Cascading style sheets for XML

Sebastian Rahtz
July 2001

Summary

The purpose of this session is to give a flavour of one of the family of Web standards, CSS. Others in the group include:

- XML: Extensible Markup Language
- XSL: Extensible Style Language
- DOM: Domain Object Model
- XHTML: HTML in XML
- MathML: Mathematical Markup Language
- SVG: Scaleable Vector Graphics
- ...etc ...

CSS: why stylesheets?

- To separate form from content
- To allow alternate renderers for different classes of readers
- To achieve effects not possible in HTML
- To provide presentation for XML
- To specify consistency across a range of documents

Note that CSS only supports *embellishment* of HTML or XML, and not serious *transformation*. Full details of CSS can be found at `http://www.w3.org/Style/CSS/`.

CSS support in Web browsers

Very variable!

- There is no 100% reliable CSS renderer
- Opera (Windows, Linux and some others) is probably the best
- Mozilla / Netscape 6 are good
- Internet Explorer 5 is reasonable
- Netscape 4 has some serious flaws and does not work with XML

We will mainly cover here features of CSS version 1; version 2 is scantily supported; version 3 is now being decided on.

A sample of CSS

```
body {  background: white;
        margin: 3em;
        }

div2 > head {color: blue;
        font-style: italic;
        border-style: dashed;
        border-width: medium;
        text-align: center;        }

xref  {color: red;}

p {display: block; }
```

The elements of CSS

CSS declarations cover five basic types of presentational information, called *properties*:

- Foreground and background colors and background images.
- Font properties.
- Text properties (word spacing, letter spacing, etc.).
- Boxes (margins and borders around block elements, floating elements, etc.).
- Classifications (control over list styles and the formatting of elements–whether they should be presented inline or displayed as a block, for example).

How do you include your style sheet?

Style information can be included in a document in several ways:

- With a style sheet processing-instruction at the top of an XML document:

```
<?xml-stylesheet type="text/css"
                 href="http://..."?>
```

- With a style sheet <LINK> in the <HEAD> of an XHTML document:

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

- In the <style> element in the <head> of XHTML documents.

Simple style rule

A simple style rule has the following form:

```
element-name { property: list }
```

where the element name is anything that appears in your XML file (<p>, , <emph> etc.), and the property list is the associated style commands.

```
item { color:red;
     background-color: yellow;
   }
```

Complex style rule

If you list a series of element names separated by spaces, the rule applies only to instances of the last element that have all of the previous ones before them. For example,

```
div head emph { color: green }
```

assigns the property to <emph> only if it occurs inside of an <head> inside of a .
If you list element names separated by > characters, the elements must appear *exactly* in that relationship, with no intervening elements

```
body > div > head { font-size: 120%}
```

If you list a series of names separated by commas, the rule applies to *all* of them.

```
eg, corr, sic { color: blue;}
```

Pseudo-elements

There is a small, but significant, class of *pseudo-elements*, which are specified as an element name followed by a colon and a *state name*. The possible state names are:

link When the element is a hypertext link

visited When the element is a *visited* hypertext link

active When the element is a link, and being activated

hover When the element is a link, and the mouse is hovering over it

before When the processor is about to render an element

after When the processor has just rendered an element

XML example of pseudo-elements

```
gi:before {
  content: "<";
}

gi:after {
  content: ">";
}
```

This puts < before the content of the <gi> element, and > afterwards.

XML example of pseudo-elements (2)

You can also show the value of attributes, using the `content` property and the *attr* function, as in

```
note[n]:before {
  content: attr(n);
  vertical-align: super;
}
```

which says that if a `<note>` element has an ‘n’ attribute, we want to put the value of that attribute, superscripted, before the element content.

Subclassing by attribute value

You can narrow down the effect of a rule by specifying attribute values in square brackets after the element name:

```
hi[rend='bold']   { font-weight: bold;}
hi[rend='italic'] { font-style: italic ;}
```

Subclassing by ID

Another form of subclassing is by ID. This is the mechanism that allows you to avoid embedding specific style information in your document even for a unique element. If you have a specific element that must be uniquely treated, you can give it an ID:

```
<p id="special-case">
```

and then assign style information to that element with a #:

```
#special-case { properties }
```

Note that IDs are *required* to be unique within a document. This requirement may not be enforced by current browsers, but it is a requirement for a truly conforming document.

Foreground and background colors

There are two properties for specifying colors: `color` and `background`. The `color` property controls the foreground color of an element. Usually this is the color of the text of an element. Colors may be identified either by name or by RGB value.

The `background` property controls the background color or texture of an element. When an image is specified for use as a texture, its position, scrolling aspect, and repeatability can be controlled.

The normal set of colors

Name	Percent	Integer	Hexidecimal	
aqua	rgb(0%,100%,100%)	rgb(0,255,255)	#00FFFF	#0FF
black	rgb(0%,0%,0%)	rgb(0,0,0)	#000000	#000
blue	rgb(0%,0%,100%)	rgb(0,0,255)	#0000FF	#00F
fuschia	rgb(100%,0%,100%)	rgb(255,0,255)	#FF00FF	#F0F
gray	rgb(50%,50%,50%)	rgb(128,128,128)	#808080	#888
green	rgb(0%,50%,0%)	rgb(0,128,0)	#008000	#080
lime	rgb(0%,100%,0%)	rgb(0,255,0)	#00FF00	#0F0
maroon	rgb(50%,0%,0%)	rgb(128,0,0)	#800000	#800
navy	rgb(0%,0%,50%)	rgb(0,0,128)	#000080	#008
olive	rgb(50%,50%,0%)	rgb(128,128,0)	#808000	#880
purple	rgb(50%,0%,50%)	rgb(128,0,128)	#800080	#808
red	rgb(100%,0%,0%)	rgb(255,0,0)	#FF0000	#F00
silver	rgb(75%,75%,75%)	rgb(192,192,192)	#C0C0C0	#BBB
teal	rgb(0%,50%,50%)	rgb(0,128,128)	#008080	#088
white	rgb(100%,100%,100%)	rgb(255,255,255)	#FFFFFF	#FFF
yellow	rgb(100%,100%,0%)	rgb(255,255,0)	#FFFF00	#FF0

Font properties

There are five properties that control which fonts are used:

`font-family` Identifies the font family, or typeface, to use. A series of names may be requested; the first available font will be used. There are five classes of "generic" fonts that may be specified as a last resort, `serif` for serifed faces like Times Roman; `sans-serif` for san-serif faces like Helvetica; `monospace` for fixed-width fonts like Courier; `cursive` for swash faces like Zapf Chancery, and `fantasy` for other hard-to-classify faces like Grunge or Western.

Font properties (2)

`font-style` Identifies the style of the face, `normal`, `italic`, or `oblique`.

`font-variant` Identifies another variation on the face, either `normal` or `small-caps` in CSS1.

`font-size` The size of the face. Font size may be specified in absolute units (eg `10pt`) or relative to the 'current' size (eg `120%`)

`font-weight` The weight or boldness of the font, specified with either a keyword (`bold` or `bolder`, for example) or as a member of the ordered series 100, 200, 300, . . . , 900, where higher numbers are correspondingly darker.

Text properties

Several text properties are available:

`word-spacing` Modifies the default inter-word spacing.

`letter-spacing` Modifies the default inter-letter spacing.

`text-decoration` Choices are `underline`, `overline`, `line-through`, `blink` or `none`>.

`text-transform` Shifts text to uppercase or lowercase, with values `capitalize`, `uppercase`, `lowercase` or `none`.

Text properties (2)

`text-align` Specifies alingment as `left`, `right`, `center` or `justify`.

`text-indent` Determines the amount of indentation on the first line of a block of text.

`line-height` Specifies the distance between the baselines of consecutive lines of text.

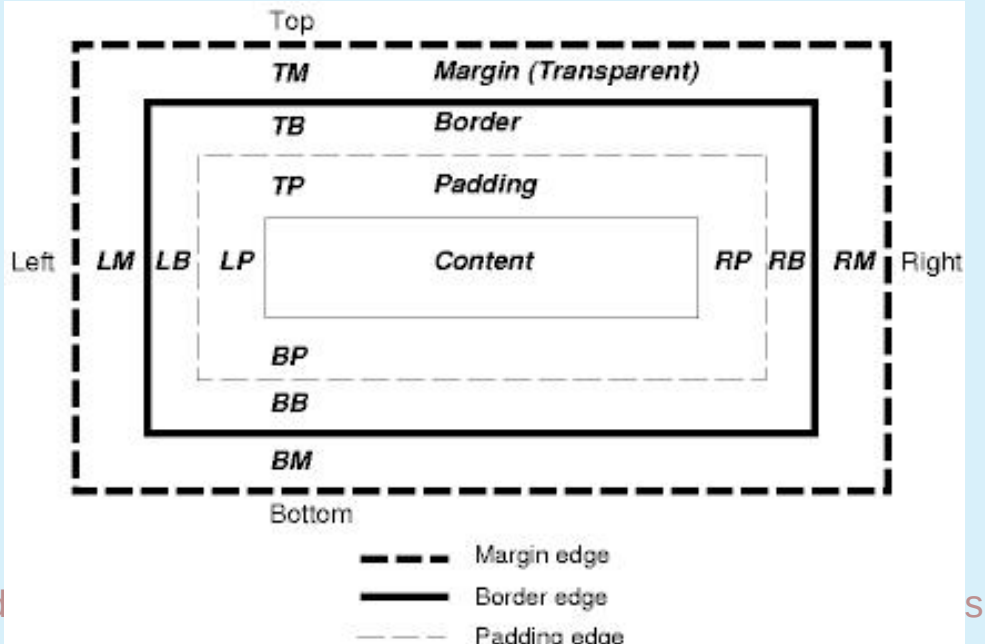
Text properties (3)

`vertical-align` Adjusts the vertical alignment of an element. Possible values are

- **baseline** align the baseline of the element (or the bottom, if the element doesn't have a baseline) with the baseline of the parent
- **middle** align the vertical midpoint of the element (typically an image) with the baseline plus half the x-height of the parent
- **sub** subscript the element
- **super** superscript the element
- **text-top** align the top of the element with the top of the parent element's font
- **text-bottom** align the bottom of the element with the bottom of the parent element's font

Boxes

Each element has a margin, inside the margin is an optional border, inside the border is optional padding, and inside the padding is the actual formatted content of the element. The box properties allow you to determine how the content of the element is formatted to fit in the space provided.



Boxes (2)

`margin` (`margin-bottom`, `margin-top`, `margin-left`, `margin-right`) Determine the size of the top, bottom, left, and right margins. Setting `margin` adjusts all of the margins simultaneously.

`padding` (`padding-bottom`, `padding-top`, `padding-left`, `padding-right`) Adjusts the amount of padding on the top, bottom, left, and right sides of the element. Setting `padding` adjusts all of them simultaneously.

Boxes (3)

`border` (`border-top`, `border-bottom`, `border-left`, `border-right`) Selects the nature of the border on the top, bottom, left, and right sides of the element. Setting `border` adjusts all of borders simultaneously. You can specify the `-width`, `-color`, and `-style` of the border. ie

```
border-top-style: solid;
border-bottom-width: 12pt;
border-left-color: black;
```

Boxes (4)

`width`, `height` Identifies the width and height of the rectangle that contains the formatted content. Images should be scaled to the specified size, if necessary.

`float` Identifies an element that should float to the left or right of a flow of text, allowing the text to flow around it.

`clear` Specifies where floating elements may occur with respect to the element. For example, specifying `clear: left` indicates that there may be no floating elements to the left of the element; this will force the element to start below an image floating on the left side of the display, for example.

Classifications

There are three classification properties, `display`, `list-style`, and `white-space`:

`display` Allows you to specify what category of object an element belongs to: ie a block element, like a heading or paragraph (*block*); an inline element, like emphasis or anchors (*inline*); or a list-item block element, like TEI `<item>` (*list-item*). An additional category is *none*, which indicates that the content of the element should not be displayed at all.

Classifications (2)

`list-style` Influences the selection of numbers or bullets for lists. In addition to selecting one of the built-in enumeration or bullet styles, you can specify an image for use as the bullet character. You can also influence the position of the list mark with respect to the flow of the text. An `outside` list mark occurs to the left of the entire list item, whereas text wraps under an `inside` mark.

`white-space` Identifies how the line breaking of an element is to be accomplished. Possible values are `normal`; `pre`, where all white space is significant; and `nowrap` where white space serves primarily as a delimiter, but no wrapping is done.

An example XML TEI stylesheet

```
body
{
  font-family: "Times New Roman";
  font-size: 12pt;
  margin-top: 5px;
  margin-left: 5px;
}

emph {
  display: inline;
  font-style: italic;
}

hi {
  font-weight: bold;
}

p {display: block}
```

An example XML TEI stylesheet (part 2)

```
item {
  display: list-item;
  margin-left: 12pt;
}

list[type="bulleted"] {
  list-style-type: disc;
  list-style-position: outside;
}

list[type="ordered"] {
  list-style-type: decimal;
  list-style-position: outside;
}
```