

Class rules provide the answer. They are essentially "custom" rules that can apply to any number of HTML tags. Class rules are laid out the same as selector rules, except the class name itself is preceded by a dot (.) and you must reference the rule in your HTML tags with a `class` attribute. Inheritance, as described above, is still in effect; that is, if you apply a class for red coloring to a `<p>` tag its content would be red, while the same class applied to a `<b>` tag would display its content as red and bold, because the content inherits both the declarations of the class and the inherent style of the tag in which it resides. For this reason, depending on intended results, it is prudent to state explicitly which properties to define in a class rule's declarations (*i.e.*, specify `font-weight: normal;` where you want to avoid bold in all cases, regardless of the HTML tag's innate styling).

Anatomy of a Class Rule

```
.myClass {  ← class
  color: #FF0000;
  font-weight: bold;
}  ↑ declarations
     (properties: values;)
```

Class Rule Referenced in HTML Tags

```
<p class="example">This would pick up the properties of the
  declarations in the .example class rule.</p>
<td class="example">This would do the same.</td>
```

External CSS

To get the widest reaching control over styling, it's best to draw from rules that are completely separate from the HTML document. This is actually quite easy: remove the rules listed in the `<style>` tags in the document's head and paste them into a new text file. Save the text file as [*filename*].css. Now link out to this file from the HTML page with a `<link>` tag in the document's head.

Linked External CSS File

```
...
   <link rel="stylesheet" type="text/css" href="[filename].css">
</head>
```

Since they are simply text, CSS files may actually be saved with any extension. At Automark, we save CSS files with an .asp extension to allow for the possibility of future dynamic treatment. As of this writing, the total number of CSS files used in DCMS is undetermined, but you are likely to at least see cssmaster.asp. This file would be linked out as follows:

Linked External CSS File with ASP Extension

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

Rule Grouping

If you can kill two birds with one stone, do it. Selectors and classes whose rules feature identical properties can be combined with a comma. Let's take the example of a class rule intended for an anchor tag. Anchor tags, since they represent hyperlinks, are good candidates for **pseudo classes**, which include "states" of a rule. The following is syntactically correct, but it takes up 24 lines.

Superfluous Class Rules

```
.navTop:link {
   font-family: Verdana;
   font-weight: bold;
   color: #000000;
   text-decoration: none;
}
.navTop:visited {
   font-family: Verdana;
   font-weight: bold;
   color: #000000;
   text-decoration: none;
}
.navTop:active {
   font-family: Verdana;
   font-weight: bold;
   color: #FF0000;
   text-decoration: underline;
}
.navTop:hover {
   font-family: Verdana;
   font-weight: bold;
   color: #FF0000;
   text-decoration: underline;
}
```

Pseudo Classes

Certain tags, like anchors (**<a>**), feature a number of "states" (for example, the mouse is hovering over the content of this tag; now it is clicking, etc.). In CSS, pseudo classes allow you to set styles specific to these states. Pseudo classes are preceded with a colon (:) and follow selectors or classes in a rule.

Note that the properties overlap in many of these declarations. In fact, the only visual differences between the state of `:link` and `:visited` classes versus the state of `:active` and `:hover` classes are the lack or presence of an underline and a change in color. These rules cause their respective anchor tags to display an underline and appear red while the mouse hovers or clicks on the tag's content, but not otherwise.

The consolidated version is clearly more efficient, weighing in at a mere 12 lines (see next page).

Consolidated Class Rules

```
.navTop {
   font-family: Verdana;
   font-weight: bold;
}
.navTop:link, .navTop:visited {
   color: #000000;
   text-decoration: none;
}
.navTop:active, .navTop:hover {
   color: #FF0000;
   text-decoration: underline;
}
```

The first rule covers `.navTop` only (no pseudo classes), which covers all states and sets the properties common to all declarations. The `:link` and `:visited` states are grouped, since they're the same, as are `:active` and `:hover`. In each case, we have specified only the information that changes.

If `navTop` anchors are intended to inherit the base font of the whole document, we could omit the `font-family` property and shorten our code yet again.

Inheritance Revisited

Matter of Font

Headings provide a strong case for understanding and effectively using inheritance. In the past, every DCMS heading (typically a class rule) would define its own font size. The problem with this approach is that it left us with "hard coded" headings, which meant that if we changed the size of the base font, we likely had to change the sizes of our headings to match.

In general, headings follow a predictable decrease in size from first-order down. Rather than specifying these sizes manually, Automark now uses the **em** measurement on fonts in all cases except the base, which uses px (pixel) measurement. Theoretically, you could assign a font size of 12px to the `<body>` tag (the base) and expect the rest of the document to follow suit unless otherwise specified, everywhere from standard paragraphs to forms and tables, lists, and so on. As it happens, not all browsers recognize this inheritance. To cover the bases, we rely on rule grouping to set a base font.

Rule Grouping to Set a Base Font

```
body, div, p, label, td, th, ol, ul {
   font-family: "Times New Roman";
   font-size: 12px;
   font-weight: normal;
   color: #000000;
   text-decoration: none;
}
```

Now we can safely drop text anywhere in the document and confidently expect it to appear "normal" until we specify something else. If we set

Ems

An em is a typographical unit of measurement that applies a percent-based adjustment to a default standard. If the base font is 12px and a heading is set to 2em, the heading will size to 24 pixels (200% of the base). A heading of 1.5em would be 150% of the base size. You may set ems at less than zero, such as 0.5em (50%).

headings to carefully calculated ems, we can change the base font size and see the headings update respectively without any effort. In the following example, <h1> tag content will always be 2 ems, or twice the size, of the base font, even if the base font changes.

Example of Em Usage

```
h1 {
   font-family: Verdana;
   font-size: 2em;
}
```

Contextual Inheritance

Rules that rely on **contextual** inheritance narrow the focus of CSS based on the placement of HTML tags. To accomplish this, define two selectors or classes (or a mix) and separate them with a space.

Contextual Selector Rule

```
p b {
   background-color: #FFFF00;
}
```

This refers to any <b> tag that falls anywhere within a <p> tag.

```
<b>This would not pick up the CSS.</b>
<p><b>This would, because B appears within P.</b></p>
<p><small><b>This would too, because B still appears within
  P.</b></small></p>
```

Contextual Class Rule

```
.example .highlight {
   background-color: #FFFF00;
}
```

This refers to any tag whose class="highlight" that falls anywhere within a tag whose class="example".

```
<span class="highlight">This would not pick up the CSS.</span>
<p class="example"><span class="highlight">This would, because the
  "highlight" tag appears within the "example" tag.</span></p>
<p class="example"><blockquote><span class="highlight">This would
  too, because the "highlight" tag still appears within the "example"
  tag.</span></blockquote></p>
```

It is important to note that with contextual inheritance, it makes no difference how many HTML tags fall between the two selectors. As long as the second occurs *somewhere* within the first, you're okay. This provides significant consolidation for navigation. You could define properties for all <a> tags, for example, which fall within a <table> tag whose class equals the style in question. In the past, Automark added a class rule to every <a> or <td> tag, but now we can get away with a single reference.

In the example that follows, all of the anchor tags may be changed with a single tweak to the class attribute of the <table> tag (see next page). In addition, the volume of code text in the HTML has been reduced.

Consolidated Navigation

```
.navSub01 a { [declarations] }
...
<table class="navSub01">
<tr>
   <td>
      <a href="somewhere01.asp" ... >Link 01</a>
      <a href="somewhere02.asp" ... >Link 02</a>
      <a href="somewhere03.asp" ... >Link 03</a>
      <a href="somewhere04.asp" ... >Link 04</a>
      <a href="somewhere05.asp" ... >Link 05</a>
   </td>
<tr>
<table>
```

Contextual rules can be multi-stacked to produce a remarkable effect in lists. By extending the `<ul>` tag, for instance, you can replace the bullets with graphics that change according to list depth (first-order circles, sub-order squares, sub-sub-order bagels, etc.). Contextual inheritance offers this kind of muscle without touching the HTML if you go the route of selector rules.

Contextual Selector Rules that Extend a List

```
ul {
   list-style-image:  url(circle.gif);
}
ul ul {
   list-style-image:  url(square.gif);
}
ul ul ul {
   list-style-image:  url(bagel.gif);
}
```

Child Inheritance

Rules that rely on **child** inheritance are similar to contextual rules. They also narrow the focus of CSS based on placement of HTML tags, but HTML tags referenced by child rules must be immediately adjacent, with no tags between. To accomplish this, define two selectors or classes (or a mix) and separate them with a greater-than sign (>).

Child Selectors

```
p > b {
   background-color: #FFFF00;
}
```

This refers to any `<b>` tag that falls immediately within a `<p>` tag.

```
<b>This would not pick up the CSS background color.</b>
<p><b>This would, because B is immediately within P.</b></p>
<p><small><b>This would not, because another tag comes between B and
P.</b></small></p>
```

Coding Guidelines

Rule Appearance

Rules may be collapsed into a single line without affecting usefulness. TopStyle allows you to toggle between these formats effortlessly.

Expanded Rule

```
h1 {
   font-family:  Arial;
   font-size:  20px;
}
```

Collapsed Rule

```
h1 { font-family:  Arial; font-size:  20px; }
```

Commenting

Comments in CSS files clarify formatting choices for fellow developers. To include text that is not part of a CSS rule, simply sandwich your comments between the proper comment demarcation, `/*` ... `*/`.

Comment Example

```
/* This is a comment and will not appear in the style. */
.example {
   background-color: #FFFF00;
}
```

Pratfalls

In order to provide cross-browser compatibility, follow the guidelines below.

- Class names cannot contain spaces. If they do, the rule will be misconstrued as an example of contextual inheritance.
- Class names cannot contain underscores (_).
- Class names are case sensitive. At Automark, we use Hungarian Notation (or Camel Case)in multi-word classes. When combining words, remove spaces and capitalize the first letter of each word starting with the second word. For example, “my class rule” becomes “myClassRule.” Acronyms are treated as words (“my CSS” becomes “myCss”).

Additional Resources

Websites

Index DOT Css
http://www.blooberry.com/indexdot/css/index.html
All the properties (font-size, font-family, text-decoration, etc.), browser compatibility, and much more.

WCSchools.com
http://www.w3schools.com/css/default.asp
Their motto says it: “the best things in life ARE free.”

Official CSS Website
http://www.w3.org/Style/CSS/

Developer Netscape Issues
http://intra.automark.net/
Research & Development > R&D Resources > Developer Netscape Issues
Written in-house, this document spells out hazards along the path to cross-browser compatibility.

Reference

Simply by using TopStyle, you have access to Help files that can assist greatly in understanding CSS syntax. Dreamweaver MX also provides reference materials from O'Reilly.