Structure and Style for the Web

High Tech Center Training Unit

of the California Community Colleges at the
Foothill-De Anza Community College District

21050 McClellan Road
Cupertino, CA 95014
(408) 996-4636

www.htctu.net

Table of Contents

Creating Structure for Web Pages

Overview

When creating a Web page, it may be necessary to impose "structure" to the content. In other words, it may be necessary to distinguish between certain text elements such as headings or body text or lists of items. Adding structure to a Web page provides a method to control the logical order of information by as well as identify the "type" of content. For example, is the title at the top of the paragraph a heading or is a group of items actually an organized "list" of content. Adding structure to a page can aid in identifying the "meaning" of a piece of content as well as aid in supporting assistive computer technologies.

The benefit to assistive computer technologies (particularly screen-readers), is that a user can navigate between various pieces of information if structure is used on a Web page. Visually, a user can scan a page looking for content that "stands-out" and is relevant to their specific needs. Content may be formatted with bold effects or italics or font sizing to distinguish it from the rest of the information. However, to an individual who may not be able to "see" the page content, the changes to the presentation of the content are not usable. With the introduction of some structure, not only can a person "see" the visual change, but a person using assistive computer technology can identify the various content changes as well. What can be accomplished visually (i.e., changing the look of content) can also be accomplished using structure.

In addition to supporting assistive computer technology, the use of structure makes it easier for content authors to visually format how the information will look on the Web page. The structural elements used on the Web page provide the "hooks" for applying the various presentational elements via cascading style sheets (CSS). For some additional reading on the use of structure on Web pages, please refer to the WebAIM site (http://www.webaim.org/techniques/structure/) or the HTML Techniques for WCAG 1.0 (http://www.w3.org/TR/WCAG10-HTML-TECHS/).

Headings

Similar to the headings in this document, headings provide a method to separate content and organize information into discreet "chunks", providing a content organization and hierarchy of information. Within a Web page, headings also provide benefits for individuals using assistive computer technology such that users can navigate through the page using the various headings as "bookmarks" to jump from content to content.

Headings are designated by the tag <h1>, <h2>, <h3>, etc. There are a total of six headings permitted for a Web page. When using headings, <h1> designates the most important information of the page (a page title perhaps) with the subsequent headings representing content of lower importance. As a rule, you should not use a higher numbered heading before a lowered number heading, unless that lowered numbered heading has already been in use on the page.

Heading Example: (not allowed)
H2
        H1

Heading Example: (allowed)
H1
  H2
        H3
  H2

Lists

Similar to adding headings or identifying body text, another method for adding structure to a Web page is to specify content that is to be a list. Lists can be either bulleted (called "unordered") or numbered (called "ordered"). An "ordered" list may contain a sequence of either letters or numbers. An "unordered list" is designated by the tag **<ul>** and an "ordered list" is designated by the tag **<ol>.**

Lists can be valuable when there is a need to present information that could be represented in groups. For instance, a group of hyperlinks or course objectives could be represented using a list. The "rollover menus" used on websites could also be reformatted as lists, with the "revealed content" as a sub-list of the main content. (A bit of CSS and/or JavaScript can also be used to visually present the content).

Example Unordered list Code:

```
<ul>
<li>List Item One</li>
<li>List Item Two</li>
<li>List Item Three</li>
</ul>
```

Lists may also exist as a set of definitions (known as a definition list). A definition style list identifies the definition term followed by the actual definition below the term itself. For example, a list of HTML terms followed by a definition of each could be organized with a definition list.

Example Definition list Code:

```
<dl>
<dt>HTML</dt>
<dd>Stands for Hypertext Markup Language, a language for creating Web pages</dd>
<dt>XML</dt>
<dd>Stands for eXtensible Markup Language, also called a meta-language</dd>
</dl>
```

Data Tables

Data tables are exactly what the name implies - a table that contains data about some information sequenced in a specific format. For example, if you have a list of days, a list of appointments on specific days, and a list of specific times for those appointments on specific days, it would make sense to develop a table to display that information. Here is a practical example below:

	Sun.	Mon.	Tues.	Wed.	Thurs.	Fri.	Sat.
9:00 AM	Sleep		Work	Work		Work	Sleep
12:00 PM	Nap	Lunch			Lunch		Nap
2:00 PM	Hamster Feeding			Cat Pruning			Breakfast
5:00 PM				Hospital			Nap
6:00 PM		Football					

The Scope Method

The "scope" method is one method that can be used to improve the accessibility of a data table. The scope method can also be used to meet paragraph G of the US Section 508 Standards for Web-based Internet and Intranet Information and Applications (1194.22). Essentially, the scope method identifies the row and/or column header information for an entire row and/or column. The scope method does not require the Web page author to directly associate each table cell with its appropriate row and/or column header. In other words, with the scope method, the data within the entire row or column is automatically associated with the identified header information.

Scope Code

(from Section 508 Standards, www.access-board.gov)

The first row of each table should include column headings. Typically, these column headings are inserted in <TH> tags. These tags at the top of each column should include the following attribute:

scope="col"

By doing this simple step, the text in that cell becomes associated with every cell in that column. The first column of every table should include information identifying information about each row in the table [row headers]. Each of the cells in that first column are created by either <TH> or <TD> tags. Include the following attribute in these cells:

scope="row"

While this technique dramatically improves the usability of a web page, using the scope attribute does not appear to interfere in any way with browsers that do not support the attribute.

Scope Example:

```
<table>
<tr>
<th> </th>
<th scope="col" >Spring</th> <th scope="col" >Summer</th> <th
scope="col" >Autumn</th> <th scope="col" >Winter</th> </tr>
<tr> <td scope="row" >Betty</td> <td>9-5</td> <td>10-6</td> <td>8-
4</td><td>7-3</td>
</tr>
<tr> <td scope="row" >Wilma</td> <td>10-6</td> <td>10-6</td> <td>9-
5</td> <td>9-5</td>
</tr>
<tr> <td scope="row" >Fred</td> <td>10-6</td> <td>10-6</td> <td>10-
6</td> <td>10-6</td>
</tr>
</table>
```

This table would be displayed as follows:

	Spring	Summer	Autumn	Winter
Betty	9-5	10-6	8-4	7-3
Wilma	10-6	10-6	9-5	9-5
Fred	10-6	10-6	10-6	10-6

Other Methods

Data tables that consist of more than one column or row headings may require an alternate solution for meeting the accessibility requirements. Another method to address data table accessibility is to use the "id and headers" method. A full-day training and manual on creating accessible forms and tables is available at the HTCTU website

http://www.htctu.net/trainings/manuals/tutmain.htm. Please refer to that manual for specific instructions on how to use other methods to code data tables using the "id and headers" approach.

For some different data table designs, visit:
http://icant.co.uk/csstablegallery/
To incorporate the data table designs using the above templates, select the "Download" hyperlink for the theme you are interested in applying to your content.

Inline Elements

There are a variety of inline elements that can be used to improve the structure of content on a Web page. Some inline elements are used routinely when creating website (e.g., the <a> tag for hyperlinks) whereas others are not used as often. The following are a few inline elements that may be useful when creating more structured Web content in higher education:

**, **
The strong and em tags cause the text to be represented as bold or italics, respectively. Instead of formatting text to be bold, use the , tag to format text to look bold. Similarly, for italics, use the , tags to visually identify italic text.

<abbr>
The abbreviation tag (<abbr>) is used to identify if some is being abbreviated. For example, HTML is the abbreviation of HyperText Markup Language and can be identified correctly using structure. The abbreviation code would be:

```
<abbr title="HyperText Markup Language">HTML</abbr>
```

<acronym>
Similar to the abbreviation tag, the acronym tag identifies an acronym within the structure of the Web page. The advantage to assistive computer technology is that the user can be told the acronym title as opposed to just hearing jumbled letters. The acronym code would be:

```
<acronym title="In My Humble Opinion">IMHO</acronym>
```

<code>
The code tag allows a content author to designate a specified amount of content as computer code as opposed to text content in the remainder of the Web page. Generally, Web browsers will render <code> content as a monospaced, serif font in order to set it apart visually and structurally from the rest of the page content. This is useful for those working in computer applications to understand that the <code> content is different from surrounding text.

Forms

Forms on Web pages have several HTML tags and attributes that are necessary for accessibility. Specifically, the use of the <label> tag is important for linking the form input field with the appropriate Web page text. Using the <label> tag with an "id" attribute can have benefits to customizing the presentational element of a Web page form as specific styles can be linked to specific "id" attributes. This allows a Web page designer to include the required accessibility content and provide a "hook" into the form to customize the visual presentation of the Web-based form.

Example Form Code:

```
<form name="feedback" method="post" action="getinfo.php"
id="myform">
 <div>
  <fieldset>
  <legend>Contact Information:</legend>
    <label for="firstname">First Name:</label>
    <input type="text" id="firstname" name="fname" size="20" />
    <label for="lastname">Last Name:</label>
    <input type="text" id="lastname" name="lname" size="20" />
  </fieldset>
<input type="submit" value="Submit Information" name="submit" />
   </div>
</form>
```

A full-day training and manual on creating accessible forms and tables is available at the HTCTU website http://www.htctu.net/trainings/manuals/tutmain.htm. Please refer to that manual for specific instructions on how to make forms accessible to assistive computer technologies.

Cascading Style Sheets

Overview

Cascading Style Sheets (CSS) are a method of Web design that visually presents Web page content according to a presentation style that is separate from the page code. There are several advantages to using CSS to format the presentation elements of a Web page. Using CSS, a Web page author can separate document content from the manner in which it is presented, (ideally) allowing for more fluid transitions between various platforms. CSS provides control for spacing, alignment, and positioning of content without relying on the need for layout tables or frames. Font style, color, and font size can all be manipulated using CSS as well. While not all Web browsers render a CSS-based website in the same way, more recent Web browsers have better support for the various style sheet properties.

Accessibility Benefits

CSS offers the potential for users to customize their web browser such that the content displayed in their preferred Internet browser will present the information in the manner the user chooses. For a low-vision individual, this provides the opportunity to enlarge web content displayed in the internet browser at a resolution appropriate to the individual. Using CSS provides the user with the option of manipulating the way in which the Web content is presented in their specific Internet browser.

Style Sheets, Part 1

A style sheet is simply a document that identifies how a specific selector should be presented visually (or aurally) in the Web browser. Style sheets can be either Inline, Internal, External or Imported. Imported style sheets are useful when you wish to use CSS designs that may not be supported in older Web browsers.

Inline

Inline style sheets apply the formatting elements locally – that is, at the level of the text within the web page document. An inline style sheet may contain the following:

```
<h1 style="font-family:sans-serif; font-size:20pt; color:#996633">
```

Internal

Internal style sheets reside within the **<head>** section of an HTML document and affect the presentation of content throughout the document. An internal style sheet may contain the following:

```
<head>
<style>
<!—
body {font-family:Helvetica, sans-serif; font-size:extra-large;
color:orange}
-->
</ style>
</ head>
```

External

External style sheets exist outside of the document structure and are referenced when the document (or web page) is loaded into the browser. External style sheets have the advantage of being easy to change in order to affect a large number of web pages simultaneously. An external style sheet may contain the following:

```
h2 {font-family: Helvetica, sans-serif;
      font-size: extra-large;
      color: wheat
}

p {font-family: serif;
      font-size: 14;
      color: black;
}
```

This specifies all level 2 headings will be Helvetica font (or another sans-serif font), extra-large size, and a wheat color. All paragraph content will be a serif font (the default on the computer), 14 point, and a black color.

Imported

Imported style sheets are similar to **External** style sheets in that the attributed styles are *imported* into the document within the `<Style>` tag. Older Web browsers do not recognize the `@import` element and will therefore ignore the associated style sheet. An imported style sheet could include the following:

```
STYLE TYPE="text/css" MEDIA="screen, projection">
<!--
  @import url(http://www.htmlhelp.com/style.css);
  @import url(/stylesheets/punk.css);
  DT { background: yellow; color: black }
-->
</STYLE>
```

Style Sheet Formatting

The syntax for a linked style sheet is contained in a text document with an extension of **.css**. The **.css** files remain on the web server and are linked to their respective web page documents. The syntax for style sheets is the following:

```
selector  {property1: value1;
      property2: value1, value2;
      property3: value1;
}

selector  {property1: value1;
      property2: value1;
}

selector  {property1: value1;
      property2: value1;
      property3: value1;
}
```

Selector

These elements are those for which you are setting a style. For instance, the selector is identified by a HTML tag, a specified class, or id element used on the page.

Headings:	h1, h2, h3, h4, h5, h6
Body Text:	p, blockquote
Hyperlinks:	a
Lists:	ul, li, ol

Properties

Properties represent the category of the desired presentation appearance. Properties include descriptions for the font-family, font-style, color, etc. These properties are then followed by the appropriate value for this category. A list of the different properties and the associated values is in the next section.

Values

Values are what specify how the property is to be displayed within the style sheet. Each property has several values from which to choose in order to present the content according to the web designers intent. A list of the different values with their associated properties is available in the next section.

Font Properties

The font properties identify the specific font to be used on a Web page as well as how the specific font should look based on its style, size, color and weight.

Property:
font-family

Values:
serif (e.g., Times)
sans-serif (e.g., Arial)
cursive (e.g., *Zapf-Chancery*)
fantasy
monospace (e.g., `Courier`)
or specific font name

Example code:
```
p {font-family: "New Century Schoolbook", Times, serif }
```

Property:
font-style

Values:
normal
italic
oblique

Example code:
```
h1 {font-style: oblique }
p  {font-style: normal }
```

Property:
font-weight

Values:
normal
bold
bolder
lighter
100-500 (lighter)
600-900 (darker Example code:

Example code:
```
h1 {font-weight: 800 }
```

Property:
font-size

Values:
absolute-size (xx-small, x-small, small, medium, large, x-large, xx-large)
relative-size (larger, smaller)
length (pt)
percentage (%)

Example code:
```
h1 { font-size: large }
p  { font-size: 12pt }
li { font-size: 90% }
```

Color and Background Properties

The color property and background property allows a developer to define values using a color chart or a verbal representation of a particular color. Additionally, using the background property, a style sheet author can visually manipulate both colors as well as images to control the location and presentation of images on the Web page. If using the background property with images, it is important to note that you cannot provide an alternative text description for the image and must communicate that information in another manner.

Property:
color

Values:
aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow, wheat. Can also use color values.

Example code:
```
h1 { color: blue }
h2 { color: #000080 }
```

Property:
background-color

Values:
color-value (RGB, Hex, etc.)
transparent

Example code:
```
body { background-color: white }
h1   { background-color: #000080 }
```

Property:
background-image

Values:
<url>
none

Example code:
```
body { background-image: url(/images/foo.gif) }
p  { background-image: url(http://www.htmlhelp.com/bg.png) }
```

Property:
background-repeat

Values:
repeat
repeat-x
repeat-y
no-repeat

Example code:
```
p.candybar { background-image: url(candybar.gif); background-repeat:
repeat-x }
```

The background-repeat property determines how a specified background image is repeated. The repeat-x value will repeat the image horizontally while the repeat-y value will repeat the image vertically

List Properties

Using some CSS, lists can take on a variety of structure for Web pages. Lists may show up as navigation aids, a set of definitions for a page, or a group of form fields. Below are some properties for manipulating how a list is visually presented in the Web browser.

Property:
list-style-type

Values:
none
disc ○
circle ●
square ■
decimal 1 2 3 etc.
lower-roman i ii iii etc.
upper-roman I II III etc.
lower-alpha a b c etc.
upper-alpha A B C etc.

Example code:

```
li.square { list-style-type: square }
ul.plain  { list-style-type: none }
ol        { list-style-type: upper-alpha } /* A B C */
```

The list-style-type property specifies the type of list-item marker, and is used if list-style-image is none or if image loading is turned off.

Property:
list-style-image

Values:
none
<url of image>

Examples:

```
ul.check { list-style-image: url(/LI- markers/checkmark.gif) }
ul li.x  { list-style-image: url(x.png) }
```

The list-style-image property specifies the image that will be used as list-item marker when image loading is turned on, replacing the marker specified in the list-style-type property.

Property:
list-style-position

Values:
inside
outside

Examples:

```
li.square { list-style-position: inside; list-style-type: square}
```

The list-style-position property takes the value inside or outside, with outside being the default. This property determines where the marker is placed in regard to the list item. If the value inside is used, the lines will wrap under the marker instead of being indented.

Some CSS authors also create styles using the just the `list-style` property. The `list-style` property is a shorthand method instead of writing out the properties `list-style-type`, `list-style-position`, and `list-style-image`.

Examples:

```
li.square { list-style: square inside }
ul.plain  { list-style: none }
ul.check  { list-style: url(/LI-markers/checkmark.gif) circle }
ol        { list-style: upper-alpha }
ol ol     { list-style: lower-roman inside }
```

Length and Percentage Units

Length units

The format of a length value is an optional sign character ('+' or '-', with '+' being the default) immediately followed by a number (with or without a decimal point) immediately followed by a unit identifier (a two-letter abbreviation). After a '0' number, the unit identifier is optional.

Some properties allow negative length units, but this may complicate the formatting model and there may be implementation-specific limits. If a negative length value cannot be supported, it should be clipped to the nearest value that can be supported. There are two types of length units: relative and absolute. Relative units specify a length relative to another length property. Style sheets that use relative units will more easily scale from one medium to another (e.g. from a computer display to a laser printer).

These relative units are supported:

```
h1 { margin: 0.5em }    /* ems, the height of the element's font */
h1 { margin: 1ex }     /* x-height, ~ the height of the letter 'x' */
p  { font-size: 12px }  /* pixels, relative to canvas */
```

The relative units 'em' and 'ex' are relative to the font size of the element itself. The only exception to this rule in CSS1 is the 'font-size' property where 'em' and 'ex' values refer to the font size of the parent element.

Pixel units, as used in the last rule, are relative to the resolution of the canvas, i.e. most often a computer display. If the pixel density of the output device is very different from that of a typical computer display, the UA should rescale pixel values. The suggested *reference pixel* is the visual angle of one pixel on a device with a pixel density of 90dpi and a distance from the reader of an arm's length. For a nominal arm's length of 28 inches, the visual angle is about 0.0227 degrees.

Child elements inherit the computed value, not the relative value:

```
body { font-size: 12pt;  text-indent: 3em;  /* i.e. 36pt */;}

h1 { font-size: 15pt }
```

In the example above, the 'text-indent' value of "h1" elements will be 36pt, not 45pt.

Absolute length units are only useful when the physical properties of the output medium are known. These absolute units are supported:

```
h1 { margin: 0.5in }    /* inches, 1in = 2.54cm */
h2 { line-height: 3cm }   /* centimeters */
```

```
h3 { word-spacing: 4mm }  /* millimeters */
h4 { font-size: 12pt }    /* points, 1pt = 1/72 in */
h4 { font-size: 1pc }     /* picas, 1pc = 12pt */
```

Percentage units

The format of a percentage value is an optional sign character ('+' or '-', with '+' being the default) immediately followed by a number (with or without a decimal point) immediately followed by "%".

Percentage values are always relative to another value, for example a length unit. Each property that allows percentage units also defines what value the percentage value refer to. Most often this is the font size of the element itself:

```
p { line-height: 120% }  /* 120% of the element's 'font-size' */
```

Classes of Style

An advantage of using CSS for web pages is the ability to globally affect the presentation of specific throughout a website. However, there is always the situation in which certain elements require different formatting. This can be accomplished by using the class element in order to specify the final presentation of select elements.

The following code would apply a large, sans-serif, italicized font to all first-level headings.

```
H1   {font-family: Arial, sans-serif;
       font-style: italic;
       font-size: large;
}
```

However, if you wanted to have some of the first-level heading to appear in a blue, serif font, it would be necessary to create a class the specifies the desired appearance.

```
.specialh1 {font-family: Times, serif;
             color: blue; }
```

In the HTML code, it would be necessary to call in this special class in order to display this specific presentation style.

```
<h1>This is an example of the regular Heading 1 style</h1>

<h1 class="specialh1">This is an example of the special Heading 1
style</h1>
```

Using the `<div>` tag, a designer can also apply specific styles to whole blocks of content on a web page. Content within the `<div>` tag would change the appearance to that specified by the class element.

Example:

```
<div class=”specialh1”>
      <p>Paragraph content would go here…..</p>
      <table>
            Table data would be formatted as well….
      </table>
</div>
```

Inheritance

Another advantage to using CSS in web page design is the ability of different styles to *inherit* appearance values from previous levels. For instance, in the previous example discussing the Class property, the Heading 1 style was a large, sans-serif, italicized font.

```
h1    {font-family: Arial, sans-serif; font-style: italic; font-
size: large;}
```

We also created a special class that changed the font-family to Times and the color to blue.

```
.specialh1 {font-family: Times, serif; color: blue;}
```

However, because we did not specify differences to the font-style and the font-size, the .specialh1 class will retain those property values when displaying the web page. This is due to inheritance.

Contextual selectors

Inheritance saves CSS designers typing. Instead of setting all style properties, one can create defaults and then list the exceptions. To give 'EM' elements within 'H1' a different color, one may specify:

```
h1 { color: blue }
em { color: red }
```

When this style sheet is in effect, all emphasized sections within or outside 'H1' will turn red. Probably, one wanted only 'EM' elements within 'H1' to turn red and this can be specified with:

```
h1 em { color: red }
```

The selector is now a search pattern on the stack of open elements, and this type of selector is referred to as a *contextual selector*. Contextual selectors consist of several simple selectors separated by whitespace (all selectors described up to now have been simple selectors). Only elements that match the last simple selector (in this case the 'EM' element) are addressed, and

only if the search pattern matches. Contextual selectors in CSS1 look for ancestor relationships, but other relationships (e.g. parent-child) may be introduced in later revisions. In the example above, the search pattern matches if 'EM' is a descendant of 'H1', i.e. if 'EM' is inside an 'H1' element.

```
ul li    { font-size: small }
ul ul li { font-size: x-small }
```

Here, the first selector matches 'LI' elements with at least one 'UL' ancestor. The second selector matches a subset of the first, i.e. 'LI' elements with at least two 'UL' ancestors. The conflict is resolved by the second selector being more specific because of the longer search pattern. Contextual selectors can look for element types, CLASS attributes, ID attributes or combinations of these:

```
div p    { font: small sans-serif }
.reddish h1     { color: red }
#x78y code      { background: blue }
div.sidenote h1 { font-size: large }
```

The first selector matches all 'P' elements that have a 'DIV' among the ancestors. The second selector matches all 'H1' elements that have an ancestor of class 'reddish'. The third selector matches all 'CODE' elements that are descendants of the element with 'ID=x78y'. The fourth selector matches all 'H1' elements that have a 'DIV' ancestor with class 'sidenote'.

Several contextual selectors can also be grouped together:

```
h1 strong, h2 strong, h1 em, h2 em { color: red }
```

Which is equivalent to:

```
h1 strong { color: red }
h2 strong { color: red }
h1 em { color: red }
h2 em { color: red }
```

Hyperlinks and CSS

Hyperlinks (or anchors) have a unique set of options that can be used to customize the "look" of the link depending on the hyperlink's status. A hyperlink may have a status of either visited, hover, or active which can then be manipulated to achieve a different presentational "look" on the page. All this can be done without the need for JavaScript or other scripting languages.

Setting up your anchor styles in the style sheet requires a specific order of the different anchor pseudo-classes. The best thing is to remember the saying: **L**o**V**e/**HA**te.

```
a:link          {color: blue;}

a:visited  {color: purple;}
```

```
a:hover    {color: blue; text-decoration: none; background-color:
yellow;}

a:active   {color: red;}
```

The ! Important Property

Using the **! important** property, web designers can force specific styles to appear on a web page. The **! important** declaration will overide all other CSS rules for the specified property and value. Thus, the web page authors can create style sheet rules that will overide those as set by users.

Example:

```
h1 { color: black ! important; background: white ! important }
p  { font-size: 12pt ! important; font-style: italic }
```

This will format the heading 1 styles to have a black text color with a white background. Additionally, all paragraphs will have a 12pt font-size, but may or may not have a font-style of italics.

It is not recommended that web designers use the **! important** property to format web page content. One advantage to using CSS is that there is some flexibility in the manner in which web content is rendered. A low-vision user could create their own style sheet in order to make the on-line content more usable for their needs. The following style sheet is just one example that may be useful to a low-vision user.

```
h1   { font-family: Verdana ! important;
font-style: normal ! important;
font-size: xx-large ! important;
color: black ! important;
}

p    { font-family: Times ! important;
font-style: normal !important;
font-size: large ! important;
color: yellow ! important;
background: black ! important;
}
```

Style Sheets, Part 2

In addition to the various style sheet options that can be designed for the Web page, it is also possible to create style sheets for other media types. These different media types allow to configure the style of print-based media, handheld devices, embossed media, Braille feedback devices, or projection systems. Not all Web browsers can recognize the different media types or reconfigure the different display outputs, so it is best to always test the desired output.

Print Styles

To use a different style sheet for your printed Web page material, simply start a second style sheet with different properties and values. You can then link to this different style sheet in a similar fashion as the Web-based styles. The advantage of having a print-based style sheet is that you can control how the print information will look and what elements from your Web page you wish to include on the printed version. For instance, you could set up a print-based version to not print your images or identify different font parameters (font-family, font-size, color, etc.) for the print-version.

Print Style Sheet link:

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

In addition to setting different font attributes to the printed version of your document (or stripping out certain content), you can also use the print style sheet to include items that may not get printed normally. For instance, if there is a hyperlink on the page but no actual Web address, then it becomes difficult for the user to identify what (or where) the hyperlink was actually linking to on the Web-based version of the page. With a print style sheet, you can set the printed version to print out the URL of hyperlinks as well. (View more examples of print style sheets can be found in Eric Meyer's article at http://www.alistapart.com/articles/goingtoprint)

Print URLs:

```
a:link:after, a:visited:after    {content: " (" attr(href) ") ";}
```

One additional item to consider when using print style sheets is if the `float:` property is utilized in the screen-based style sheet. If so, then this can cause problems as to what content may actually get printed, regardless of the styles set in the print style sheet. An easy solution is to simply set `float: none;` in order to ensure the material is printed correctly.

Print CSS Float Example:

```
body {float: none !important;}
```

Web Resources

W3Schools – Learning CSS
http://www.w3schools.com/css/default.asp

Units of Cascading Style Sheets
http://www.w3schools.com/css/css_units.asp

Colors for Cascading Style Sheets
http://www.w3schools.com/css/css_colors.asp

Additional Information: http://www.w3schools.com/css/css_colornames.asp

W3C Cascading Style Sheets Information
http://www.w3.org/Style/CSS/

CSS ZenGarden
http://www.csszengarden.com/

CSS Layout and Styles
http://glish.com/css/

http://icant.co.uk/csstablegallery/

http://www.alistapart.com/

Web Developer Toolbar for Firefox
http://chrispederick.com/work/webdeveloper/

Web Developer Toolbar for IE
http://www.nils.org.au/ais/web/resources/toolbar/index.html

Additional Information: http://www.webaim.org/techniques/articles/aistoolbar