Cascading StyleSheets

Introduction to Cascading Style Sheets (CSS)

This document points out major features of the CSS system at a very high level and, is designed to be used in conjunction with parallel discussions and demonstrations. CSS is a rule based language designed to be called from an HTML, XHTML, or XML document to supply additional typographic styling events. Note that CSS is not an XML application but can be called from XML applications. At its most abstract level, I could describe CSS as: *a declarative, rule based pattern language*. (I would like to point out that this is also a good way to describe XSLT)

The file that contains CSS rules/instructions is called a “stylesheet” for the straight-ahead reason that if the browser follows these rules, then the affected document will be typographically altered from the browser’s defaults, that is, styled. The connection between a CSS stylesheet and a host HTML document (or an XHTML or XML document) is the presence of a call-out in the host referencing that CSS file.

The most compelling reason for stylesheets is that they *encapsulate or hide* presentation code or instructions from content. Further, the CSS stylesheets allow an author to do more than either HTML or XHTML can do natively, again due to the separation that allows additional processing.

Just What *is* a Stylesheet?

A style sheet is a declarative set of rules that are read by a rendering engine (such as browser software) in order to transform a source document (internally represented as a "tree" model) into a presentation document (internally represented as a "tree" model) that can be displayed to a user, usually visually, but aural style sheets are also supported by some experimental browsers. Here is a first quick look at a CSS stylesheet assumed to be in a file named `mystyle.css`. More detail follows later in this document. (a reference to this file would be called out within an HTML document by an author who wished the browser to honor those typographic requirements. The actual linkage code appearing in the HTML document would look like:

<link href=”mystyle.css” type=”text/css” rel=”stylesheet” />

).

Example CSS-1

```
/* mystyle.css (This is a comment line in a CSS file) */
h1 { font-family:Arial; font-size:16pt; }
h2 {font-family:Arial; font-size:14pt; }
p { font-family: "Times New Roman"; font-size:1em; }
```

The example above first displays a comment line, bracketed by "/*" and "*/", as is used in C, C++, and Java. Then three *rules* follow that will apply to those particular tags in any associated (X)HTML document. The body of the rule is bracketed by "{" and "}" symbols, called “braces”. The text preceding the body is called the "selector". The first rule, with a selector of "h1", means

that the content between the starting <h1> tag and the ending </h1> tags in the (X)HTML document, requests that the browser use *Arial* type face (if available, otherwise the closest available font, such as Helvetica). Arial is the common sans-serif font used on Windows platforms while Helvetica is common on a Mac. Additionally, the text is to be set in "16pt" size. (These are "printers points", at 72 points per inch)

Similarly, the second rule says that the content between the <h2> and </h2> tags in an associated (X)HTML document is to use the *Arial* font with a smaller point size of "14pt".

The last rule, with a selector of "p", says that the content between and tags should use the "Times New Roman" font with a font size that sets a size relative to the browser's default size. The use of relative measures is encouraged since these settings will scale according to the user's preferences and actions.

Keeping Content and Presentation Separate with CSS

At the same time W3C was standardizing HTML, some help in keeping content separate from presentation was offered in the form of another W3C complementary presentation language called Cascading Style Sheets (CSS). This language, introduced in 1994, was specifically designed to help "style" or present HTML pages. Now, a designer could declare the style of an HTML heading, for example, in a separate CSS document, called a style sheet. The style for a particular heading, for example, is declared in the form of a *rule,* that specifies all the presentation typographic events that are to be applied to all the matching headings in the HTML document. For example, the rule might specify that all *h1* first level headings in the HTML document should be set in the font family "Arial", bold, with a text point size of 16 points. (Note: The HTML file and the CSS style sheet are connected by a tag in the HTML file that specifies the name of the CSS style sheet to be accessed). In this "heading" case, the browser would read the HTML file and when it came to a *h1* first level heading, it would read the associated CSS style sheet and selectively override its own default settings for *h1* first level headings. The general result is that presentation details (kept in the CSS document) are separate from the content (kept in the HTML document).

Multiple Targets, Same Source Document

Another reason for the value of keeping the presentation separate from the content is the need to often present the same content on different client target devices or in different ways on the same device. In this case, it is the presentation that needs to be varied, and that is just what CSS is designed to do. (If the structure of the content is to be transformed, there is another language called XSLT, that can be run prior to the final styling by CSS). This need for some kind of transformation followed by presentation, occurs frequently when a user wishes to receive information in a way that matches their needs or constraints, such as a visually impaired reader or a user receiving information over a very limited cellphone. In both cases, the content may be the same (or abbreviated/transformed), but the presentation modes will be very different. Having the content in a separate document then allows that content to remain the same while several different style sheets could be applied to that same document thus rendering different presentations for different audiences or devices.

Origins and Evolution

An ancient origin of stylesheets is simply handwritten (or spoken) instructions accompanying a manuscript, telling a scribe how it is to be written (or chiseled). Later, the instructions told a printer how a manuscript was to be typeset. In more modern times, this set of instructions is called a "template" in Windows products or "formats" in other products, or simply "stylesheets", and directs a computer or a laser printer how a document is to be rendered.

Examples: DTP (Desktop Publishing), Word Processors (WP), Web Styling

Desk Top Publishing has always used stylesheets in order to transparently and immediately render text in a *What You See is What You Get (WISYWIG)*. Framemaker from Adobe, is an example DTP package that has built in style sheets that may be overridden by the document author. Similarly, word processors like Microsoft Word and others behave in analogous ways.

Browser Implementations

The browser manufacturers have not kept up very well with the evolving CSS specs. The current crop of browsers, IE 6.0 and Netscape 7.2, still don't implement all of the CSS2 specification that was finalized in 1998! So, the best an author can do is to use the CSS1 specification, which the current browsers *do* support, and test out which of the CSS2 features are mutually supported.

CSS1, CSS2, CSS3, Mobile CSS

The CSS specification is now at level CSS3, after beginning at CSS1 in 1996. Much has changed of course, and many new devices have caused a re-thinking of how specs are to be developed and issued. Currently, one of the most important approaches to spec development is to construct and issue them in a modular fashion. So, instead of one monolithic, complete spec, we are now seeing specifications for XHTML, CSS, and SVG for that matter, being issued in phases, and in *modules* that describe some coherent set of functionality. The new CSS3 spec is now completely modular as described by the W3C *Introduction to CSS* Working Draft of May 23, 2001. There is a section in there titled "Why Modules" that states:

> As the popularity of CSS grows, so does the interest in making additions to the specifications. Rather than attempting to shove dozens of updates into a single monolithic specification, it will be much easier and more efficient to be able to update individual pieces of the specifications. Modules will enable CSS to be updated in a more timely and precise fashion, thus allowing for a more flexible and timely evolution of the specification as a whole.
>
> For resource constrained devices, it may be impractical to support all of CSS. For example, an aural browser may be concerned only with aural styles, whereas a visual browser may care nothing for aural styles. In such cases, a user agent may implement a subset of CSS. Subsets of CSS are limited to combining selected CSS modules, and once a module has been chosen, all of its features must be supported.

Mobile CSS

As an example of this modularization, CSS now has a module called *Mobile CSS* that specifies a minimal set of properties, roughly comparable to the original CSS1 spec. In fact, it is this specification that forms the basis for this course's CSS property set.

Rule Structure

CSS is a Declarative Pattern Language

The basic elements of the CSS language are its *Rules*. A rule is composed of components as follows:

The **** diagram shows the CSS rule structure ***. First off, the whole structure is called a *Rule*. The *Rule* then decomposes into a *Selector* and a *Declaration Block*. The *Declaration Block* in turn is composed of a list of *declarations*. Within each of those *declarations* there is a *property* separated by a colon from its value. That value, in turn can be a list too. For example, in the figure, I show a *Selector* that will select all the *em* (that is, the "emphasis") elements . in the document.

Once the selector matches an *em* element, it then applies a color property value of "red" to the text and also makes the text "italic".

Selectors

Selectors allow you to specify which elements you would like to apply styling to. The following types of selectors will be considered:

- Type Selector - this selects elements in the document according to type. For example:

  `h1 { color:silver; }` . This matches any element *h1*, that is, a first level head, and makes its text "silver".

- Group Selector - this select a group of elements and applies the same declarations to all of them. For example:

  `h1,h2,h3,h4 { font-family:Arial; }` applies the same font "Arial", to all of the listed head elements.

- Descendent Selector - (Note: this was called a *contextual selector* in CSS1) this selects elements that are descendents of another element within the document tree. For example:

  `p em {color:red; }` selects *em* elements from the document and makes their text content red, if any ancestor is a *p*. Note that the p element can not only be a parent element, but could be a grandparent, great great grandparent and so on. Just so long as the *em* element can trace up through the document tree and find a *p*.

  As another example, you might have:
  `p.warning em {color:green; }` This specifies that if there is a paragraph in the HTML document that is specially marked as a "warning" paragraph (this would mean that it was marked as:

  `<p class="warning" > This is a warning paragraph text with subtext that <em> turns green </em> and on and on. </p>` , then if there is an tag nested within it, it's text will turn green!

- Universal Selector - this will select any element. For example:

  `* {color:purple; }` will cause all text in the document to be purple, unless overridden.

- Child Selector - this will select any element with a specific parent For example:

`p > strong {color:green; }` will cause a *strong* element to color its text green only if it is an immediate child of a *p* element, with no intervening tag.

- Adjacent Sibling Selector - this will select any element that follows a specific element in the marked up document. For example:

  `p + cite {color:yellow; }` will cause a *cite* element to color its text yellow only if it immediately follows a *p* element (at the same level). This means there is no other element between the *cite* and the *p* element.

- Class Selector - The class selector is used to assign properties to a named class. For example: In the stylesheet we could specify that "specially" marked paragraphs would have the following properties and values. Here the specially marked feature is indicated by the ".warning" suffix on the paragraph selector. So, any paragraph element marked as a "warning" paragraph will have its text colored red and made italic. (We have seen an example of this earlier). Similarly, I could specify that a head tag marked as warning would also need to be set as red italic text. The stylesheet code would look like this:

```
p.warning { color:red; font-style:italic; }
h1.warning { color:red; font-style:italic; }
or more compactly,
p.warning, h1.warning {color:red; font-style:italic; }
or really compact!
*.warning {color:red; font-style:italic; }
```

  Within the HTML document, we mark such paragraphs by inserting the attribute "class" with value = "warning" inside the paragraph start tag as shown below:

```
. . . <!-- previous HTML code -->
<p class="warning" > Do NOT drop this carton!
</p>
. . . <!-- additional HTML code -->
<h1 class="warning" > Safety Instructions </p>
. . . <!-- additional HTML code -->
```

- ID Selector - This type of selector is used to assign properties to an element with a specific and unique identifier. As contrasted with the "Class" selector above, that could apply to multiple elements, the ID selector can apply to only one. For example:
  In the stylesheet we could specify a uniquely marked paragraph that would have the following properties and values. Here the specially marked feature is indicated by the "#unique123" suffix on the paragraph selector. So, a paragraph in the HTML document marked as a "unique123" paragraph will have its text colored red and made italic. A paragraph marked as "unique456" will have its text color set to blue. The stylesheet code would look like this:

```
p#unique123 { color:red; font-style:italic; }
p#unique456 { color:blue; font-style:italic; }
```

  Within the HTML document, we mark such paragraphs by inserting the attribute "id" with value = "unique123" inside the paragraph start tag as shown below:

```
. . . <!-- previous HTML code -->
<p id="unique123" > Do NOT drop this carton! </p>
. . . <!-- additional HTML code -->
<p id="unique456" > Do NOT drop this carton! </p>
```

```
. . . <!-- additional HTML code -->
```

Inheritance

Inheritance is where a nested (child) element inherits the property values of its parent or ancestor, if these property values aren't declared in the child. For example, suppose we had the following entries on a style sheet:

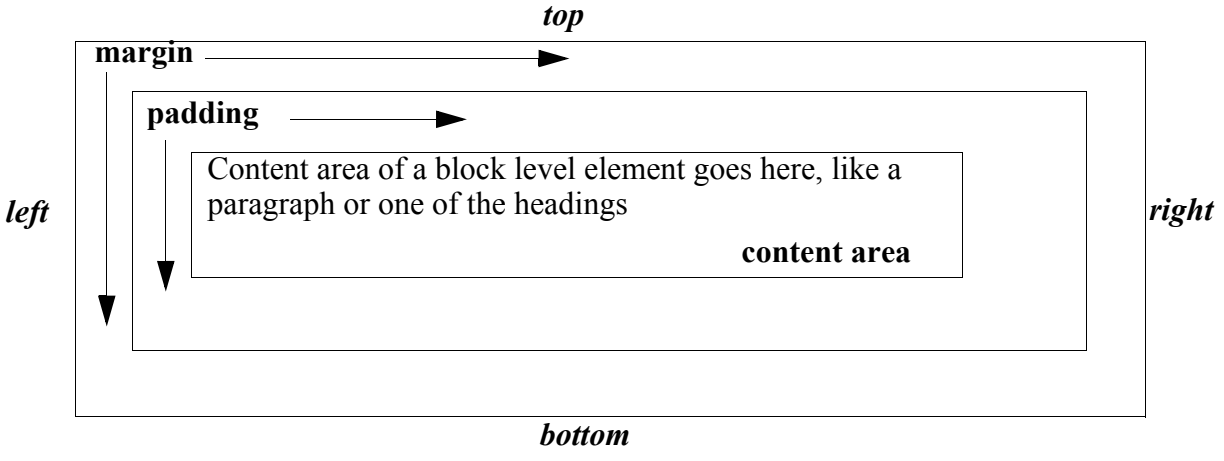
```
body {color:black;}
p {color:red;}
h1{ font-size:1.25em;}
```

In this case, the *body* element encloses all of the other elements, and so, unless overridden, all of the text in the document will be black. Since the *p* element overrides the color element, the outcome will be *red* paragraph text. Also, any tags nested within the *p* element will inherit this *red* text, unless it overrides the color property. The *h1* element, being nested within the *body* element, will inherit the black color.

Box Model

The CSS Box model describes how element content is laid out, primarily in the visual mode. The two basic layout patterns within CSS are the Box model and the Inline layout model. In effect, these are "layout managers" that may be familiar to the reader from the graphics components of languages like Java. CSS assumes that every *element* generates one or more boxes. These are rectangular and are called *element boxes*. This box has a content core that is surrounded by padding, borders, and margins, all of which can be null, that is, set to a width of zero.

Additional layouts are concerned with "positioned elements" and "floating elements". Consider the diagram below that shows some detail of the Box model. It is a feature of CSS that the background color of an element covers all of the area up to the outer edge of the border. This will fill the content area and the padding. The fundamental equation governing these parts of the "Box" is as follows: *margin-left+ border-left-width + padding-left+width + padding-right + border-right-width + margin-right += value of width of parent's content area*



In-Line Model

Setting inline text is slightly more involved and the following quotes are from the CSS2 specification that may be found at the W3C.

> In an inline formatting context, boxes are laid out horizontally, one after the other, beginning at the top of a containing block. Horizontal margins, borders, and padding are respected between these boxes. The boxes may be aligned vertically in different ways: their bottoms or tops may be aligned, or the baselines of text within them may be aligned. The rectangular area that contains the boxes that form a line is called a line box.
>
> The width of a line box is determined by a containing block. The height of a line box is determined by the rules given in the section on line height calculations. A line box is always tall enough for all of the boxes it contains. However, it may be taller than the tallest box it contains (if, for example, boxes are aligned so that baselines line up). When the height of a box B is less than the height of the line box containing it, the vertical alignment of B within the line box is determined by the 'vertical-align' property.
>
> When several inline boxes cannot fit horizontally within a single line box, they are distributed among two or more vertically-stacked line boxes. Thus, a paragraph is a vertical stack of line boxes. Line boxes are stacked with no vertical separation and they never overlap.

In general, the left edge of a line box touches the left edge of its containing block and the right edge touches the right edge of its containing block. However, floating boxes may come between the containing block edge and the line box edge. Thus, although line boxes in the same inline formatting context generally have the same width (that of the containing block), they may vary in width if available horizontal space is reduced due to floats. Line boxes in the same inline formatting context generally vary in height (e.g., one line might contain a tall image while the others contain only text).

When the total width of the inline boxes on a line is less than the width of the line box containing them, their horizontal distribution within the line box is determined by the 'text-align' property. If that property has the value 'justify', the user agent may stretch the inline boxes as well.

Since an inline box may not exceed the width of a line box, long inline boxes are split into several boxes and these boxes distributed across several line boxes. When an inline box is split, margins, borders, and padding have no visual effect where the split occurs. Formatting of margins, borders, and padding may not be fully defined if the split occurs within a bidirectional embedding.

Inline boxes may also be split into several boxes within the same line box due to bidirectional text processing.

Property Values

Once a property is specified, such as a *color* property, then that requires a *value* to be associated with it. On the other hand, a property such as a margin on the left side of the text, that is *margin-left*, it can take a numeric such as 0.5in. Below you will see several sections of types of values that properties can take. It will be convenient to specify several keyword that will represent these categories of values. *The keywords used in this course will be:*

- <color keyword >
- <length >

- <percentages >
- <url>
- <hexadecimal>

Value Representations

To indicate grouping and options within values the following conventions will be observed:

- A vertical bar (|), is used to separate two or more alternatives. Only one may be used.
- A sequence of words means that they all must occur in the order shown.
    1. Square brackets ([ ]), are used to group values.
- Double vertical bars (||) indicate alternates, but you can select any one (or more) in any order.

To indicate modification of keyword, value types, or groups, the following symbols will be used:

- An asterisk (*) indicates that the preceding entity (that is keyword, value type, or group), can occur zero or an unlimited number of times.
- A question mark (?) indicates that the preceding entity can occur zero or once, that is, the entity is optional.
    1. A plus sign(+) indicates the preceding entity must occur at least once, with an unlimited upper bound.
- A pair of integer within curly braces sets minimum and maximum occurrences for the preceding entity. For example `margin{0,4}` indicates that the word `margin` can occur from zero up to a maximum of four times.

Value Framework

Color Values

Color values can be used to specify a foreground color, that is, text, or can be used to set the background color. In addition, borders can have colors specified. A keyword for the color values described below is *color*. The particular types of values that may be used to specify colors follow:

- Hexadecimal values - #RRGGBB where RR corresponds to how much "red" is in the color, GG corresponds to the amount of "green" while BB denotes the "blue" quantity. Each of these values can range from zero to 255 that is, #00 to #FF in hexadecimal. Pure red would be represented as - #FF0000, pure green by - #00FF00 and pure blue would be written as - #0000FF. Gray would be #808080.
- Hexadecimal shorthand. This setting uses only three value, with each value considered duplicated. For example red could be written as - #F00, which is shorthand for #FF0000. Gray would be - #808080
- rgb(r%, g%, b%) - Using percentages from 0- 100, then red would be written as - rgb(100%, 0%, 0%), while blue would look like - rgb (0%, 0%, 100%). Gray would be rgb(50%, 50%, 50%);

- rgb(r, g, b) - using integer values between 0- 255 (these correspond to the hexadecimal values discussed earlier). Now red would be - rgb(255, 0 0), blue would look like- rgb(0,0,255), whereas white would be - rgb(255, 255, 255). Gray is - rgb (128,128,128). Higher values are clipped to 255.
- <keyword> - CSS has 16 colors defined by keywords: aqua, black, blue, fuchsia, gray, green lime, maroon, navy, olive, purple, red, silver, teal, white, yellow.

Length Values

Absolute and relative lengths are supported, allowing a leading - or + sign. The numeric value is followed by a unit identifier. These values are referred to by the keyword *length*.

- em (em-height) - *this is the preferred measure on the web. It is a measure that depends on the current font's character box height. Traditionally, this was equivalent to the width of the capital letter "M" in the current font. It has evolved from that however, and is used as a more general length measure, both horizontal and vertical.
- px (pixel) - Computer displays are made up of pixels (picture elements). These are the small dots that make up the total image you see on the screen. Since each monitor has (usually) a different number of pixels per inch, this is also a problematic measure to use for accurate work.
- in (inch) - not recommended for use on the web

  cm (centimeter) - not recommended for use on the web

  mm (millimeter) - note recommended for use on the web
- pt (point) - this is the traditional typographers unit of measure. There are 72 points per inch. Again, these do not map consistently to any web environment and are therefore discouraged for web work. For printed media they are still useful however.
- pc (pica) - this is again a typographers measure with 1 pica defined as 12 points. The same warnings apply as above for points. There are 72 points per inch and so 6 picas are an inch.

Angle Values

- These are used for *aural* properties, but there are no aural styles currently supported in the major browsers. The units here can be in radians or degrees.

Time Values

- These are used for *aural* properties, but there are no aural styles currently supported in the major browsers. The units here can be seconds or milliseconds.

Frequency Values

- These are used for *aural* properties, but there are no aural styles currently supported in the major browsers. The units here can be Hertz, (that is cycles per second) or kiloHertz.

String Values

- This allows an arbitrary sequence of (potentially Unicode) characters to be codified as a quote delimited string. The quote can be either a single or a double quote.

Percentage Values

- These are integers or real numbers optionally preceded by a sign (+ , -). These values are referred to as the keyword *percentage*.
- URI values (Universal Resource Indicator) are used for unique identifications, or more usually to refer to a more specific type of URI called a URL (Universal Resource Locator). A common use is to have the URI value point to the location of a specific file, often a graphic file. For example, to set a background image we could write:

```
body {background-image: url(http://www.company.com/images/jpg1.jpg) }.
```

This example will attempt to fetch a "jpg" image from a location on the web and use that image as background for the `body` element. These values will be referred to using the keyword *uri*.

Pseudo-Classes

The `:link, :visited, and :active` are called *pseudo classes* and are used to assign properties to link states such as unvisited, visited, active and hover. These are all associated with the *a* anchor element. For example:

1. a:link {color:purple;}
2. a:visited {color:gray;}
3. a:active {color:red;}
4. a:hover {color:fuchsia;}

Pseudo-Elements

Pseudo-Elements lets a user agent virtually insert markup into a document, and then apply styles to that virtual markup. This allows the user agent to style things like "the first line of an element". Since this virtual markup is treated as an element-like structure, the selector is called a "Pseudo-element". These elements are as follows:

1. `first:letter` - this selector is used to apply styles to the first letter of an element. For example: `E:first-letter {color:red;}` would apply a style to the first letter of element "E".
2. `first:line` - this selector is used to apply styles to the first line of an element. For example: `E:first-line {color:red;}` would apply the style to the first displayed line of element "E".
3. `:before` - this selector is used to put generated content into the document before the content of an element. For example:
   `E:before {content:"Look Ahead!"; color:blue;}` would place content before the content of the element "E".
4. `:after` - this selector is used put generated content after the content of an element. For example:
   `E:after {content:"Look Behind You!"; color:blue;}` would place content after the content of the element "E".

@ Rules

The *at-rule* set of constructs simply distinguish a block of instructions.

@import

This rule is used to import a stylesheet into another stylesheet. For example:

```
<style type="text/css" > @import url(extra.css) </style >
```

@media

This rule is used to specify the target client media for a set of style rules. This allows the developer to place multiple sets of style rules within a single style sheet. The media available are: print, aural, braille, handheld, projection, screen, embossed, tty, tv, or all. For example:

```
@media print {body {margin-left:5cm;} }
@media screen {body {margin-left:2em;} }
@media print,screen {body {color:gray;} }
```

@page

This rule is used to define the page context for printed media. For example:

```
@page {size: 8.5in 11in; margin: 1in; }
```

CSS - {HTML, XHTML} Interactions

The linkage between the HTML or XHTML documents and CSS is usually effected by the *link* element that is inserted within the *head* section of an (X)HTML document. The structure of this *link* is shown in the following example:

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

CSS Reference - CSS Mobile Profile 1.0

This reference set of CSS properties specifies a profile of the CSS level 2 spec, appropriate for mobile devices such as wireless phones or low end PDAs. Note that these styling properties described here can be used by HTML, XHTML, and XML documents alike.

The source of the properties found in the table below come from the CSS Mobile spec.This is a W3C specification and its authors place this spec in context as follows:

> The CSS Mobile Profile specifies a conformance profile for mobile devices, identifying a minimum set of properties, values, selectors, and cascading rules. The resulting CSS Mobile Profile is very similar to CSS1.

We will extract from this spec both its selector table and its property table, as shown below.

Legend for the Table

The following symbols and their interpretations are as follows:

Symbol	Interpretation
|	alternation (or) , one of the values must occur
[[. . .]]	choose any one, in any order. Can choose multiple components

[. . .]	group elements
{ n,m }	apply to the preceding at least "n" times and no more that "m" times.

CSS Reference Table (Selectors and Properties)

Pattern	Meaning	Selector Type
*	Matches any element	Universal selector
E	Matches any E element	Type selectors
E, F	Matches E or F elements	Type selectors
E F	Matches any F element that is a descendant of an E element	Descendant selectors
E:link E:visited	Matches element E if E is the source anchor of a hyperlink of which the target is not yet visited(link) or already visited (visited)	The link pseudo-classes
E:active	Matches E element during certain user actions	The dynamic pseudo-classes
E:focus	Matches E during certain user actions	The dynamic pseudo-classes
div.warning	Same as div[class = "warning"]	Class selectors
E#myid	Matches any E element ID equal to "myid"	ID selectors

CSS Property Table

Property Name	CSS Values	Initial value(s)
'background' note: this is a shorthand notation for the alternatives listed	['background-color' || 'background-image' ||'background-repeat'||background-attachment' || 'background-position'] | inherit	body{background: blue url(snow.gif) top center scroll
'background-attachment' tiling context and scroll state of a background element	scroll | fixed | inherit	h2 { background-attachment: scroll}

'background-color' set color of background, including padding	<color> | transparent |inherit	p.danger {background-color: red;}
'background-image' this is a reference to an image to be placed in background of an element	<uri> |none | inherit	table { background-image: url(snow.png) }
'background-position' origin of a repeated image	[[<percentage> | <length>]{1,2} | [top | center | bottom]| [left | center | right]] |inherit	0% 0% body {background-position: top left; }
'background-repeat' defines directions of repeated background image	repeat|repeat-x|repeat-y|no-repeat | inherit	repeat
'border' note: this is a shorthand notation for the alternatives listed	['border-width' || 'border-style' || color]|inherit	`h3 { border: 0.15em inset blue; background-color:transparent;}` See individual properties
'border-color' set the color of 1 to four border segments , that is, top right bottom or left border	<color>{1,4} |transparent|inherit	See individual properties
'border-style' set the style of 1 to four border segments , that is, top right bottom or left border	<border-style> {1,4}|inherit	See individual properties
'border-top' 'border-right' 'border-bottom' 'border-left'	['border-top-width'||'border-style'|| <color>]|inherit	See individual properties

'border-top-color' 'border-right-color' 'border-bottom-color' 'border-left-color' set the colors of border segments	<color> |inherit	The value of the 'color' property
'border-top-style' 'border-right-style' 'border-bottom-style' 'border-left-style' set the styles of border segments	<border-style> |inherit	none
'border-top-width' 'border-right-width' 'border-bottom-width' 'border-left-width' set the widths of border segments	<border-width> |inherit	medium
'border-width' short hand for setting border segment widths	<border-width>{1,4}|inherit	See individual properties
'clear' keep an element from being displayed next to floated elements	none|left|right|both|inherit	none

'color' set the foreground color of an element (usually this is the text color)	<color>|inherit	user agent dependent
		`h1 { color:#f00; background:transparent;}`
		`h3{color:green; background:transparent;}`
		`p { color:#ff3300; background:transparent;}`
		`em {color: rgb(255, 0,0); background:transparent; }`

'display' set the basic category of an element thus determining its display properties	inline|block|list-item|none **other values for CSS1, CSS2	inline `p { display:block; padding: 0.5em; border:0.1em;}`
'float' make an element move to one side of the parent element's content area. This lets content flow around it.	left|right|none|inherit	none
'font' this determines the text presentation set of characters	[['font-style'||'font-variant'||'font-weight']? 'font-size'[line-height']? 'font-family']|caption|icon|menu|message-box|small-caption|status-bar|inherit	`p { font: bold 1.15em/120% Arial,sans-serif;}` See individual properties
'font-family' this is the family of fonts to be selected from	[[<family-name> | <generic-family>]* [<family-name>|<generic-family>]|inherit	user agent dependent `p { font-family: Arial, sans-serif;}`
'font-size' this is the size of the characters in the chosen font	<absolute-size>|<relative-size> <length>|<percentage>|inherit	medium
'font-style' this is a variation within the font *	normal|oblique|italic|inherit	normal `p { font-style:italic; font-family: Arial, sans-serif;}`
'font-variant' this is a variation within the font	normal|small-caps|inherit	normal `p { font-variant:small-caps; font-size:1.3em; font-style:italic; font-family: Arial, sans-serif;}`

'font-weight' this sets the type density or 'blackness'	normal|bold|bolder|lighter 100|200|300|400|500|600|700|800|900|inherit	none `h3 {font-family: Helvetica, sans-serif; font-weight:800;}`
'height' this sets the height of an element's content area	<length>| <percentage> |auto|inherit	auto img.view1 { height: 20em; }
'list-style' set the type of list style desired	['list-style-type'||'list-style-position'||'list-style-image']|inherit	See individual properties
'list-style-image' this refers to an image to be used as the marker for a list	uri | none | inherit	none
'list-style-position' this places the marker relative to the list content	inside|outside|inherit	outside
'list-style-type' this sets the ordinal or numeric style of the marker	disc|circle|square|decimal|lower-roman|upper-roman| lower-alpha|upper-alpha|none|inherit	disc
'margin' this sets the spacing around an element (see also padding and border width)	<margin-width>{1,4}|inherit	See individual properties `body {margin: 2em 1em 3em 4em; }`
'margin-top' 'margin-right' 'margin-bottom' 'margin-left' this sets the margin for that segment of the element that is, margin above, right, bottom, left	<margin-width>|inherit	none

'padding' this sets immediate space around the element	<padding-width>{1,4}|inherit	none
'padding-top' 'padding-right' 'padding-bottom' 'padding-left' this sets the padding for that segment of the element that is, padding above, right, bottom, left	<padding-width>|inherit	none

'text-align'	left|right|center|justify|<string>|inherit	user agent dependant
		`h3 {text-align: center;}`
		`p { text-align: justify; }`

'text-decoration'	none| [underline||overline||line-through||blink]|inherit	none
		`h3 {text-decoration: underline; color:blue; background:transparent;}`
		`a { text-decoration: underline; color:purple; }`

'text-indent' indent the first line of text (of a block element)	<length>| <percentage>|inherit	0
'text-transform' capitalization instructions	capitalize|uppercase|lowercase|none|inherit	none
'vertical-align' set the vertical alignment of text within a line or table cell	baseline|sub|super|inherit	baseline

'white-space' this determines how browser handles white space in the source document	normal|pre|nowrap|inherit	normal
'width' this sets the width of an element's content area	<length>| <percentage>|auto|inherit	none `h3 {width: 15em; height:7em; border: 0.5em inset green; padding:2em 1em 2em; }`

Additional Features of the CSS Language

Linking and Styling Approaches

Here are the ways you can invoke the properties specified in Table I, shown later in this document.

The Style Element

This STYLE element is contained within the HEAD element and is used to embed styles within the HTML page itself. For example (this is nested within the HEAD element) :

```
< style type= "text/css" >
h1 { color:red; margin-top:2em; }
</style >
```

Link Element

The LINK element is nested within the HEAD element and is used to reference an external style sheet to be used with this web page. For example, here is a call out to use the style sheet denoted by the *href* attribute.

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

&import At-Rule

This rule is nested within the STYLE element and must appear before any other style rules. It is used to import a style sheet into another style sheet. Note that the styles in the imported sheet will be overridden by style appearing within the rest of the STYLE element. So, for the example below, if the "anothersheet.css" had an style such as h1 {color:green; } it would be replaced by the red color specified in the body of the STYLE element.

```
< style type= "text/css" >
&import url("anothersheet.css");
h1 { color:red; margin-top:2em; }
</style >
```