Style Sheets for HTML

Joe English

November 18, 1994

Contents

Abstract

This is version 0.1 of a proposal for a style-sheet mechanism suitable for use on the World-Wide-Web. It is primarily oriented towards HTML level 2.0, but may be used to support other SGML document types as well.

This proposal defines a set of style properties, mechanisms for assigning those properties to document elements, and mechanisms for associating collections of style specifications with HTML documents. It also includes suggestions for managing users preferences within a browser and how to resolve user preferences with externally supplied style specifications.

1 Introduction

[[Throughout the document you may find editorial comments like this one. These are notes to myself reminding me of what needs to get written and other issues that need to be discussed. At this point, the document consists ***mostly*** *of editorial comments.]]*

A *stylesheet* is a collection of style specifications prepared by the document author. Stylesheets are defined as an SGML document type. Stylesheets are stored and transmitted separately from the HTML document.

Goals

[[Fill this part in. Some goals: style sheets should not be ***too*** *much harder to write and parse than HTML; should provide functions that the WWW provider community has demanded; must be consistent with principles of generalized markup; should be predictable; must support variety of display devices; should be applicable to non-HTML document types; must be extensible; probably others.]]*

Notes on terminology

In this document, the term *author* refers to a person or program which creates hypertext documents. A *designer* is a person who creates stylesheets. A *browser* is software which displays HTML documents. A *user* is the person who uses a browser, and the term *reader* refers to the combination of the browser and the user.
Note: For example, under this proposal a *designer* is able to specify colors but an *author* is not (unless she is also the designer). *Browsers* should use the indicated colors if possible unless the *user* instructs otherwise. In other words, color specifications appear in stylesheets, not in HTML documents, and whether or not the color specification is honored depends on platform capabilities and user preferences.
[[Define: applicable, active, relevant.]]
[[Define: source document, style sheet, preference sheet.]]

Platform capabilities

Documents may be displayed on a wide variety of output devices: high-resolution color bitmapped displays, VT100 terminals, laser printers, speech synthesizers, and many others. The different devices may be classified by individual features:

- Character cell vs. bitmapped. Are inline graphics supported? Can multiple typefaces be displayed?
- Interactive vs. static. Can the user interact with the document? (Selecting links, submitting forms, etc.)
- Paginated vs. continuous scroll. Do “page breaks” make sense? Are there headers and footers? Should intra-document hyperlinks include a page number?
- Color vs. grayscale vs. monochrome.
- Visual vs. audio.

  [[This proposal tries to address browsers of all combinations of capabilities, but speech synthesizers and other non-visual browsers are sorely underrepresented. ***Any*** *feedback on how to better support them would be greatly appreciated.]]*

[[For static devices like paper, need to print textual locator for anchors (URL in footnote maybe?), different layout for forms (may want to print a “blank” form to be filled in with pencil, or may want to print whateve values were interactively selected.)]]

Browsers are free to ignore any properties which they are unable to render, and are in fact encouraged to ignore any which would detract from a uniform interface.

Note: For example, a browser which highlights hypertext anchors by underlining them is encouraged to ignore any underline specifications in the stylesheet.

2 Properties

2.1 Units of Measurement

Conventional units of measurement like inches, points, and centimeters are not generally applicable for device-independent display. Specifying dimensions in physical units will not have the same effect on different output devices: a one-inch left and margin may be rather narrow on A4 paper, but can use up a good portion of the display real estate on small monitors.

To address this, stylesheets provide various units of measurement which are defined relative to

- The total display size (`pcd`, `nlh`)
- The current font (`em`, `en`, `ex`)
- The display resolution (`p`).

Dimensions are specified as integer decimal numbers followed by an alphabetic suffix indicating a unit. (In SGML terms, a `NUTOKEN`). For example, a horizontal space of `75pcd` is equal to 3/4 of the display width.

Question: Should floating point numbers be allowed in dimension specifications?

[[Distinguish between horizontal, vertical, and thickness units?]]

[[Discuss: relative specifications like `+10pcd`*; very useful for things like margins.]]*

Font-relative units

The `em` unit is the typographical “em width”. It is defined by the current font and is typically equal to the width of a capital letter `M`. This unit may only be used in horizontal contexts.

The `en` unit is the typographical “en width”. It is defined by the current font and is typically equal to one half of an `em`. This unit may only be used in horizontal contexts.

The `ex` unit is the typographical “x height”. It is defined by the current font and is typically equal to the height (from baseline to top) of a lowercase letter `x`.

The `lh` unit stands for “line height”. It is defined as the normal distance from baseline to baseline (including leading) of the current font. For example, if the current font is 10 point Times-Roman with two points leading, `1lh` is equal to 12 points. Vertical contexts only.

[[Discuss leading: since the required leading is highly dependent on the actual font, it is specified by the reader, not the designer. Designers may however ***increase*** *the linespacing to get “double-spacing” etc.]]*

Display-relative units

The `pcd` unit stands for “per cent of display”. The valid range is between 0 and 100 inclusive. When used in a horizontal context, `1pcd` is equal to 1/100th of the total display width; when used in a vertical context, `1pcd` is equal to 1/100th of the total display height.

The *display size* is defined by the output device:

- For a GUI br,owser this should be the size of the window used to display the document (*not* the total screen size). If the window is resized, a `pcd` should be adjusted accordingly.
- For hardcopy output, this should be the paper size minus the top, bottom, left and right margins.
- For character-cell terminals, there isn't much choice.

Note: To take character-cell browsers into consideration, designers should specify indentation and other horizontal dimensions in even multiples of `5pcd`. This corresponds to 5 spaces on an 80-column terminal.

The `nlh` unit stands for "normal line height". It is defined as the normal distance from baseline to baseline of the normal body font at the normal size.

Note: While `1lh` may mean different things at different points in a document, `1nlh` always refers to the same height regardless of the current font.

The `p` unit is used for fine-resolution spacing and for specifying line thickness. It represents the finest useful resolution available on the output device.

For bitmapped display devices, `1p` should be interpreted as one pixel.

For laser printers, `1p` should be interpreted as one point when used as a horizontal or vertical space. When used to specify line thicknesses, it should be interpreted as something finer, like a decipoint.

Note: The idea is that a `1p` thick line should be a "hairline" rule, and vertical space of `1p` should be the smallest amount of space easily visible to the eye.

Physical units

Question: Should physical units like "inches" and "centimeters" be provided as well?

[[Suggest using TeX's two-letter abbreviations as a standard. Check TeXbook, add list here.]]

Notes on units of measurement

Since some of the units are dependent on the current font, it is important to apply font specifications *first*, so that any font-relative width and height specifications are based on the correct font.

Certain naming conventions are used for dimension attributes. Names ending in "`skip`" refer to vertical space, "`sep`" to horizontal space. "`thick`" is used for line thicknesses, and "`width`" and "`height`" are used for the dimensions of objects.

[[Allow also browser-configured dimensions? E.g., bigskip, medskip, smallskip for vertical space, bigspc, medspc, smallspc, thickspace, thinspace, for horizontal? Horizontally: thinspace < thickspace <= 1en/tiny <= smallspc <= 5pcd <= medspc <= 20pcd <= bigspc <= 40pcd]]

[[The units defined here are not sufficient for producing fine typography. Then again, if you're interested in fine typography you won't be using HTML or these stylesheets anyway; provisions for paper are only included for users who want to get hardcopy for personal use, not to produce publishable output.]]

2.2 Fonts

Specifying fonts is problematic.

Font sizes may vary considerably: a 12 point font is small on a 1024x1024 X terminal, but large on a classic Mac. The only font families that are available on even half of the common platforms are Times Roman, Helvetica, and Courier; however, if a user has any fonts *other* than these three available, she will probably want to use those instead! Designers have no way of knowing what fonts will be available to the reader, or even what font formats the reader can use. Even if the browser and server could negotiate on a common font format, say PostScript Type 1 or X BDF, there are serious copyright implications involved in shipping fonts across the network.
[[I don't think mechanisms for font downloading are even worth considering at this point, given the limited number and quality of freely distributable fonts. But see p. 8 where it's briefly considered anyway...]]
Consequently, fonts are specified in the style sheet with three parameters:

- A logical font family
- A logical font size
- A font shape

This scheme provides a limited "logical palette" of fonts for designers to choose from, and readers are able to select the actual typefaces and sizes to which the logical fonts map.
Note: It will also be necessary to specify the text encoding and directionality, but these are more properly properties of the document itself, not just presentation hints. This proposal does not address these issues, except to say that browsers could use encoding and directionality as extra parameters to be used in font selection.

2.2.1 Font families

The `fontfam` style property specifies one of the following logical font families:

normal A text font, used for the main body text.

heading A display font, suitable for headings and figure captions.

fixed A monospaced font, suitable for displaying code listings and other preformatted text.

alternate A second text font, suitable for body copy but visually distinct from the `normal` font. Might be a sans-serif font if the main font is roman, or vice-versa.

[[Rename "normal" family to something like "text" or "body". The word "normal" is overloaded.]]
The set of actual fonts to which each logical font family corresponds is configured by the reader.
Question: Is this scheme totally harebrained, or only partially so?
Note: The rationale behind this particular choice of font families is based on certain observations: it is fairly common for section headings to use a different font family than the body text, and there are many cases where having a second text font is useful (e.g., a document with annotations might display the primary text in the `normal` family and annotations in the `alternate`, or a designer could use `alternate` for figure captions and sidebars.) The `fixed` family is necesary to support preformatted text. Suggestions are welcome. See also p. 7 for other notes.

2.2.2 Font sizes

The `fontsize` property specifies a logical font size. As with the font family, the actual size in points or pixels is determined by the reader.
[[There are two different options. Not sure which one is best.]]

2.2.3 Alternate 1: names

The font size is specified as one of the following name tokens, from smallest to largest:

- `tiny`
- `small`
- `normalsize`
- `large`
- `big`
- `huge`

Note: The names are based on those used in LaTeX. LaTeX actually has a wider range of sizes, since it distinguishes `\large`, `\Large`, and `\LARGE`. The same scheme could be used for stylesheets, except that case is not significant in SGML name tokens. (Declaring the **fontsize** attribute as `CDATA` would prevent validating parsers from checking its value.)

2.2.4 Alternate 2: Numbers

The `fontsize` property is specified as an integer between 0 and 7 inclusive, with 0 being the smallest, 7 the largest, and 3 the "normal" or default size.
Note: This was taken from Netscape.
Note: The Netscape `<BASEFONT>` tag – which changes style properties asynchronously with the document structure – is not supported in this proposal.
Question: For font sizes, which is preferable: numbers or names?
[[I think using numbers is slightly better than names for a couple reasons: it's less ambiguous ("Is large larger than big or is big bigger than large?"), it provides a slightly wider range of sizes (I ran out of adjectives), and it allows for relative font size specifications like `<style gis="emph" fontsize = "+1">`*.]]*
[[Another option is to combine the two: allow large 1 2 3 4, huge 1 2 3 4, small 4 3 2 1, tiny 4 3 2 1 for users who like lots and lots of variation in font sizes. They can configure their browsers to use them; for a more uniform layout only the names would be used. This may enhance predictability too.]]

2.2.5 Font shapes

The `fontshape` property is used to select a variant of the current font family. Legal values are:

plain The "normal" or default typeface for this family (Times Roman, Computer Modern Roman).

bf A heavier variant of the plain font (Times Bold, Computer Modern Bold Extended).

it An italic, oblique, or slanted variant of the plain font (Times Italic, Helvetica Oblique).

bi A heavier variant of the italic family (Times Bold Italic, Helvetica Bold Oblique).

sc A caps-and-small-caps variant of the plain font. (Computer Modern Small Caps).

tt A "teletype" or "computer" style font; probably taken from a different actual font family than the `plain` face, but should be chosen to be visually compatible with it (Courier, Lucida Teletype).

Note: Character cell browsers may use terminal emphasis to emulate font shapes; for example, "bright" or "standout" mode for boldface and underlining in place of italics. Reverse video is *not* recommended for this purpose.
Note: Inventive browsers may choose to fake the `sc` variant by using the capital alphabet from a single typeface at two point sizes. This is not generally recommended for print, but it should look OK on low-resolution bitmapped displays. Another option is to simply use the `plain` variant and fold lowercase letters to uppercase.
Question: Is the `sc` variant even necessary? It looks good for headings, and may be useful for trademarks, but it probably isn't vital.
Note: Unlike the `fixed` family, the `tt` variant need not be a fixed-width font.

2.2.6 Notes on font selection

The only requirements for browsers concerning font selection is that they *must* provide at least one variant of the `fixed` family, and that all `fixed` variants must be monospaced. This is to support the HTML **PRE** element and other constructs where character spacing is critical to the meaning of the document content. If *all* variants in the `fixed` family have identical metrics, even better, but designers should be aware this is not necessarily the case (e.g., text in `<style fontfam = fixed fontshape = bf>` might not align with text in `<style fontfam = fixed fontshape = it>`.)

It is suggested that family and size changes only be applied to block-level elements like headings and paragraphs; shape specifications (and colors and special effects) may be applied anywhere. This suggestion is primarily for designers, though it would be reasonable for browsers to enforce it by ignoring family and size changes on phrase-level elements.
[[I had a rationale for saying this, but I've forgotten what it was. Unless it was just that "inline point size changes are ugly", in which case ignore the last paragraph.]]

2.2.7 Suggestions for "cutting corners"

Readers need not provide a unique typeface for every combination of family, shape, and size. This is in fact desirable, since there are a large number of combinations and loading lots of fonts can be resource-intensive.

Some suggestions for limiting the number of actual fonts used are listed here.

- Can use the same set of fonts for the `alternate` and `heading` families;
- Can use the same set of fonts for the `alternate` and `normal` families;

- Can use the same set of fonts for all three of `alternate`, `heading`, and `normal`. This help prevent "spreading fonts across the page like peanut butter"

  [[Attributiondammit?]]

- Can use the `fixed` family in place of the `tt` shape for all other families.

- Can provide three sizes instead of six or eight, mapping 0-2 to "small", 3 to "medium" and 4-7 to "large". (Or 4-5 to "large" and 6-7 to "huge".)

- Need not provide small sizes for the `heading` family;

- Need not provide anything other than `normalsize` for `alternate` and `fixed` families.

- Browsers *should* provide all sizes for the `normal` family, in case designers wish to use a single font family for all elements.

2.2.8 Open questions

In some respects, the overall approach specified here is too limiting ("I want my ransom note to display just like I wrote it!"), and in other respects there are too many choices. I think that four logical families strikes a good balance between flexibility for authors and configurability for readers, but suggestions are welcome.

This section lists some of the open questions and alternatives to consider.

2.2.9 Multiple shape specifications

Question: Should shape specifications be cumulative?

Some authors may expect bold and italic specifications to have a cumulative effect, i.e., that an "italic" phrase inside "bold" text should be rendered in "bold italic."

[[The utility of this has always puzzled me, but every Mac program, desktop publishing application, and even LaTeX2e seem to think that font selection should work like that. I think the notion that you can do arithmetic with fonts that way is misleading: Times Bold Italic is not just Times Roman plus bold plus italic.]]

If shape specifications accumulate, this can lead to a combinatorial explosion of different typefaces. Not all the combinations will be available (or even make sense! Bold italic small caps?). The interaction between cumulative shape specifications, font availability, and the other inheritance mechanisms proposed here will be very hard to predict.

[[I've had problems with NFSS along the same lines.]]

[[I'm inclined to say that font shapes should not be cumulative, but I do have some ideas for how a priority scheme could work in case the majority feel this is a good idea.]]

2.2.10 Other schemes

Question: Would further decomposition of the `fontshape` parameter be useful?

For example, could take the NFSS approach and specify series and shape as separate parameters.

Could further decompose into weight, width, and slant, a la [8] and several others.

Could come up with a comprehensive list of logical font properties (serif vs. sans, roman vs. transitional vs. modern, upright vs. oblique, large x height vs. small x height, and so on) and have browsers pick the best match based on available fonts.
[[Need to check out Panose system. Sounds like it does this sort of thing.]]

2.2.11 Specifying actual fonts

[[It is my opinion that any font selection scheme that does not leave the final choice of fonts solely to the browser and user is doomed to failure. Nevertheless...]]

Could add a **FONTDESC** element, which would specify an actual font to be used for a particular logical font. It would contain multiple **FONTSPEC** elements, one for each known platform and/or font format:

```
<fontdesc fontfam=normal fontsize=normal fontshape=it>
    <fontspec notation=XLFD>
-adobe-times-medium-r-normal--*-120-*-*-*-*-iso8859-1
    </fontspec>
    <fontspec notation=postscript>
/Times-Roman findfont 12 scalefont
    </fontspec>
    <fontspec notation=TeX>
cmr10 at 12pt
    </fontspec>
    <fontspec notation=NFSS>
cmr/m/n/12/14
    </fontspec>
    <fontspec notation=MSWindowsFont>
???
    </fontspec>
    <fontspec notation=MacintoshFont>
???
    </font>
</fontdesc>
```

This would quickly get painful, since there would need to be a different **FONTDESC** for every combination of family and style used in the stylesheet.

The size parameter could be factored out into a separate mapping from logical sizes to actual point sizes and leading. Factoring out any other attributes will be difficult, since although some schemes support it – NFSS and partially XLFD – others do not at all (TeX and PostScript).

Could let designers specify new sets of logical font families. This would leave browsers that didn't grok the font notation with no fallback though.

2.3 Special effects

[[Various options: undlerline, strikethrough, blinking (what the hell), reversed type, boxed, others. See DTD.]]

2.4 Colors

All the colors used by a style sheet must be declared in the (optional) **COLORS** element. This element contains one or more **COLOR** elements, each of which specifies a single color.

COLOR elements have two required attributes: **ID**, a unique identifier, and **RGB**, which defines the color. Colors are referenced by ID.

Colors are defined by their red, green, and blue components using the X11 hex notation: a pound sign followed by 3, 6, 9 or 12 hexadecimal digits. The digits are interpreted as three groups of 1, 2, 3 or 4 half-bytes, the first specifying the red component, the second green, and the third blue. Hex digits `A` through `F` may be upper or lower case.

```
<stylesheet>
<!-- Example of color specifications -->
<colors>
    <color id=red rgb="#F00">
    <color id=green rgb="#00FF00">
    <color id=blue rgb="#000000000FFFF">
    <color id=grey rgb="#c0c0c0">
    <color id=white rgb="#FFFFFF">
</colors>

<!-- Highlight all code sections in blue: -->
<style gis = "code kbd pre" fgcolor=blue>
</style>

<!-- Headings in red on grey, for whatever unholy reason: -->
<style gis = "h1 h2 h3 h4 h5 h6" fgcolor=red bgcolor=grey>
</style>

</stylesheet>
```

Note: Under this scheme, all the colors used by a stylesheet are listed in one place, so browsers can do colormap allocation up front if necessary. This also makes it easier to modify color specifications if the designer (or the user!) doesn't like them.

Question: Is there a better notation for RGB than the hex notation?

Three comma-separated decimal numbers from 0-255 maybe?

Question: What's a better way (better than RGB) to specify colors?

Note: RGB values are probably not be the best way to specify colors. Any color-capable implementation should be able to interpret them, but not with identical results on all platforms.

[[HSV? YUV? Pantone numbers? TekCMS? A predefined set of color names? (This may be hard to come by: even the supposedly standard X11 color database does not define the same set of names from platform to platform.) Should a gamma correction be included? Suggestions are welcome.]]

Note: It will be possible to include multiple attributes on **COLOR** elements, one for each color specification scheme. Browsers would use whichever specification format they understood, falling back on the RGB attribute (which would still be required) if none of the others are understood.

2.5 Alignment and placement

[[The usual: ***ALIGN*** *attribute, can be left, center, or right.]]*
[[Discuss: perhaps `start` *and* `end` *would be better names than* `left` *and* `right`*, since HTML will not always be exclusively ISO Latin 1 left to right. Then again, left, center, and right are pretty deeply ingrained in the collective unconscious.]]*
[[Discuss: difference between alignment and placement; center alignment specified on block-level elements should affect line placement on contained paragraphs, but not in contained display elements like ***PRE*** *and* ***TABLE*** *(or should it?); these should be centered as a unit instead.]]*
[[Discuss: how HTML 3 alignment attributes fit into this scheme (discussed elsewhere actually); how Netscape's ***CENTER*** *element fits in.]]*
Note: HTML 3 proposes `JUSTIFY` and `INDENT` as possible of **ALIGN** attribute values. Since these options are orthogonal to alignment, they are specified on separate attributes in the stylesheet DTD.
Note: The HTML DTD specifies `TOP`, `MIDDLE`, and `BOTTOM` as options to the **ALIGN** attribute on **IMG** elements. In the stylesheet DTD, **ALIGN** is only used for *horizontal* alignment. Vertical alignment – top, middle, and bottom – is specified on the **VALIGN** attribute.

2.6 Separator specifications

A *separator* is defined here as any formatting construct that separates what comes before with what comes next.
[[Duh.]]

Separators may be explicit, such as an HTML **BR** or **HR** element, or implicit, such as before and after block-level displays like headings, addresses, and forms. Visually, separators can take many forms: vertical whitespace, horizontal rules, inlined graphic images like that stupid Kilroy GIF, or any combination of the above. For example, some user manuals have a thick rule, a few points of whitespace, a thin rule, and an inch or so of whitespace before each chapter heading.

There are too many combinations to specify using a fixed set of attributes, so the style sheet DTD includes a **SEPSPEC** element for defining more complex separators.

SEPSPEC elements have an **ID** attribute, a unique identifier by which they are referenced. They may contain any number of **HRULE**, **VSPACE**, and **IMAGE** elements. (These roughly correspond to the HTML **HR**, **BR**, and **IMG** elements, but are given different names to avoid confusion, since they have slightly different attributes.
[[Or perhaps to introduce confusion. I'm not sure.]]
Question: Should the HTML names **HR**, **BR**, and **IMG** be used instead of **HRULE**, **VSPACE**, and **IMAGE**?

```
<stylesheet>

<note>
Display a page-wide a thick rule and a hairline rule
followed by 4 lines of whitespace before each chapter heading.
</note>

<sepspec id=chapsep>
    <hrule thick = 3p width = 100pcd align = center>
```

```
    <vspace vskip = 3p>
    <hrule thick = 1p width = 100pcd align = center>
    <vspace vskip = 4nlh>
</sepspec>
<style
    gis = "h1"
    fgcolor = blue
    presep = chapsep
    postskip = 3nlh
>
</style>
<note>
Since the only separator needed *after* the
chapter heading is 3nlh of whitespace,
that is specified on a source attribute (postskip);
SEPSPEC elements aren't needed for simple
cases like this.
</note>
</stylesheet>
```

Each of these elements and their attributes are defined below.
Note: The **SEPSPEC** scheme adds complexity, and doesn't do much that can't already be done in (Netscape) HTML. However, it may make document creation and maintenance easier and may help to keep information-free tags out of source documents.

Vertical whitespace

The **VSPACE** element is used to specify vertical whitespace in a separator. It has one attribute, **VSKIP**, which is a vertical dimension defining the amount of space to insert.

Rules

The **HRULE** element specifies a horizontal rule. It has several attributes:

width The width of the rule. Horizontal dimension.

align How to place the rule if it is narrower than the display width. Legal values are `left`, `center`, and `right`.

thick The rule thickness. Default value is `1p`, i.e., a hairline rule. If a value is given for **thick**, it should be in `p` units.

*[[Netscape uses **SIZE** for this attribute. "Thick" is more descriptive, I think.]]*

fgcolor The foreground color of the rule. If present, must be the ID of a **COLOR** element defined in the stylesheet.

*[[May need other attributes. A common presentation style is a Motif-style "3-D" effect with top and bottom "shadows"; maybe need a **shading** attribute with legal values* `etchedin`, `etchedout`,

and `solid`*. Probably should* ***not*** *specify top and bottom shadow colors, as these are best computed by the browser from the foreground color.]]*

Inline images

[[Move this section elsewhere, as it applies to more than just separators.]]

The **IMAGE** element defines a bitmapped image which may be used as an icon, bullet, dinbat, or rule.

[[Is "dingbat" a technical term?]]

IMAGE has the following attributes:

URL A Uniform Resource Locator specifying where to retrieve the image. If this is a relative URL, it should be interpreted relative to the URL of the *stylesheet*, not the base document.

ID An optional unique identifier

REFID If present, specifies the ID of another **IMAGE** element in the stylesheet. This allows images to be conveniently reused.

[[Ought to specify the different image formats that are allowed, but since HTML doesn't we don't either. In practice, only `image/gif` *is universally supported, and that's probably the best format for rules and dingbats anyway.]]*

When used as a vertical separator, the following **IMAGE** attributes are also used:

align Defines how the image should be placed on the page. May be one of `left`, `center`, or `right`.

[[Someone somewhere sometime posted a set of proposed attributes for the HTML ***IMG*** *element for when images were used as rules. It defined options like how the image should be truncated if it was too wide, if it should be centered or tiled if it was too narrow, and a couple others. I cannot for the life of me find this posting in any of my archives. Anyone remember this? Anyway, something similar should go here.]]*

IMAGE elements may appear outside any other elements in the stylesheet. They may be used inside **SEPSPEC** separator specifications, or referenced by ID on **STYLE** elements as, e.g., bullets to use for list items.

```
<stylesheet>

<image id = kilroy
    url = "http://www.art.com/bitmaps/stupid-kilroy-gif.gif">

<image id = rainbow
    url = "http://www.art.com/bitmaps/stupid-rainbow-gif.gif">

<image id = blueball
    url = "http://www.art.com/bitmaps/blueball.gif">

<note>Use "Kilroy was here" picture for all HRs</note>
<style gis = "hr"
```

```
    icon = kilroy
    preskip = 2p
    postskip = 2p
>
</style>

<note>Use rainbow line after all 1st-level headings</note>
<sepspec id = h1sep>
    <vspace vskip = 2p>
    <image refid = rainbow>
    <vspace vskip = 2p
</sepspec>
<style gis = h1 postsep = h1sep>
</style>

<note>Blue balls for bullets</note>
<style context = "ul li"
    icon = blueball
>
</style>

</stylesheet>
```

[[Need different properties for rule icons and bullet icons? "icon" is used in two places.]]

2.7 Enumeration rules

*[[Basic stuff for **OL**s: **numstyle** attribute with legal values* `arabic`, `lcroman`, `ucroman`, `lcalpha` *and* `ucalpha`.*]]*
[[Netscape uses `1`, `i`, `I`, `a`, *and* `A` *to mean the same things, but that means it can't be validated by an SGML parser. They also use the **TYPE** attribute, which is already used for **INPUT** with a different meaning; trying to avoid overloading names too heavily in this proposal.]]*
*[[Could also add more complex **ENUMSPEC** specifications to allow stuff like* `Section 2.2` *and* `Figure (2.2a)` *to be automatically generated. Check out FOSIs and DSSSL for ideas...]]*

2.8 Generated text

*[[This is text that's automatically inserted for phrase-level elements and a few others. May not even be necessary, but would be nice. Example: Lynx used to put asterisks around **STRONG** text and underlines around **EMPH**, like the Usenet conventions. Personally, I liked that a lot better than what it does now.]]*
[[Include: prefix, suffix text.]]
[[Maybe include before and after text also. (distinction: prefix and suffix text displayed in the same style as the element; before and after text displayed the same as the containing element.)]]
[[Maybe maybe include: "message text" for graphical browsers; a short message that could be displayed on the status line whenever the pointer waves over an element. Useful on anchors, possibly

on other semantic elements.]]

3 Structure of stylesheets

The format of stylesheets are formally defined in SGML by the stylesheet document type definition. This DTD requires an SGML declaration with an increased `NAMELEN` parameter, such as the the SGML declaration for HTML.
[[Oops! This one won't work – it needs a vastly increased `ATTCNT` *also. Default is 40, too small. Time to reorganize the DTD...]]*

Typical stylesheets will look like

```
<!doctype stylesheet PUBLIC -//ART.COM//DTD Style Sheet//EN//DRAFT>
<stylesheet>
<note>
ednote: put more complex example here.
</note>
</stylesheet>
```

Question: Can I borrow an owner identifier prefix for the FPI? `-//IETF//` maybe?
Note: If acceptable, the FPI for the style sheet DTD will be something like `-//IETF//DTD HTML Style Sheet//EN`

Stylesheets consist chiefly of **STYLE** elements. Most of the style processing is specified by attributes of **STYLE** elements.

There are two major categories of **STYLE** element attributes:

qualifiers which are used to determine when the style element is applicable, and

specifiers which define the properties to be applied to the document when it is.

Processing model

When a browser encounters a start-tag in the source document, it looks up a *single* **STYLE** element in the style sheet, determined by qualifier attributes. Each specifier attribute found on the style element is used to modify a style property. There is a one-to-one correspondence between specifier attributes and style properties. If no specifier is found on the **STYLE** element for any property, the value of that property is inherited from the parent element.

When an element ends (by an explicit or implied end-tag), all style properties revert to their previous values (i.e., those of the parent element).
Note: Browsers should maintain a stack of style properties that is pushed and popped in parallel with the stack of open document elements.
Note: Not all properties are inherited by default. For example, a separator specification for a **FORM** element (which may specify, for example, that a horizontal rule should be drawn before and after the form) would not be inherited by children of the form. That a property should not be inherited is indicated in the stylesheet DTD by an attribute with a non-`#IMPLIED` default value.
[[This won't work. ***SEPSPEC*** *elements are referenced by* `IDREF` *attributes, which can't be defaulted in the DTD.]]*
[[If no style element is found, may need to resolve to an artificial empty `<STYLE>` *tag to override non-inheritable properties.]]*

Other elements

Other elements such as **COLORS**, **IMAGE**, and **SEPSPEC** may appear in the stylesheet to define properties that are too complex to be specified on a single attribute. These are described in detail elsewhere.

The **NOTE** element may appear just about anywhere; it's for including human-readable comments in the stylesheet. This may only contain plain text (`#PCDATA`).
Note: No HTML markup is allowed inside **NOTE** elements.

STYLE element content

Question: What should the content model for **STYLE** elements be, and what should it mean?

All of the important information is currently stored on attributes. Since all intra-stylesheet linking is through id references, the semantics of including other elements inside **STYLE** elements another is not yet defined.

There are several possible meanings for the **STYLE** element's content model:

Inheritance All **STYLE** elements contained in another automatically inherit properties from the parent.

Qualification Elements contained in a **STYLE** element are only applicable when the outer element is active. (That is, the content of a **STYLE** element is an implicit style set.)

Specification Elements like **SEPSPEC**s and **IMAGE**s inside **STYLE** elements could be used to specify properties. The contained elements would have a **PROPERTY** attribute which names the property they specify (e.g., `<SEPSPEC property=postsep ...>`).

Illegal **STYLE** elements could just be declared `EMPTY`. This might be the best option, since it's really easy to forget the `</style>` end-tag.

*[[Currently leaning towards using containment for specification. This seems the most useful. It may also be necessary to partition style properties into smaller groups and use a different element for each group, since the number of attributes is quite large; then a **STYLE** element could contain, say, a **PARSTYLE**, a **BLOCKSTYLE**, and a **CHARSTYLE**, each of which would specify a set of properties.]]*

4 Specifiers

This section describes all the style attributes applicable to each HTML element. Rather than list each HTML element individually, they are grouped into "classes" or "architectural forms". For example, the HTML elements **H1** through **H6** all have the architectural form `heading`, and all use the same style attributes.

The complete list of architectural forms used in this proposal are semi-formally defined by the WWW-Arch meta-DTD.
[[Even if the rest of this proposal turns out to be worthless, the WWW-Arch DTD might be useful on its own. If it ever gets finished.]]

[[Discuss: various "format" properties, which can affect what other specifiers are applicable. E.g., if a heading has `headfmt = display` *the block-level attributes apply; if it has* `headfmt = runin` *then the paragraph attributes do.]]*
[[Discuss: "fallback" values; suggestions for defaults if a browser does not support a property to enhance predictability.]]
*[[Discuss: Even if a style specification is is not **relevant** to an element, it should still be **applied** to the current property set, as it may be relevant to contained elements.]]*

4.1 Document-wide properties

To specify initial or global stype properties, designers may use a **STYLE** element applicable to the **HTML** or **BODY** element. Properties specified there will be inherited by all other document elements.
Note: Even if the source document does not contain explicit `<HTML>` or `<BODY>` tags, browsers will imply their presence since they are contextually required in the HTML DTD.

```
<stylesheet>

<style gis = body
    fontfam = normal
    fontsize = normalsize
    fontshape = plain

    lmargin = 5pcd
    rmargin = 5pcd

    parindent = 2em
    parskip = 1lh
>
</style>
</stylesheet>
```

4.2 Phrases

All the HTML "highlighting" or "phrase-level" elements. They may contain text, links, inline images, and other phrase-level elements; they may appear inside paragraph-level elements, headings, list items, and links. (And, in "non-strict" HTML, at block level as well. In other words, just about anywhere.)

HTML elements: **b cite code em i kbd samp strong tt var** .

The following style specifiers are applicable to phrases:

Font properties The font specification properties **fontfam**, **fontsize**, and **fontshape**. See 2.2.

Special effects Various special effects like underlining and strikethrough. See 2.3.

fgcolor The foreground color. See 2.4.

[[There are a bunch of others, I'm sure. Can't think of them right now.]]

The background color may only be changed on block-level elements.

Question: Should it be possible to specify the background color as well as the foreground color for phrase-level elements? How should this be rendered?

4.3 Blocks

HTML elements: **address? blockquote**

[[Also the proposed HTML 3.0 ***FIGURE****, others?]]*

[[Are ***ADDRESS*** *and* ***BLOCKQUOTE*** *different forms? Blockquote allows block content in strict DTD; ADDRESS only allows text content.]]*

Attributes

lmargin The left margin. A relative horizontal dimension.

rmargin The right margin. A relative horizontal dimension.

bg The background color for this block. Must be the ID of a **COLOR** element defined in the stylesheet. See 2.4.

preskip Amount of vertical whitespace to insert before the element. A vertical dimension.

postskip Amout of vertical whitespace to insert after the elemnt. A vertical dimension.

presep The ID of a **SEPSPEC** element defined in the stylesheet; this separator should be displayed before the element. If **presep** is specified, then **preskip** should be ignored.

postsep Same as **presep** except inserted after the element.

[[Need to add block-level special effects like frame styles etc. Maybe include Motif-like shadow styles for "boxes"? Boxed block elements need to specify inner and outer margins, placement (left, center, right); should text flow around them?]]

Note: The **presep** and **postsep** separator specifications should be formatted using the style properties defined for the block element. For example, if a **STYLE** for a block-level element specifies a foreground color and a **SEPSPEC** which includes a horizontal rule (see example on p. 11), the rule should be displayed in the specified foreground color just like the other block contents.

Note: Browsers may add an extra "outer margin" around the display. For graphical browsers this could be a few pixels; for hardcopy it may be an inch or so. Note that the outer margin is *not* included in the total display area, so for example a width of `100pcd` on 8.5 by 11 paper would be 6.5 inches, not 8.5, if there were 1-inch outer margins.

4.4 Paragraphs

The `paragraph` class is the basic unit of formatting. Paragraphs may contain text, phrase-level elements, inline displays, and some block-level elements like lists.

HTML elements: **dd li p**

Note: Note that the list elements **LI** and **DD** are treated just like paragraphs for formatting purposes.

Style attributes

All attributes relevant to blocks, plus the following:

`align` Line justification. See 2.5.

`parindent` Horizontal dimension. Specifies the first-line indentation.

[[margins, space before, space after; get list from DTD.]]

4.5 Hyperlinks

*[[HTML **A**, HyTime* `clink`. *The HTML **LINK** element is* `metainfo`, *not* `hyperlink`.*]]*
[[Currently trying to decide whether it should even be legal to specify styles for anchors. This is one area where consistency across documents is of the utmost importance. Then again, providers might want to specify it anyway, and users might not mind. This might be best left to browser implementors and users to decide.]]
[[Possibilities: the usual special effects: underlined, boxed, colors, fonts and so on. Other options: marginal icons, addition to a "links" menu, others?]]

4.6 Lists

HTML elements: **dir dl menu ol ul**

There are two different classes of lists in HTML: single-part lists like **OL** and **UL** and two-part lists like **DL**.

Each entry in a two-part list contains zero or more "term" parts (the **DT** element) and zero or more "definition" parts (**DD**).

Single-part lists may be thought of as a special case of two-part lists, where the "definition" part is taken from of the list item (**LI**) element and the "term" part is automaticallyy generated by the browser (e.g., a number for **OL** elements, a bullet for **UL**s, or nothing for **MENU**).

Further, the "term" part is conceptually very similar to a heading, and the "definition" part is just likk a paragraph. This is exactly how this proposal treats all lists: a `list` contains a sequence of `headings` and `paragraphs`, where the headings may be automatically generated.

4.7 Inline Displays

HTML elements: **img input**

4.8 Block Displays

HTML elements: **option pre textarea**
[[For display and inline forms, need subsidiary attribute describing the type of display. This is application-specific (e.g., IMG, TABLE, MATH, etc.) but must have one reserved value (`TEXT`*?) indicating that the contents are also formattable text. Ditto for* `float` *form.]]*

4.9 Headings

HTML elements: **dt h1 h2 h3 h4 h5 h6** .
Note: The list element **DT** is treated as a special kind of heading for formatting purposes.

Style attributes

The `headfmt` property determines the overall appearance of the heading. Valid options are:

`display` The heading should be formatted as a block by itself. This is the default behaviour for most current browsers. If this form is used, all the attributes relevant to blocks are also relevant.

`runin` The heading should be flowed into the following text. If this form is used, all the attributes relevant to paragraphs are also relevant.

`margin` The heading should be placed in the left margin next to the following text.

```
.
.  Example of   If 'headfmt' is specified as 'margin',
.    marginal   browsers should produce a display that
.     heading   looks something like this.  Note that
.               the heading itself may wrap.  Different
.               alignment options are available, so the
.               heading may be right, left, or center
.               justified.  Designers who use this
.               option should be sure to leave plenty
.               of space in the left margin!
.
```

[[I've undoubtedly missed some. Any other options?]]
Note: If the next element after a `runin` heading is a paragraph, the heading should be interpreted as having appeared *inside* the paragraph element.
[[List which properties should be taken from the `heading` *and which from the* `para` *when formatting the paragraph as a whole. (space before, paragraph indent from heading, others from paragraph?)]]*
Note: If a `runin` heading is *not* followed by a paragraph or text, (e.g., a block) it should be formatted as a standalone paragraph.

4.10 Metainfo

Metainfo elements are those which contain auxilliary data about the document which should not be displayed in the main document. By definition, no formatting styles are applicable to metainfo elements.

HTML elements: **base isindex link meta nextid title** .

4.11 Divisions

HTML elements: **body form head html select**
[[Need separate section/division (Hierarchical vs. Containing Division as in InfoMaster)? Bert Bos suggested something similar; probably worth doing.]]

4.12 Floating elements

[["Floating" element is one that is not displayed at the point it appears in the document; No current HTML elements, but HTML+ ***MARGIN*** *and* ***FOOTNOTE*** *would work this way.]]*

4.13 Notes

There are still many missing items.

Multiple columns

Question: Should multiple columns be specifiable?

Multi-column formatting has been omitted, although many people seem to want this capability. *[[It doesn't seem like an appropriate formatting option for online browsing to me, since it will force the user to keep scrolling up and down if the display is not high enough to fit all the text.]]*

The number of columns, gutter width, vertical separators, et cetera, could be specified for block-level elements and/or the document element. HTML would really need a division or section element for this to be fully effective.

Scrolling browsers might be encouraged to ignore this directive.

Tables

[[Special concerns for tables: many proposed HTML table formats include a great deal of presentation information in attributes on the table elements themselves, like borders, column widths, etc. Tables by their nature are usually marked up with in a presentation-oriented fashion; need to reconcile that information with stylesheet information. Math will probably present similar incompatibilities. For now, much like HyTime's NOTLOC and SGML's NOTATION features, we punt and call math and tables "external" elements which must be formatted by their own rules...]]

Forms

[[This still requires more work. HTML forms are pretty badly broken. May have to punt here too...]]

5 Determining style applicability

The rules for resolving style element applicability are given below.

5.1 Lookup based on Generic Identifiers

Each **STYLE** attribute has an optional **GIS** attribute, which is a list of generic identifiers (element names) to which the style applies.

When the browser encounters a start-tag in the source document, it looks for that elements generic identifier in the **GIS** attribute of all active **STYLE** elements.

[[What if more than one style element matches? Need to specify priority scheme, or make it an error.]]

```
<stylesheet>
<!-- First-level headings (H1) centered, large, boldface
     Second-level slightly smaller, flush-right.
     Third- and fourth- smaller still,
     and H5 and H6 same as the text size.
-->
<style
    gis = "h1"
    fontfam = heading
    fontshape = bf
    fontsize = huge
    align = center
></style>
<style
    gis = "h2"
    fontfam = heading
    fontshape = bf
    fontsize = large
    align = right
></style>
<style
    gis = "h3 h4"
    fontfam = heading
    fontshape = bf
    fontsize = normalsize
    align = right
></style>
<style
    gis = "h5 h6"
    fontfam = heading
    fontshape = plain
    fontsize = normalsize
    align = right
></style>
</stylesheet>
```

5.2 Style inheritance

As is apparent from the previous example, writing a complete style sheet can quickly become unmanageable. A mechanism for *style inheritance* is also defined.

Each **STYLE** element has an optional **ID** attribute, which is an identifier unique to the style sheet. The optional **INHERIT** attribute names another **STYLE** element in the same stylesheet; any properties not specified on a `<STYLE>` element are taken from the element named by the **INHERIT** attribute.

An inherited style element may in turn inherit properties from another style, and so on. Inheritance chains must be acyclic.

```
<stylesheet>
<style id=headings
    fontfam = heading
    fontshape = bf
    align = right
></style>

<style
    gis="h1" inherit=headings
    fontsize=huge align=center>
</style>
<style
    gis="h2" inherit=headings
    fontsize=large>
</style>
<style
    gis = "h3 h4" inherit=headings
    fontsize=normalsize>
</style>
<style
    gis = "h5 h6" inherit=headings
    fontshape=plain fontsize=normalsize>
</style>

</stylesheet>
```

Note: Style attribute inheritance may be performed all at once, when the stylesheet is read. (But see p. 28 for further discussion.) That is, browsers may “precompute” all the style specifications inherited by each **STYLE** element up front, so the inheritance chain doesn’t need to be followed for each document element as it’s being formatted.
Question: Should the **INHERIT** attribute allow multiple inheritance?
Note: Multiple inheritance is not currently specified. This could be implemented, but the benefits do not seem worth the extra complexity to me.

5.3 Context-sensitive processing

So far, no good.

With the mechanisms defined so far, all the elements of a single type in a document share the same style specifications. It is desirable to allow context-sensitive style specifications so, for example, paragraphs inside block quotes can be formatted differently than paragraphs inside the main body.

Two alternate proposals are presented. The first is based on "style sets", in which **STYLE** elements are divided into groups; different groups may be active at different points in the document processing based on context. The style set mechanism is strongly influenced by SGML's "LINK" feature.

The second is based on "context patterns", in which the source document element hierarchy is matched against a context pattern qualifier attribute on **STYLE** elements. The context pattern mechanism is partially influenced by the X11 resource database.

Note: The two mechanisms are mutually exclusive alternatives. While it would be possible to implement both, it is probably better if one were chosen over the other.

Note: The context pattern mechanism seems more intuitive at first, but I feel that the style set mechanism would be much more powerful. Both seem about equally easy (or difficult) to implement.

Note: [9] uses something similar to style sets to specify context-sensitive style assignment: in that proposal, a style specification may contain other specifications which are applicable only when the outermost one is active. [10] uses something (vaguely) similar to the pattern mechanism, but turbocharged with an expression language.

Note: Both of these mechanisms would be much more useful if "section" or "division" elements were added to HTML.

5.3.1 Style sets

This mechanism groups style elements into disjoint *style sets*. Any number of style sets may be applicable at a given point in the document.

The initial style set consists of all the **STYLE** elements contained directly in the top-level **STYLESHEET** element. Other style sets are enclosed in **STYLESET** elements.

Each **STYLESET** element has a unique identifier, given by the **ID** attribute. **STYLE** elements have an optional **USESET** attribute, which is treated (almost) like other style properties. The **USESET** attribute names a style set which is to be activated for a source document element.

Browsers look for applicable **STYLE** elements in the currently active style set. If no applicable style element is found, the previous style set is searched (i.e., the one that was active in the parent element), and so on up the document hierarchy until the initial style set is reached.

[[This could be explained a lot better. The concept is really much simpler than it sounds!]]

For example, the following style sheet fragment specifies that **EM** and **STRONG** emphasis are indicated by font-changes in most places, but that color should be used instead inside headings (presumably because headings are already set in boldface).

```
<stylesheet>

<colors>
<color id = red rgb = "#FF0000">
<color id = blue rgb = "#0000FF">
</colors>
```

```
<!-- base style for headings: -->
<style id=headings
    USESET = INHEADING
    fontfam = heading
    fontshape = bf
    align = right
></style>

<style gis="h1" inherit=headings fontsize=huge>
</style>
<style gis="h2" inherit=headings fontsize=large>
</style>
<style gis = "h3 h4" inherit=headings fontsize=normalsize>
</style>
<style gis = "h5 h6" inherit=headings fontsize=normalsize>
</style>

<!-- Default styles for EM and STRONG -->
<style gis="em" fontshape=it></style>
<style gis="strong" fontshape = bf></style>

<!-- when the INHEADING set is active,
     different styles for EM and STRONG are used instead
-->
<styleset id=inheading>
    <style gis="em" fgcolor=red></style>
    <style gis="strong" fgcolor=blue></style>
</styleset>

</stylesheet>
```

Note: [9] allows context-sensitive style specifications by allowing style specifications to be nested. This proposal uses a level of indirection – the **USESET** attribute and separately defined sets – so that style sets may be easily reused. The resolution mechanisms seem to be similar in both cases, i.e., if an applicable style rule cannot be found in the currently active set, the previous set should be checked.
[[Add: ***POSTSET*** *attribute, analagous to* `#POSTLINK`*. This will allow specifications like "all paragraphs* ***after*** *an* `<H3>` *...". This is something the pattern mechanism can't easily handle.]]*
[[How about `<?USESET ...>` *processing instruction, analagous to* `<!USELINK ...>` *declaration? Nah, probably not...]]*

5.3.2 Context pattern matching

A perhaps more intuitive way of specifying context-sensitive style processing is to check the current element hierarchy against a *pattern*. The pattern is specified in the **CONTEXT** qualifier attribute.

[[Need to check out FOSI e-i-cs, which seem similar.]]

The **CONTEXT** attribute is a list of generic identifiers which are matched (left to right) against the document hierarchy (outermost element to innermost).

*[[Should **GIS** and **CONTEXT** attributes be mutually exclusive, or could **GIS** be used (if present) as the last component of the context pattern?]]*

In a **CONTEXT** pattern, `?` (question mark) may be used instead of a GI to match any single element, and `*` (asterisk) matches a series of zero or more elements.

Context patterns implicitly begin with a `*`. To "anchor" a pattern to a specific depth, the pattern may begin with `HTML`.

[[Need to fully describe the pattern-matching algorithm, since subtle differences in implementations can lead to unexpected results (as I found out when we upgraded our system from X11R4 to X11R5!). Suggest using the X11R5 rules since the R4 specification is vague.]]

Here is how the earlier example could be accomplished using patterns:

```
<stylesheet>

<colors>
<color id = red rgb = "#FF0000">
<color id = blue rgb = "#0000FF">
</colors>

<!-- styles for EM and STRONG inside headings:
     Use colors instead of font changes for emphasis
-->

<style context = "em" fontshape=it></style>
<style context = "h1 * em" fgcolor=red></style>
<style context = "h2 * em" fgcolor=red></style>
<style context = "h3 * em" fgcolor=red></style>
<style context = "h4 * em" fgcolor=red></style>
<style context = "h5 * em" fgcolor=red></style>
<style context = "h6 * em" fgcolor=red></style>

<!-- this is getting old fast...
     need to introduce more powerful patterns like:
-->
<style context = "strong" fontshape = bf></style>
<style
    context = "(h1|h2|h3|h4|h5|h6) * strong"
    fgcolor=blue>
</style>

</stylesheet>
```

Note: A **CONTEXT** pattern may *end* with a `*` or `?`, as in `<pattern context = "h1 *" fontshape = bf fontfam=heading>`. This may be used to augment the normal property inheritance mechanism.

Question: What should happen if more than one **style** element matches? Use only the most-specific match? Or use all matching specifications, with specifications in the most-specific match overriding those in less-specific matches? The latter would be more useful, but a great deal more complex to implement.
[[Need to explain "most-specific match" wrt. matching algorithm.]]

5.4 Specifying styles in the document

So far, still no good.

The most important reason for stylesheets is so authors can use presentation to convey information. HTML does not have a rich enough tag set to identify all possible semantic information, and it never can.
Note: But see [19] for one way arbitrary semantic information could be encoded without modifying the base tag set. If a similar scheme is deployed for HTML, it will be supported by stylesheets.

This proposal suggests a change to HTML itself that allows authors to embed style information in the document. The change is simple: Add an optional **STYLE** attribute to every HTML element. This attribute is the ID of a **STYLE** element in the document's style sheet.
Note: The HTML **STYLE** attribute must have a declared value of `NAME`, not `IDREF`, since stylesheets are not part of the HTML document.
[[Example ATTLIST declaration here...]]
Note: Documents that use this mechanism will be tied to a specific stylesheet or class of stylesheets.

For example, this document contains many phrases that refer to SGML objects: elements, attributes, and so forth. In HTML, they are all marked up as **CODE** elements, but it would be useful if a different **STYLE** element could be specified for different types of objects; for example, elements in the HTML DTD could be displayed in red and elements in the stylesheet DTD could be in blue:

```
<stylesheet>
<style id=html-elem fontshape=tt fgcolor=red>
    <note>Style used for HTML elements</note>
</style>
<style id=ss-elem fontshape=tt fgcolor=blue>
    <note>Style used for STYLESHEET elements</note>
</style>
<note>attributes are displayed just like elements</note>
<style id=html-att inherit = html-elem>
</style>
<style id=ss-att inherit=ss-elem>
</style>
<style gis = "code" fontshape=tt>
</style>
</stylesheet>
```

and in the document:

```
In HTML, they are all marked up
as <code style=html-elem>CODE</code> elements,
but it would be useful
```

```
if a different <code style=ss-elem>STYLE</code>
element were used for...
```

Note: A reader with a monochrome monitor (or a better sense of typographic design!) should be able to hack the stylesheet to use different presentational effects to make this distinction. If a reader wanted to distinguish between attributes and elements, that would also be possible just by modifying the stylesheet, as long as the information is encoded in the document. This is, I feel, one of the biggest advantages of logical markup over explicit formatting directives.
Note: Suggestion to browser implementors: Selecting an anchor (**A** element) with a **rel=stylesheet** attritute specification could be interpreted as a request to redisplay the current document with the referenced stylesheet. This would allow authors to prepare multiple "views" of a document, distinguished by stylesheets alone.

HTML equivalents

Some HTML elements already have attributes which indicate specific formatting properties, such as **COMPACT** on **DL** (and **ALIGN** on various elements in HTML 3). Others imply formatting changes by themselves, such as **B** and **I**, and Netscape's **FONT** and **CENTER**. p. 39 gives a list of all such elements and attributes, and the equivalent style properties.

When an explicit HTML formatting directive is encountered, the following resolution rules are suggested:

- If the element's start-tag also has a **STYLE** attribute, all "native" HTML style specifications should be ignored.
- Otherwise, the HTML constructs should be mapped to their equivalent stylesheet specification, and these override specifications determined by the normal stylesheet mechanism.

Note: The rationale is that authors may wish to supply some specifications for browsers which are not stylesheet-capable, while giving a more complete specification to those that are.
Question: If a **STYLE** attribute is present along with HTML formatting directives, should the HTML directives be ignored, or should they be interpreted as an element-specific override?
Question: If HTML formatting directives are present with no **STYLE** attribute, should the regular style processing take place or should it be bypasssed for the element in question?

5.5 Notes on style qualifiers

Inheritance

There are *two* separate inheritance hierarchies for style properties: in the document hierarchy, elements inherit style properties from their parent element; and in the style sheet, **STYLE** elements may inherit specifications from other **STYLE** elements through the **INHERIT** attribute.

There is an important distinction between style *properties* and style *specifications*. A *property* is a parameter value maintained internally by the browser to make layout decisions, whereas a *specification* is an attribute on a **style** element which *modifies* a property.

This distinction is very important when dealing with relative dimension specifications. For example, take the following style sheet fragment:

```
<stylesheet>
<style id=s1
    fontfam=alternate
    lmargin="+3em"
>
</style>
<style id=s2 gis="blockquote"
    inherit=s1
    lmargin="+5em"
></style>
<style id=s3 gis="p" fontshape=plain>
</style>
</stylesheet>
```

Suppose the current left margin is `1em`, and a **BLOCKQUOTE** element is encountered. The applicable style element is `s2`, which inherits specifications from `s1`. The specification in `s2` for `leftmargin` *overrides* the specification in `s1`, so the left margin is indented by `5em`, not `8em`.

Conversely, suppose that a **P** element occurs inside the **BLOCKQUOTE**. The applicable style element is `s3`, which does not have a specification for `leftmargin`, so the pararagraph inherits the left margin property from the **BLOCKQUOTE** element: it does *not* inherit the `leftmargin="+5em"` *specification*, so the left margin for the paragraph will be `6em` (unchanged), not `11em`.

Also note that a **STYLE** element may specify properties for an element that are not directly applicable to the element itself. For example, a style for **BLOCKQUOTE** elements may specify a `bullet` property, which is not used by the **BLOCKQUOTE** itself. The specification must be applied anyway, so that it may be inherited by **LI** elements inside **UL**s inside the **BLOCKQUOTE**.

6 Linking Stylesheets to the Document

Browsers should retrieve the stylesheet to use with an HTML document from one of the following places, in order of precedence:

- A `<LINK>` element in the **HEAD** with a **REL** attribute of `STYLESHEET` (case-insensitive). The **HREF** attribute is the URL of the stylesheet to use.

  ```
  <!doctype HTML PUBLIC "-//IETF//DTD HTML//EN//2.0">
  <head>
  <link rel=stylesheet
        href="http://www.foo.com/styles/default.ss">
  <title>Some Document</title>
  </head>
  ```

- If the document was retrieved via HTTP, from a `Link:` or `WWW-Link:` HTTP response header with a `rel=stylesheet` qualifier.

  ```
  WWW-Link: rel=stylesheet;
    href="http://www.foo.com/styles/default"
  ```

Partial URLs should be interpreted relative to the *document*'s base URL.

- A default style sheet specified by the user (optional)
- A default style sheet for the browser

When style sheets are transmitted over HTTP, they should have a media type (MIME Content-Type) of `text/sgml`.
[[Look into this: is there a `dtd` *parameter?]]*
Note: The other style sheet proposals all use the same mechanism for associating stylesheets with documents. It will probably be necessary to specify *what kind* of stylesheet is being referred to. If so, stylesheets defined by this proposal may be designated with a link relation of `alfonso` instead of `stylesheet`. I expect no name-clashes here.

6.1 Multiple style sheets

This proposal assumes that no more than one style sheet is used for a document at any time. The potential interactions between two or more independently written style specifications are highly unpredictable.
Note: [10] attempts to deal with this issue; should look there for ideas.

The most oft-stated reason for wanting to combine style sheets, however, is so that users may selectively override portions of external stylesheets. See the next section.

6.2 User preferences

Browsers are encouraged to provide users with the ability to configure the default style sheet. It is also desirable if users may selectively override parts of an external stylesheet without discarding the entire specification.

To accomplish this, style sheets may specify a *weight* for each attribute. The weight is an integer from 1 to 3 for external stylesheets, and from 0 to 4 for user's configurations. A different weight may be specified for each style attribute. Stylesheet authors should assign weights to attributes based on how important that particular attribute is:

1 - Incidental "This is just how I like to see it; go ahead and change it if you like."

2 - Important style attribute is used to convey additional nonessential information. ("Corporate identity" is in this category.)

3 - Critical This attribute is used to convey essential semantic information. (E.g., all element names in blue, all attribute names in red.)

User preferences are specified in a stylesheet just like the document style sheet. Browsers process both stylesheets in parallel, using the user's specification for an attribute if it has a higher weight, and the browser's specification otherwise.
Note: If the preference sheet has assigned a higher weight than the stylesheet for an attribute, then any specifications for that attribute in the stylesheet should be ignored even if there is none in the preferences sheet. E.g., user assigns `<weight prop=color value=4>`, and stylesheet assigns a color for an element, and the applicable style in the preferences file lists no color, then the color should not be changed.

[[Give an example, and rewrite the last paragraph. It's incomprehensible.]]
Note: Browsers may wish to restrict the user preference sheet to a simpler subset of the full stylesheet language; e.g., no context-sensitive processing, no ID lookups.
Note: The valid range for weight values is intentionally small; this is to enhance predictability.

A SGML definitions

Listing 1: stylesheet.dtd

```
<!-- DTD for style specifications
     10 Jul 1994
     "-//ART.COM//DTD Style Sheet//EN//DRAFT"
     $Id: stylesheet.dtd,v 1.4 1994/11/16 02:51:05 joe Exp $
-->

<!--
    ============================================================
    Parameters for attribute declared values
    ============================================================
-->

<!entity % dimen "NUTOKEN"      -- dimension -- >
<!entity % hdimen "%dimen;"     -- horizontal dimension -->
<!entity % vdimen "%dimen;"     -- vertical dimension -->
<!entity % rdimen "CDATA"       -- relative dimension -->

<!entity % units -- informational only; %units; not used in DTD --
        "... ednote: list units here "
>

<!--
    dimen, hdimen, and vdimen are absolute dimensions.
    Must be an integer followed by one of the "%units;" tokens.

    rdimen attributes are relative if they begin with + or -,
    absolute otherwise.

    To specify negative absolute rdimen, prefix with =

    Convention:
        vertical dimensions are "xxxskip",
        horizontal dimensions are "xxxspace".
-->

<!entity % color  "IDREF"       -- ID of COLOR element in stylesheet -->
<!entity % sepspec  "IDREF"     -- ID of SEPSPEC element in stylesheet -->
```

```
<!-- Fonts: -->

<!entity % fontfam
   "normal
  | heading
  | alternate
  | fixed
">

<!entity % fontsize
   -- or: 0-7? --
   "tiny
  | small
  | normalsize
  | large
  | big
  | huge
">

<!entity % fontshape
    "plain | bf | it | bi | tt | sc"
>

<!entity % halign       "left | center | right" >
<!entity % valign       "top | middle | bottom" >

<!-- special effects -->

<!entity % line         "noline | underline | overline | strikethrough" >
<!entity % foldcase     "nofoldcase | toupper | tolower" >

<!entity % box          "box | nobox" >

<!-- Attribute declared value "notations" -->

<!entity % RGB  "CDATA"
        -- RGB values using X hex notation, #FF00FF -->

<!entity % URL "CDATA"
        -- Uniform Resource Locator -->

<!-- Format attributes: -->

<!entity % headfmt      "(display | runin | margin)" >
<!entity % numfmt       "(arabic | lcroman | ucroman | lcalpha | ucalpha)" >
```

```
<!--
    ============================================================
    Attribute definition lists
    ============================================================
-->

<!ENTITY % a.blkstyl    -- Block-level style attributes --
"
    fontfam     (%fontfam;)     #implied
    lmargin     %rdimen;        #implied        -- left margin --
    rmargin     %rdimen;        #implied        -- right margin --
    preskip     %vdimen;        #implied
    postskip    %vdimen;        #implied
    presep      %sepspec;       #implied        -- id of SEPSPEC element --
    postsep     %sepspec;       #implied        -- override {pre,post}skip --

    box         (%box;)         nobox
    boxcolor    %color;         #implied
    bgcolor     %color;         #implied        -- background color --
">
<!-- %%% Need: background tile image -->

<!ENTITY % a.chrstyl    -- Character level style attributes --
    "
    fontsize    (%fontsize;)    #implied
    fontshape   (%fontshape;)   #implied

    fgcolor     %color;         #implied

    -- special effects --
    line        (%line;)        #implied
    linecolor   %color;         #implied
    foldcase    (%foldcase;)    #implied
">

<!ENTITY % a.hdgstyl    -- Heading style attributes --
"
    headfmt     %headfmt;       #IMPLIED
    -- used inside lists --
    icon        IDREF           #IMPLIED
    numfmt      %numfmt;        #IMPLIED
">

<!ENTITY % a.parstyl    -- Paragraph style attributes --
"
```

```
    align       (%halign;)      #implied
    xleading    %vdimen;        #implied        -- extra leading --
    parskip     %vdimen;        #implied        -- space between paras --
    parindent   %hdimen;        #implied        -- initial indent --

    obeylines   (obeylines|wraplines)   #implied
    obeyspaces  (obeyspaces|squeezespaces)      #implied
">


<!--
    ============================================================
    Content models
    ============================================================
-->

<!entity % decls        "colors?" >
<!entity % specs        "style | styleset | sepspec | image">

<!element stylesheet    - - ((%decls;), (%specs;)*) +(note) >

<!element note          - -     (#PCDATA) >

<!element colors        - -     (color+) >
<!element color         - O     EMPTY >
<!attlist color
    id          ID      #REQUIRED
    rgb         %RGB;   #REQUIRED
    -- others? --
>

<!element image         - O     EMPTY >
<!attlist image
    id          ID      #IMPLIED
    refid       IDREF   #IMPLIED  -- ID of another image --
    url         %URL;   #IMPLIED
    reftype     NAMES   #FIXED "refid IMAGE"
>

<!element hrule         - O     EMPTY >
<!attlist hrule
    width       %hdimen;        #IMPLIED
    thick       %dimen;         #IMPLIED
    align       (%halign;)      #IMPLIED
>
```

```
<!element vspace        - O     EMPTY >
<!attlist vspace
    vskip       %vdimen;        #REQUIRED
>

<!element sepspec       - -     (hrule | vspace | image)*>
<!attlist sepspec
    id          ID      #REQUIRED
>

<!-- Style elements: -->

<!element styleset      - -     (style+) >
<!attlist styleset
    id          ID      #REQUIRED
    inherit     IDREF   #IMPLIED
    reftype     NAMES   #FIXED "inherit styleset"
>

<!element style         - O     ANY
        -- ??? what should the content model be? --
>
<!attlist style
    id          ID      #IMPLIED
    inherit     IDREF   #IMPLIED        -- STYLE --
    useset      IDREF   #IMPLIED        -- STYLESET --
    postset     IDREF   #IMPLIED        -- STYLESET --

    -- Qualifiers: --
    gis         NAMES   #IMPLIED
    context     CDATA   #IMPLIED

    -- Specifiers: --
    %a.blkstyl;
    %a.chrstyl;
    %a.parstyl;
    %a.hdgstyl;

    reftype     NAMES   #FIXED     "inherit     style
                                    icon        image
                                    useset      styleset
                                    postset     styleset
                                    presep      sepspec
                                    postsep     sepspec
                                    bgcolor     color
                                    fgcolor     color
```

```
                                    boxcolor    color
                                    linecolor   color"
>
```

B Sample stylesheet for HTML

Heree is a more comprehensive stylesheet exampe, corresponding to the recommended renderings in the HTML specification.

Listing 2: sample.ss

```
<!doctype stylesheet SYSTEM>

<!--
    Sample stylesheet for "typical" HTML rendering
    $Id: sample.ss,v 1.2 1994/11/19 02:15:50 joe Exp $
-->
<stylesheet>

<!--
    Overall style for document:
    Specify small left and right marggins.
-->

<style gis = "body"
    lmargin = "5pcd"
    rmargin = "5pcd"
>

<!-- Styles for headings: -->

<note>
These are taken from the June 1993 HTML draft,
not what any browser actually does.
Need to update it forthe 2.0 version.
</note>

<style id = headings
    fontfam     = heading
    fontshape   = bf
    headfmt     = display
    preskip     = 1nlh
    postskip    = 1nlh
```

```
    lmargin     = 0pcd
    align       = left
></style>
<style gis = "h1"
    inherit     = headings
    fontsize    = huge
    preskip     = 2nlh
    align       = center
></style>
<style gis = "h2"
    inherit     = headings
    fontsize    = big
></style>
<style gis = "h3"
    inherit     = headings
    fontsize    = large
    fontshape   = it
></style>
<style gis = "h4"
    inherit     = headings
    fontsize    = normalsize
    fontshape   = bf
    lmargin     = 2pcd
></style>
<style gis = "h5"
    inherit     = headings
    fontsize    = normalsize
    fontshape   = it
    lmargin     = 2pcd
></style>
<style gis = "h6"
    inherit     = headings
    fontsize    = normalsize
    fontshape   = it
    lmargin     = 5pcd
></style>

<note>
Need to fill the rest in...
</note>

<!-- blocks -->
<style gis = "address"
></style>

<style gis = "blockquote"
```

```
></style>

<!-- containers -->
<style gis = "body"
></style>
<style gis = "form"
></style>
<style gis = "head"
></style>
<style gis = "html"
></style>


<!-- block displays -->
<style gis = "select"
></style>
<style gis = "option"
></style>
<style gis = "pre"
></style>
<style gis = "textarea"
></style>

<!-- inline displays -->

<style gis = "input"
></style>
<style gis = "image"
></style>

<!-- Lists -->
<style gis = "dt"
></style>
<style gis = "dd"
></style>
<style gis = "li"
></style>
<style gis = "menu"
></style>
<style gis = "dir"
></style>
<style gis = "dl"
></style>
<style gis = "ol"
></style>
<style gis = "ul"
```

```
></style>

<!-- Paragrapphs -->
<style gis = "p"
></style>

<!-- Phrases -->
<style gis = "a"
></style>
<style gis = "b"
></style>
<style gis = "cite"
></style>
<style gis = "code"
></style>
<style gis = "em"
></style>
<style gis = "i"
></style>
<style gis = "kbd"
></style>
<style gis = "samp"
></style>
<style gis = "strong"
></style>
<style gis = "tt"
></style>
<style gis = "var"
></style>

<!-- Sepearators -->
<style gis = "br"
></style>
<style gis = "hr"
></style>

</stylesheet>
```

C HTML equivalents of style properties

Not yet written. Needs more research.

D WWW-Arch architectural form definition

Listing 3: wwwarch.dtd

```
<!-- wwwarch.dtd
    Author:     Joe English
    Created:    7 Nov 1994
    $Date: 1994/11/19 02:15:50 $
-->

<!entity % phrase "phrase"
        -- all phrase-level elements -->
<!entity % para "para"
        -- all paragraph-level elements -->
<!entity % block "block"
        -- all block-level elements -->
<!entity % heading "heading"
        -- all heading-like elements -->

<!entity % metainfo "metainfo"
        -- all metainformation elements -->

<!-- entity declarations for content models: -->

<!entity % body.content "(%heading; | %block; | %para; )*" >
<!entity % block.content "(%block; | %para)*" >
<!entity % para.content "(%phrase; | #PCDATA)*" >
<!entity % phrase.content "(%phrase; | #PCDATA)*" >

<!element wwwarch - - (head,body)>
<!element head - - (%metainfo;)* >

<!element body - - (%body.content;)*>

<!element (%block;)     - - %block.content;>
<!element (%para;)      - - %para.content;>
<!element (%phrase;)    - - %phrase.content;>
<!element (%heading;)   - - %phrase.content;>

<!element list          - - (item+) -- single-part list -->
<!element dlist         - - (dlentry+)  -- two-part list -->
<!element dlentry       O O (heading*, item*)>
<!element item          - - (%text.content; | (%para;)+)>

<!-- this will probably be more useful when it's finished...
-->
```

References

[1] A "clearinghouse" document at CERN with references all the major extant stylesheet proposals.
`<URL:http://info.cern.ch/hypertext/WWW/Style/>`

[2] Gavin Nicol `<gtn@ebt.com>` is collecting a set of requirements for stylesheets.

[3] C. M. Sperberg-McQueen `<cmsmcq@uic.edu>` has put together a list of simple formatting primitives.
`<URL:http://tigger.cc.uic.edu/~cmsmcq/style-primitives.html>`

[4] The HTTP protocol documents at CERN.
`<URL:http://info.cern.ch/hypertext/WWW/Protocols/HTTP/>`

[5] The HTML 2.0 draft standard.
`<URL:http://www.hal.com/~connolly/html-spec/index.html>`

[6] The HTML SGML declaration.
`<URL:html.decl>`

[7] The www-talk mailing list hypermail archives.
`<URL:http://gummo.stanford.edu/html/hypermail/>`

[8] Rob Raisch's stylesheet proposal, posted to www-talk.
`<URL:.www-talk-1993q2.messages/443.html>`

[9] Pei Wei's stylesheet RFC, used in Viola.
`<URL:http://pebble.berkeley.ora.com/vdoc/style/stylesheetRFC.txt>`

[10] Cascading Stylesheets, Håkon W Lie.
`<URL:http://info.cern.ch/hypertext/WWW/People/howcome/p/cascade.html>`

[11] SoftQuad's stylesheet mechanism; soon to be released.
[[Will probably make this proposal totally pointless.]]

`<URL:http://www.sq.com/>`

[12] Steve Pepper `<pepper@falch.no>` is conducting an investigation into the possibility of using a small subset of DSSSL (the Document Style Semantics and Specification Language) as the basis for a standard style language for HTML and other SGML applications.
[[Will definitely make this proposal pointless if SoftQuad's stylesheets don't.]]

`<URL:http://www.falch.no/~pepper/DSSSL-Lite/>`

[13] A proposal by Jon Bosak `<Jon_Bosak@Novell.COM>` for the development and deployment of HDL, a new document format based on SDL.
`<URL:ftp://ftp.ora.com/pub/davenport/HDL/hdl.proposal.html>`

[14] Questions and answers about HDL.
`<URL:ftp://ftp.ora.com/pub/davenport/HDL/hdl.q-and-a.html>`

[15] Information about SDL, the Semantic Delivery Language.
`<URL:ftp://ftp.ora.com/pub/davenport/SDL>`

[16] Netscape's enhancements to HTML.
`<URL:http://home.mcom.com/home/services_docs/html-extensions.html>`

[17] The only information I have on DSSSL at present. Erik Naggum has promised a review...
`<URL:http://www.art.com/~joe/dsssl.txt>`

[18] Proposal for a DIVISION element in HTML. Not yet written; see www-talk archives for the initial proposal.

[19] Wayne Wohler's article in comp.text.sgml describing user-defined logical markup in IBMIDDoc. (Sorry, no URL; www-html archives seem to be down).