Style Sheets

Style sheets are powerful mechanism for adding styles (e.g. fonts, colors, spacing) to web documents. They enforce standards and uniformity throughout a web site and provide numerous attributes to create dynamic effects. Style sheets provide the capability to control the way in which HTML elements such as headings, paragraphs, and lists are laid out and displayed. They enable web page designers to use standard HTML elements in a limitless variety of new ways.

The advantages of a Style Sheet include the ability to make global changes to all documents from a single location. You can collect styles for certain types of projects and save them in style sheets. Style sheets give you complete flexibility and control in the way that the pages of your web applications are presented.

Style sheets are said cascade when they combine to specify the appearance of a page and hence Cascading Style Sheets or simply CSS.

In a nutshell, CSS provides a means for web authors to separate the appearance of web pages from the content of web pages. CSS is based on rules that select an HTML element and declare style characteristics for the element. You can state set of rules, known as style sheets

You can use style sheets in three ways, depending on your design needs:

- By **linking** to a style sheet from your HTML file. This method allows you to change the appearance of *multiple Web pages* by tweaking a single file. i e. include them in a separate document known as an external style sheet (External Styles heet)

For example, if your style sheet is called MYSTYLES.CSS and is located at the address http://www.company.com/mystyles.css, you would add the following to your Web page, within the <HEAD> tag:

```
<HEAD>
<TITLE>Title of article</TITLE>
<LINK REL=STYLESHEET HREF="http://www.company.com/mystyles.css" TYPE="text/css">
</HEAD>
```

- By **embedding** a style sheet in your HTML file. This method allows you to change the appearance of a *single Web page* by tweaking a few lines(internal style sheet)

To embed a style sheet, you add a <STYLE> </STYLE> block at the top of your document, between the <HTML> and <BODY> tags. This allows you to change the appearance of a *single* Web page. The <STYLE> tag has one parameter, TYPE, which specifies the Internet Media type as "text/css" (allowing browsers that do not support this type to ignore style sheets). The <STYLE> tag is followed by any number of style definitions and terminated with the </STYLE> tag.

The following example specifies styles for the <BODY>, <H1>, <H2>, and <P> tags:

```
<HTML>
<STYLE TYPE="text/css">
<!--
  BODY {font: 10pt "Arial"};
```

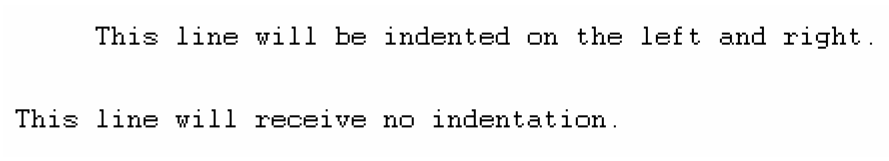
```
    H1 {font: 15pt/17pt "Arial";
        font-weight: bold;
        color: maroon}
    H2 {font: 13pt/15pt "Arial";
        font-weight: bold;
        color: blue}
    P  {font: 10pt/12pt "Arial";
        color: black}
-->
</STYLE>
<BODY>
...
</BODY>
</HTML>
```

- By adding **inline** styles to your HTML file. This gives you a quick way to change the appearance of a *single tag*, a *group of tags,* or a *block of information* on your page. (inline styles)

Inline style definitions affect individual occurrences of a tag. These are embedded within the tag itself using the STYLE parameter. The following HTML code indents a specific <P> tag:

```
<P STYLE="margin-left: 0.5in; margin-right: 0.5in">
This line will be indented on the left and right.
<P>
This line will receive no indentation.
```

Here's the result (this display requires Internet Explorer 3.0):



```
     This line will be indented on the left and right.

This line will receive no indentation.
```

The attributes that can be specified to the <style> tag are **Font Attributes,Color and Background attributes; Text Attributes,Border attributes,Margin Attributes and List Attributes**

Quick reference for the above mentioned attributes

Attribute	Description	Values	Example
Font Attribut			
font-size	Sets size of text.	points (pt) inches (in) centimeters (cm) pixels (px)	{font-size: 12pt}
font-family	Sets typeface.	typeface name font family name	{font-family: courier}
font-weight	Sets thickness of type.	extra-light	{font-weight: bold}

		light demi-light medium demi-bold bold extra-bold	
font-style	Italicizes text.	normal italic	{font-style: italic}

Background Attributes

color	Sets color of text.	color-name RGB triplet	{color: blue}
background-color	Sets the elements backgrounds color	color-name RGB triplet	{background-color: blue}
background-image	Sets the elements background image	url or none	{background-image:url(123.gif)}
background-repeat	Sets how elements background image repeats throughout the page	Repeat-x Repeat-y Repeat No-repeat	{background-image:url(123.gif);background-repeat:no-repeat}

Text Attributes

text-decoration	Underlines or otherwise highlights text.	None underline italic line-through	{text-decoration: underline}
text-align	Sets justification.	left center right	{text-align: right}
text-indent	Sets distance from left margin.	points (pt) inches (in) centimeters (cm) pixels (px)*	{text-indent: 0.5in}
text-transform	Applies a transformation to the text	Capitalize UpperCase lowercase	{text-transform:uppercase}
vertical-align	Sets the elements vertical Position.	Baseline,sub,super, top,middle,buttom	{vertical-align: sub}

Margin Attributes

margin-left	Sets distance from left edge of page.	points (pt) inches (in) centimeters (cm) pixels (px)*	{margin-left: 1in}
margin-right	Sets distance from right edge of page.	points (pt) inches (in) centimeters (cm) pixels (px)*	{margin-right: 1in}
margin-top	Sets distance from top edge of page.	points (pt) inches (in) centimeters (cm) pixels (px)*	{margin-top: -20px}
margin-bottom	Sets distance from bottom edge of page.	points (pt) inches (in)	{margin-bottom: -2cm}

		centimeters (cm) pixels (px)*	
margin	Set all margin at a time	points (pt) inches (in) centimeters (cm) pixels (px)*	{margin: 2pt}

Border Attributes

border-style	'border-style' property sets the style of the four borders	none , dotted, dashed , solid , double , groove , ridge , inset , outset	{ border-style: solid }
border-color	'border-color' property sets the color of the four borders	color-name RGB triplet	{border-color: blue}
Border-width	sets the width of an element's border	thin , medium , thick , length	{border-width:thick}
Border-top-width	sets the width of an element's top border	thin , medium , thick , length	{border-top-width:thick}
Border-buttom-width	sets the width of an element's buttom border	thin , medium , thick , length	{border-buttom:thick}
Border-left-width	sets the width of an element's left border	thin , medium , thick , length	{border-left-width:thick}
Border-right-width	sets the width of an element's right border	thin , medium , thick , length	{border-right-width:thick}
Border-top	shorthand property for setting the width, style and color of an element's top border.	Specifies width color and style	{ border-top: thick solid red }
Border-buttom	shorthand property for setting the width, style and color of an element's buttom border.	Specifies width color and style	{ border-buttom: thick solid red }
Border-left	shorthand property for setting the width, style and color of an element's left border.	Specifies width color and style	{ border-left: thick solid red }
Border-right	shorthand property for setting the width, style and color of an element's left border.	Specifies width color and style	{ border-right: thick solid red }
Border	'border' property is a shorthand property for setting the same width, color and style on all four borders of an element.	Set all property at once	{border: solid red}

List Attributes

List-style-image	sets the image that will be used as the list-item marker	url	{list-style-image: url(http://png.com/ellipse.jpg)}

List-style-type	used to determine the appearance of the list-item marker	disc , circle , square , decimal , lower-roman , upper-roman , lower-alpha , upper-alpha , none	`{ list-style-type: lower-alpha }` `/* a b c d e etc. */`
List-style	Used for setting all the list properties at a time		{list-style: upper-roman url(12.gif)}

Understanding Style Rules

Style sheet rules are easy to interpret. The following style sheet shows a simple style rule for the <P> element. Note that the style rules are contained in the <STYLE TYPE=”text/css”> element in the document’s <HEAD> section:

```
<HEAD>
      <TITLE>Sample Document</TITLE>
      <STYLE TYPE=”text/css”>
            P {COLOR: BLUE; FONT-SIZE: 24pt}
      </STYLE>
</HEAD>
```

This rule sets all <P> elements in the document to blue 24-point text. Style rules are composed of two parts: a **selector** and a **declaration**. The selector determines the element to which the rule is applied. The declaration details the exact property values. It can contain quite a number of Properties, the individual pieces of style to be applied to the selected element.

CSS Selection Technique

You must apply the style rules you build to the elements in the document. The power in CSS comes from the different methods of selecting elements. You can choose from a variety of selection methods including:

- Selecting multiple elements
- Selecting by context
- Selecting with the CLASS attribute
- Selecting with the ID attribute

More complex selection involves the creation of artificial divisions, using the elements designed expressly for CSS:

- <DIV>   - Block Division
- <SPAN>   - Inline Division

The use of these elements, in combination with CSS, effectively allows you to create entirely new HTML elements that are specific to your working environment.

Selecting Multiple Elements

Using multiple selectors lets you use less code to accomplish the same results. For example, to make both <H1> and <H2> headings green, you could use the following rules:

```
<STYLE TYPE=”text/css”>
H1 {COLOR: GREEN}
H2 {COLOR: GREEN }
</STYLE>
```

These two rules can be expressed in a single rule statement using multiple selectors for the same property. Multiple selectors must be separated by commas:

```
<STYLE TYPE=”text/css”>
H1,H2 {COLOR: GREEN}
</STYLE>
```

Selecting by Context

A context-based selector lets you specifiy the exact context in which a style is applied. To specify that <I> elements appear blue only with <H1> elements, use the following rule:

```
<STYLE TYPE=”text/css”>
H1 I {COLOR: BLUE}
</STYLE>
```

This rule states that <I> elements appear blue only when they occur within an <H1> element.

Selecting with the CLASS attribute

The CLASS attribute lets you write rules and then apply them to groups of elements that you have classified. Basically, the CLASS attribute lets you define your own tags and then apply them anywhere you want.

To create a class, first declare it within the <STYLE> element. The period (**.**) flag character indicates that the selector is a class selector. Below is an example:

```
.MyQuote {COLOR: RED;}
```

Place this rule in the <STYLE> element:

```
<STYLE TYPE=”text/css”>
.MyQuote {COLOR: RED;}
</STYLE>
```

Next, use the CLASS attribute in the document. In the following example, the code defines the <P> element as a special class named MyQuote:

```
<P CLASS=”MyQuote”>Some Text</P>
```

The selected paragraph with display the style properties of the quote class. In this example, the text color of the paragraph is red.

Selecting with the ID attribute

The ID attribute lets you create specific exceptions to a style definition. For example, you specify some elements with on “CLASS” attribute and apply them a common style. Among those elements, if you want one of them have additional style then you can use “ID”. To create a ID, first declare it within the <STYLE > element. The hash (#) character indicates that the selector is a class selector. Below is an example:

```
#MyID {FONT-FAMILY:VERDANA;FONT-SIZE:12pt;}
```

Place this rule in the <STYLE> element:

```
<STYLE TYPE=”text/css”>
#MyID {FONT-FAMILY:VERDANA;FONT-SIZE:12pt;}
.MyQuote {COLOR: RED;}
</STYLE>
```

Next, use the ID attribute in the document. In the following example, the code defines the <P> element as a special ID named MyID:

```
<P CLASS=”MyQuote” ID=”MyID”>Some Text</P>
```

The selected paragraph with display the style properties of the quote class. In this example, the text color of the paragraph is red.

Anchor pseudo-classes

User agents commonly display newly visited anchors differently from older ones. In CSS1, this is handled through pseudo-classes on the 'A' element:

```
  A:link { color: red }       /* unvisited link */
  A:visited { color: blue }   /* visited links */
  A:active { color: lime }    /* active links */
  A:hover(color:black} /*link when mouse is over it*/
```

Eg:

```
<html<<head>
<style type=text/css >
a:link{color:#445566;text-decoration:underline; FONT-SIZE: 16px}
a:visited{color:#ff6633;text-decoration:underline; FONT-SIZE: 16px}
a:active{ color:#336699;text-decoration:none  FONT-SIZE: 16px}
a:hover{ color:#ffcc22;text-decoration:none; FONT-SIZE: 18px}
</style></head>
<body>
 <a href=”test.html”>click here</a>
</body></html>
```

Introducing DIV

DIV stands for *logical DIVision.* This tag is often used to divide a Web page into sections.Each section has its own style. The DIV element is used to mark up divisions in a document. DIV is called a block-level element because you use it to affect a complete block of text. It can enclose headers, paragraphs, tables and other block-level elements, and any combination of these elements.

```
<HTML>
<HEAD>
</HEAD>
<BODY>

This text displays normally, except for the words
 <DIV style="position:absolute;top:100 px;left:100 px;color:yellow;background-color:black">
<P> This paragraph appears in yellow lettering on a black  background. </P>
 This text is also inside of the DIV block, so it displays in
 yellow on black
</DIV>

  <DIV style="position:absolute;top:300 px;left:300 px;background-color:yellow">
<P>
<table border=1 color="white">
<tr><td>test1</td><td>test2</td></tr>
<tr><td>test3</td><td>test6</td></tr>
<tr><td>test5</td><td>test6</td></tr>
</table>
</p>
this is a test within div
</DIV>
Agai n this is a normal test

</BODY>
</HTML>
```

JavaScript Style Sheets (JSS)

In addition to supporting CSS, Netscape communicator also supports a JavaScript-based approach to style sheets, referred to JavaScript Style Sheets or JSSS for short. JSSS support the styles provided by CSS and has the advantage of making these styles available as JavaScript properties. This advantage enables style properties to be created, read and updated via JavaScript scripts.

Unlike CSS, the "text/javascript" value is used to identify a JavaScript style sheet in the <STYLE> element as shown below:

```
<STYLE TYPE="text/javascript">
      /* Tag level style assigning
      Syntax: tags.HTMLTag.property = value
      */
      document.tags.BODY.backgroundColor        = 'yellow';
      document.tags.H1.color  = 'red';

      /* Class level style assigning
      Syntax: classes.className.tag.property    = value;
      Tag can be : any HTML tag
            Or can be "all"
      */

      classes.MyQuote.all.fontFamily      = 'Arial';
      classes.MyQuote.H1.fontSize         = '36pt';

      /* ID level style assigning
      Syntax: ids.ID.property = value
      */

      ids.MyID.fontFamily     = "VERDANA";
      ids.MyID.fontSize       = "12pt";
```

```
        </STYLE>
```

Why should I use CSS?
Style sheets exist, above all, to enable the following principle to be put into practice.

Web pages should separate content from appearance

As a web developer, this means that the information in your web site should go into your HTML files, but HTML files should not contain information about how that information is displayed. And you've probably guessed by now that information about how the pages should appear goes into CSS files.

You might wonder what advantages this conveys. Why go to all of this trouble? Just a couple of advantages might give you an idea about why this approach has long been considered beneficial in areas of document management that have been around a lot longer than the world wide web.

The traditional HTML approach is to "hardcode" all of the appearance information about a page. So you want all your headings in courier, and at different point sizes to the sizes built into browsers? Then, for every heading, in every page, set the font size and face properties. You've got more than 100 pages? That is a lot of editing, and a lot of re-editing if you decide to modify the appearance of your pages later. And with all of that editing there is plenty of possibility for introducing errors.

With CSS, you can decide how headings should appear, and enter that information once. Every heading in every page that is linked to this style sheet now has that appearance. Want to make every heading of level 3 more obviously different from those of level 2? Edit the style sheet, and every such heading now has the altered appearance. How many hours' work (and potential errors) have you just saved?

Another major advantage involves the management of large, sophisticated sites. With cascading style sheets, whole organizations can share a small number of style sheets, ensuring consistency across the site with no need for constant updating and editing to accommodate changes.

Managing style at large sites
Imagine how the web site for a large organization, say a corporation, might be structured. As sites grow in complexity, individual divisions, departments, and workgroups become more responsible for their own section of a site. We can already see a potential problem - how do we ensure a consistent look and feel across the whole site?

A dedicated web development team can ensure that a style guide is adhered to. How can we ensure that each workgroup, department, and division also follows the guidelines? Won't the site end up as a dog's breakfast of mish mashed styles?

With traditional HTML, the answer is almost certainly yes. Individual style sheets make life a little better, as all of the style information is concentrated in a single location, easily edited and reviewed. However, we still have the organizational difficulty of ensuring that each style sheet is the same as all the others, that the latest version has been distributed to all and updated.

Then there is the logistical nightmare of trying to have each and every style needed by every group included in the one style sheet. Different departments and groups have different needs - the legal department will have different needs from the engineering department, but they will share the basic look and feel of the company's site.

This doesn't look promising. But Cascading Style Sheets in fact provide a mechanism, the cascade, which helps address these very problems.

Cascading style sheets do not have to stand alone. They can import style from one or more other style sheets. Style sheets which import from others are said to cascade from those style sheets.

Here is how we might use the cascading nature of style sheets to help solve the organizational nightmare we described above.

At the top level, the company could have a core style sheet, which defines the basic look and feel for the corporate site. It might include aspects such as font, color, background color and so on.

Each division would have a divisional style sheet. This does not have to reproduce the core style sheet, simply to add any styles specific to the division. To ensure that this style sheet also carries the core styles, it only has to import the core style sheet. It can do this simply be including an @import rule.

We can continue down the chain, with each new style sheet adding styles necessary at that level, and importing the style sheet above (there is no need to import all style sheets, as importing a style sheet imports any style sheets imported by that style sheet.)

Using the process, we create a cascade of style sheets. When a style sheet above another in the cascade is changed, these changes cascade automatically into the web pages which are linked to the lower one too. Our organizational nightmare is easily managed using the cascading nature of style sheets.