CSS

Cascading Style Sheets

Style Sheets and HTML/XML, Generally

- Style sheets for HTML were initially advocated to
  - provide more power and flexibility
  - clean up HTML atrocities:
    - » the proliferation of tags that encode presentation (e.g., <blink>, <i>)
    - » misuse of tags (e.g., using blank images, paragraphs for format control)
  - Idea was to enhance by adding a simple style language.
- On a separate XML thread: There is no flow object interpretation of elements, so some way of associating layout properties with them would be useful.
  - Idea was to develop XML-FO, a standalone flow object language.

What Happened

- Adding a simple style language (and reforming HTML) turned HTML into a low-level flow-object language!
- Think of HTML elements as objects that direct layout, with pre-defined default properties.
  - The style language lets one control details, like font size, color, etc.
- The style language became general enough that
  - one could use it to lay out XML directly
  - one could translate XML to HTML+style via XSL
- The style language became even more powerful, introducing page models, non-visual formatting, and a host of other things.
  - So the need for something like XML-FO was greatly diminished.
- Interesting, many of the ways one uses HTML for layout didn't go away.
  - E.g., table and list layout semantics are still essential.

How to Introduce Style into an XML/HTML document

- Separate style sheets
  - Supplied by user, author, as system resources
- Reference
  - Add some way for a document to refer to an external style sheet.
  - E.g., we saw the use of processing instructions with XML to invoke an XSL style sheet; this is also proposed for CSS.
- In-line
  - New elements, attributes defined for "style hooks".
  - Using attributes, or architectural forms, is probably more in line with the SGML philosophy.
    - » In general, namespaces would help with attribute name conflicts, but I haven't seen any proposal to this effect.

Implementing Style Within HTML

- Use a **LINK** element with **rel** attribute for author-suggested specification of style sheet resources.
- (Partial) **LINK** definition (4.1 DTD):

```
<!ELEMENT LINK - O EMPTY>
<!ATTLIST LINK
    id      ID          #IMPLIED        -- SGML ID attribute --
    href    CDATA       #IMPLIED        -- URL for linked resource --
    rel     CDATA       #IMPLIED        -- forward link types --
    rev     CDATA       #IMPLIED        -- reverse link types --
    title   CDATA       #IMPLIED        -- advisory title string --
    type    CDATA       #IMPLIED        -- advisory Internet media type --
    media CDATA#IMPLIED                 -- for rendering on these media --
    >
```

Example of LINK Element for Style Sheet Purposes

```
<LINK TITLE="Stanford" rel="stylesheet"
    href="http://www.stanford.edu/stanford.dsssl"
    type="application/dsssl">
<LINK title="Cal" rel="stylesheet"
    href="http://www.berkeley.edu/cal.css" type="text/css">
<h1>Welcome to my amazing home page!</h1>
If your browser supports style sheets, try this page in Cal and
    Stanford styles!!

...
```

- Multiple **LINK** elements mean that the client should provide a choice; recommendation is that **title** be used to identify choices, say, in a menu.
  - (Anyone support this?)
- **type** can be used to disregard unsupported notation types without retrieval.

STYLE Element

- Used to embed style sheets within a document.
- **STYLE** definition:

```
<!ELEMENT      style      - -        CDATA>
<!ATTLIST      style
    …
    type       CDATA  #REQUIRED     -- content type of language--
    media      CDATA  #IMPLIED      -- designed for use with --
    title      CDATA  #IMPLIED      -- advisory title --
    >
```

- Any number allowed in **HEAD** elements.

STYLE Element Example

```
<HEAD>
<TITLE>Some Document</TITLE>
<STYLE TYPE="text/style-language-x">
  ...here appears a style sheet in style language x...
</STYLE>
</HEAD>
```

- Explicit style elements can co-exist as preferred alternatives with **LINK**'ed style sheets.

Backward Compatibility Problem

- How to prevent content of **STYLE** elements from appearing when old browser doesn't support them
- Solution: Style language should allow and ignore SGML comment characters. E.g., instead of previous example:

```
<STYLE TYPE="text/style-language-x">
 <!--
  ...here appears a style sheet in style language x...
 -->
</STYLE>
```

Attributes in Support of Style Sheets

- The following attributes are allowed in most elements, and are useful re style:
  - **style** - Provides element-specific rendering information
  - **id** – Allows reference to an entity by name
  - **class** - Allows a set of elements to be grouped together for reference
- The "element identifiers" attributes (i.e., **id** and **class**) are ostensibly general, but
  - **class** is used almost exclusively for style.
  - It is recommended that **id** be used rarely for same.

(There used to be a style entity, but I believe it has vanished:

```
<!ENTITY  %        style
  "id        ID        #IMPLIED        -- unique id --
  class      CDATA     #IMPLIED        -- list of class values --
  style      CDATA     #IMPLIED        -- style language statements --">
)
```

Attribute Examples

- **style**:

  <P style="... *style sheet right here !...*">The style sheet would have to be in the style language previously declared to be the default; the default style language can be described using an HTML **META** element.

- **class**:

  <P class="special">The style language might say that all things of **class** "**special**" should be a certain color or size, etc.

Tags to Help with Style

- There are generic structure elements, **DIV** and **SPAN**.
  - **DIV** demarcates some *sequence of elements* (i.e., a "block").
  - **SPAN** demarcates some portion of *text* (i.e., "inline" content).
- More formally:

```
<!ELEMENT       span      - -       (%inline;)*>
<!ATTLIST       span
          %attrs;   --  all the common attributes  -->
<!ELEMENT       div       - -       (%flow;)*>
<!ATTLIST       div
          % attrs;  -- all the common attributes -->
```

- These are often used just to have something to attach style specs to.

SPAN and DIV Examples

<DIV class="section">

<P>The spec states that DIV, together with the "class" attribute, be used to denote structural elements, like "abstract" or "chapter", for which HTML doesn't provide elements.

<P>DIV allows headers, lists, paragraphs, and other DIV elements.

<P><SPAN class="init">Maybe we want</SPAN> the first three words here to be set in a special way. SPAN elements can appear wherever you expect text.

<P>Presumably, somewhere, a style sheet would have to state how the "init" and "section" classes are to appear. Or, instead, we could have used the "style" attribute and put a style statement in directly in with the tags.

</DIV>

SGML/XML Generally

- External style sheets via processing instructions truest to SGML philosophy.
- Making **STYLE** and **LINK** architectural forms might be more in the SGML spirit.
- Reserving some attributes just for style:
  - While putting style in a given style language directly into document is suspect, it is probably better than misusing tags.
  - **CLASS** is extremely useful in practice.
  - Again, namespaces would help.
- Use of **ID** in conformance with general SGML practices.
- Introducing elements just for style, e.g., **DIV** or **SPAN**, is bogus. Need another way to refer to document fragments (XPointer, XLink, etc.)

Cascading Style Sheets

- Provides a "simple" style language.
  - But, naturally, keeps getting more complex.
- Allows multiple style specifications for a given document.
- A set of rules ("cascading order") determines precedence and resolves conflicts.
- Status:
  - CSS supported as of MS IE 3.0, Netscape 4.0.
  - CSS2 now widely supported (at least partially).
    - » Unlike CSS1, CSS2 can be used by XML (or any other SGML-like language) as well as HTML.
    - » Implementations vary in interpretation; backwards compatibility an issue.
  - CSS3 in the works.
    - » A "color module" is in working draft stage.

CSS2: Concepts

- A style sheet comprises a set of (one or more) *statements*.
- A statement is either an *at-rule* or a *rule set* (or *rule*).
  - There are a handful of these: **@import**, **@page**, **@media**, **@font-face**.
  - Seem to handle special cases, each with its own syntax.
    - » E.g., **@import "foo.css"**
- A rule comprises a list of *selectors* and a *declaration* (or, *declaration block*). The declaration provides elements of style; selectors associate the style with components of the document.
  - A rule can have an optional *weight*.
- A declaration is a list of *property/value* pairs.
- There is a well-defined processing model that specifies how these are used.

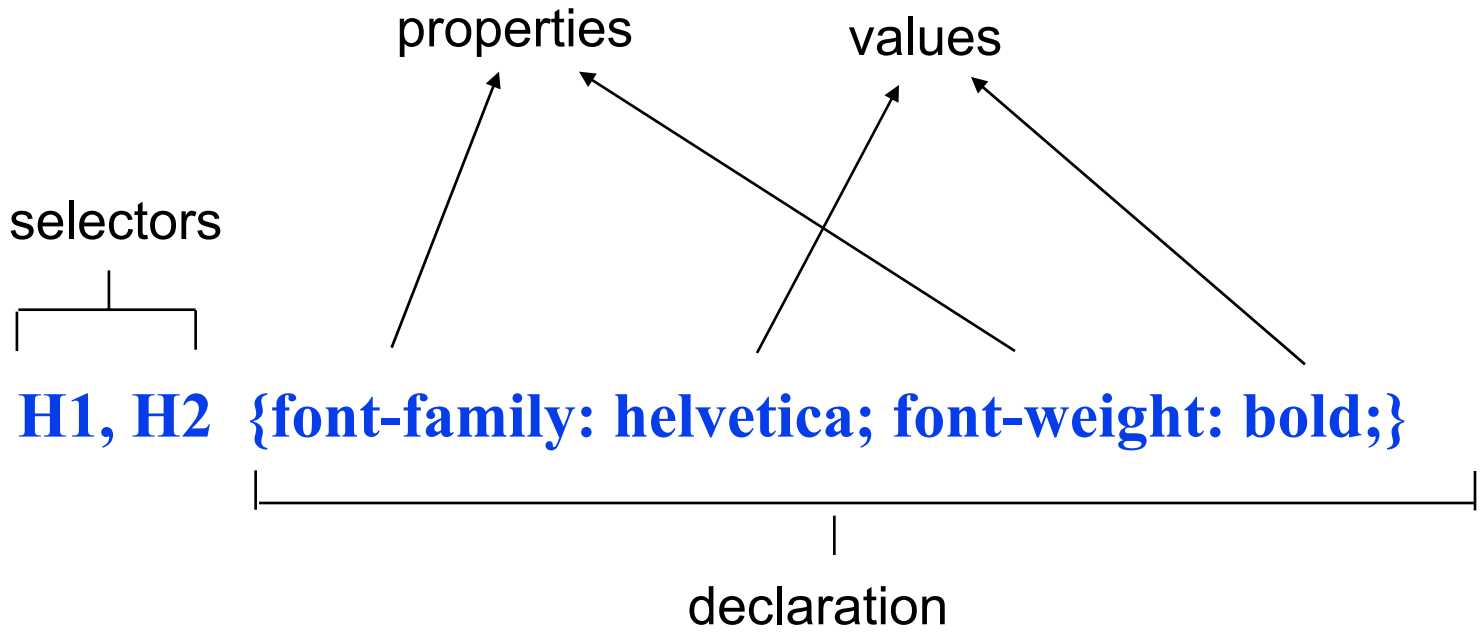
The CSS2 Processing Model

- Parse the source document and create a *document tree*.
- Identify the *target media type*.
- Retrieve all style sheets associated with the document that are specified for the target media type.
- Annotate the document tree
  - by assigning a value to every style property applicable to the target media type.
  - Values may depend on target media type, e.g., speech vs. display vs. printed pages
- From the annotated document tree, generate a *formatting structure*.
  - Exactly what this is is implementation-dependent.
  - May contain content not in the tree, omit content that is.
- Render the designated presentation.

The Canvas

- The canvas describes "the space where the formatting structure is rendered."
- Canvas has dimensions depending on media type.
  - E.g., 2 dimensions for visual, 4 (!) for sound (three-dimensional physical space—"sound surrounds"--and a temporal space)
- The canvas is
  - infinite for each dimension of the space
  - rendering generally occurs within a finite region, established by the user agent

Simple CSS Rule Example



CSS Specifies

- A language for stating selectors
- A set of properties, each with an associated range of values
  - CSS2 defines about 90 properties.
- A formatting model
- Where style sheets can appear, and how to interpret them
  - E.g., defaults, interactions of multiple sheets, priorities

Selectors

- Basic selector components include
  - all tags
  - IDs (i.e., values of **ID** attributes)
  - Classes (i.e., values of **CLASS** attributes)
  - "Pseudo-classes" and "pseudo-elements"
- Combinations
  - classes, pseudo-classes and pseudo-elements within a tag (These and basic selectors are "simple selectors".)
  - Contextual selectors - simple selectors occurring within simple selectors

Simple Selector Examples

```
H1 { color: blue }          /* A tag. */
P { font-size: 10pt }       /* Another tag. */
    /* Note that C-like comments are allowed */
.footnote { font-size: 80% }        /*Applies to all elements whose
    CLASS attribute has value "footnote", e.g, <DIV
    CLASS="footnote"> elements. */
P.critical { color: red }   /* Apply the style only to P elements with
    CLASS "critical", e.g, <P CLASS=critical> elements */
#xyzzy { font: small sans-serif }   /* Apply the style to the element
    with the unique ID attribute value "xyzzy", perhaps the element
    with start tag <P ID="xyzzy">. */
```

- Heavy reliance on CLASS, and use of ID altogether, is discouraged (but I find the former extremely useful).

Pseudo-class and -elements

- We can cover some common cases in which there aren't available structure elements by pretending that there are.
- *Anchor pseudo-classes* reflect link status.
- *Typographical pseudo-elements* pretend there is an element that might only be generated at layout time.

Pseudo-Examples

- Pseudo-classes

```
A:link { color: red }          /* unvisited link */
A:visited { color: blue }      /* visited link */
A:active { color: green }      /* link currently being pressed */
```

- Pseudo-elements

```
P:first-line { font-variant: small-caps; }   /* The first-line
    pseudo-element will make the first line as formatted of a P element
    appear as if it were inside a P:first-line element, with the resulting
    effect on style. */
P:first-letter { font-size: 200% }   /* first-letter is as if it
    surrounds the "first letter" of the element. */
```

- Combinations

```
A.footnote:visited { color: yellow } /* class and pseudo-class */
P.urgent:first-line { color: red } /* class and pseudo-element */
```

Contextual Selector Examples

H1 EM { font-variant: small-caps }/* EM elements within H1 elements should be in SMALL CAPS */

.footnote EM { font-variant: small-caps } /* EM elements within elements of CLASS "footnote". */

UL { font-size: 90% }

UL UL { font-size: 80% }

UL OL.urgent UL { color: pink } /* "Within" is construed transitively, so these create ambiguity. */

DIV.chapter P:first-letter { color: blue } /* A pseudo-element can go at the end of a contextual selector */

Combining Style Information

- Ambiguity arises because
  - A style may not be explicitly stated.
  - Several, possibly conflicting, probably partial, style sheets might be specified by a document.
  - The reader might also supply style sheets.
    - » How this is done is client-specific.
  - Several contextual rules might apply.
- The "cascade" is a procedure that resolves ambiguities.

The Cascade

- For each element
  - Find all matching declarations.
    - If none, inherit from containing element.
      - If nothing to inherit, use initial value.
  - Prefer "important" over unmarked declarations.
  - Prefer author's over reader's (over browser's default).
  - Prefer more specific over more general selectors.
  - Prefer later over earlier specifications.

Cascade (continued)

- Inheritance is by document instance structure, starting at the top-level element.
  - Whether a given property is inherited is specific to that property. E.g., (text) **color** is inherited; **background** isn't.
  - Result of applying a percentage value is inherited, not the percentage.
- "important"" can be specified as

  H1 { color: red ! important }
- Authors are given precedence over readers
  - but the client is supposed to allow the reader to turn off any style sheet, including the author's.

Specificity

- One selector is more specific than another if
  - it has more **id** attributes,
  - else if it has more **class** attributes,
  - else if it has more tag names.
- Examples:

  UL UL is more specific than **UL**.

  UL.urgent is more specific than **UL UL UL UL** ....

  UL UL.urgent is more specific than **UL.urgent**.

Implications of the Cascade

- Most of the time these rules lead to reasonably intuitive behavior:
  - Specifying just one feature of an element will most likely change just that feature, preserving those of the surrounding or of less targeted specification. E.g.:

    .emph { font-weight: bold; }

    will probably cause **emph** class elements to have the same other font characteristics as the text around them.
  - Referring to someone else's style sheet and then listing desired exceptions will resulting in the proper customization.

Curious Consequences

- Suppose we have the following rules:

```
OL LI { list-style: decimal }/* i.e., 1 2 3 ... */
UL LI { list-style: circle }      /* i.e., ○ */
UL.urgent LI { list-style: disk }   /* i.e.,  ●  */
```

  and write:

```
<UL class=urgent>
    <LI> It is vital that you take the following measures. in order:
        <OL><LI> Press the button labeled "open flush value".
                <LI>Press the button labeled "flush radioactive waste".
```

  thinking this will produce:

  - It is vital that you take the following measures. in order:
    1. Press the button labeled "open flush value".
    2. Press the button labeled "flush radioactive waste".

- However, the last rule always wins, so we end up using disks in the ordered list.

Curious Consequences (con't)

- Should be unnecessary to special-case ordered lists within unorder lists, as there isn't really any exception here.
- Recommended fix is to attach rules to list *parents*, not to the list elements:

  UL.urgent { list-style: circle }

  OL { list-style: decimal }

- Then:
  - first rule applies to the **UL** with class **urgent**
  - only the second rule applies to the **OL**
  - no rule applies to any embedded **LI**
    - » which will inherit from the surrounding list.

Curious Consequences (con't)

- Might seem as if we could declare properties of, say, bold text within a footnote by the following:

  B.footnote { font-variant: small-caps }

  and then say:

  <DIV class=footnote>

  <P>See also Schlemiel '92, ... in <B>CVETCH: Journal of Cached Fetching</B>

- Doesn't work implicitly because attributes aren't inherited. I.e., would have to say

  <P>...<B CLASS=footnote>CVETCH: ...</B>

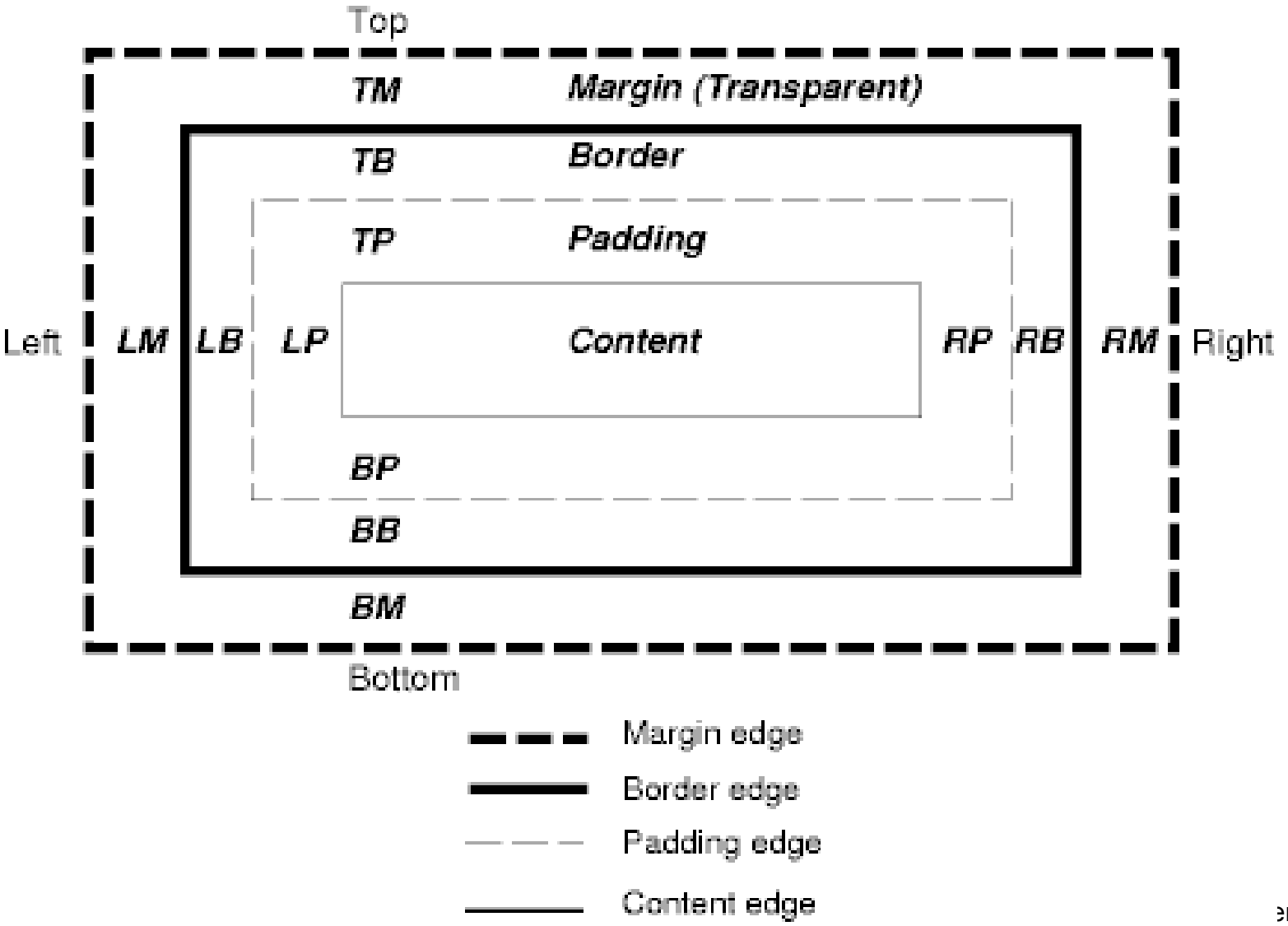
  or use declaration

  .footnote B

Support for Visual Formatting in CSS2

- A box model:
  - Rectangular boxes are generated for elements in the document tree.
  - They are laid out according to a *visual formatting model.*
- Boxes have contents, padding, borders, and margins, each with its own edges and right, left, top and bottom segments.

The four areas of the generic CSS box: content, padding, border, and margin.



Boxes (con't)

- Document elements produce 0 or more boxes, each of which will typically be positioned wrt a *containing block*
  - which is generally established by some other box (actually, by a box's padding edge.)
- Boxes may also have a "z-index", which established the layer precedence of overlapping boxes,

Boxes and Formatting

- Boxes are basically *in-line* or *block*.
  - In "normal flow", block boxes get laid out one after another vertically,
  - In-line boxes get laid out one after another horizontally.
- E.g., a paragraph is a stack of line boxes, each comprising a series of in-line boxes.
- There are other kinds of boxes:
  - *Compact* and *run-in* boxes behave like in-line or block boxes depending on the context.
  - Tables also appear to be their own box type, which can be like in-line or like block.
- Pages are also their own type of box.
- Note: It is important to be able to declare box type in XML, where elements have no default box-type.

Positioning Schemes

- There are really three basic schemes:
  - Normal flow
  - Floats
  - Absolute positioning
- However:
  - There is a variant of normal flow, called *relative positioning*.
  - There is a variant of absolute positioning, called *fixed positioning*.
- This is confused by the properties set up:
  - The **position** property can have values
    static | **relative** | **absolute** | **fixed**
  - There is a separate **float** property.
- These have some complex, but well-specified, interactions.

Floats

- Box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible.
- Content may flow along the side of a float.

Absolute Positioning

- Box assigned *an explicit offset with respect to the containing block.*
- Box is removed from the normal flow.
  - and hence has no impact on how sibling boxes are set.
  - May obscure other boxes, depending on the *stack levels* of the overlapping boxes.

Fixed Positioning

- A special case is *fixed positioning*.
  - Containing box is established by the viewport (a window or other viewing area on the screen through which users view a document).
  - For continuous media, fixed boxes do not move when the document is scrolled.
  - For paged media, boxes with fixed positions are repeated on every page.
  - Fixed positioning a set of boxes is an alternative to frames in many cases.

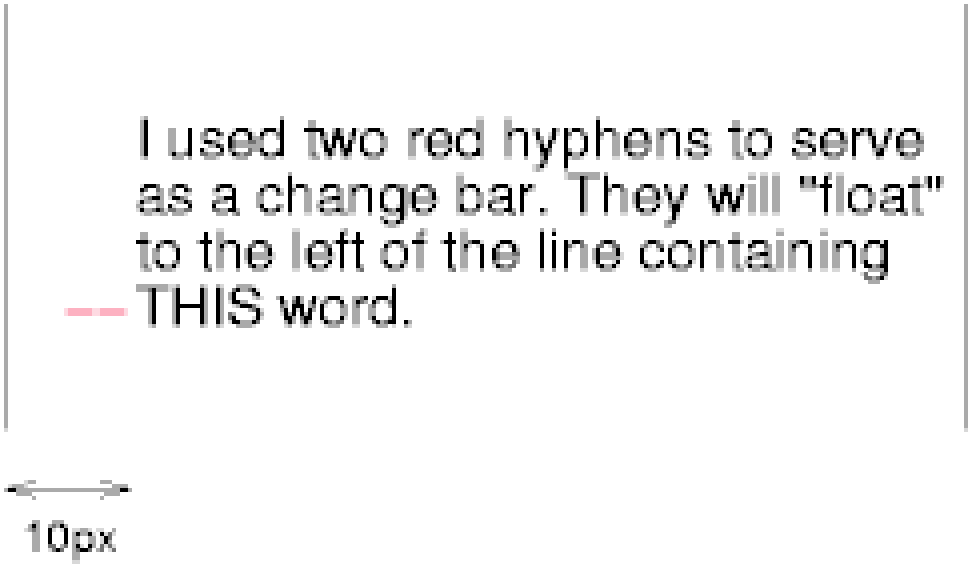
Relative Positioning

- This is just a variant of normal flow, in which a box is laid out, and then shifted from where normal layout would place it by some amount.
  - E.g., super/subscripts setting
- No effect on following boxes.
- Since a relative positioned box establishes a new containing box for its children, it is sometimes used just for this purpose, with its contents not shifted.

Absolute within Relative Example

```
<P style="position: relative; left: 10px; margin-right: 10px;">
  I used two red hyphens to serve as a change bar. They
  will "float" to the left of the line containing THIS
  <SPAN style="position: absolute; top: auto; left: -1em; color:
    red;">--</SPAN>
  word.</P>
```

Produces:



Use Positioning, etc., to Achieve Copyediting Effect

- Consider the following style elements

```
<STYLE TYPE="text/css">
  .strut { font-size: 24pt; position: relative; color: green;  }
  .anno { font-size: 10pt; position: absolute; top: 0px }
</STYLE>
```

- in the following example

```
<P style="position: relative">This is some plain HTML text.  We
   would like to place an external annotation on it, which is still
   hard to do, but we can at least make a span
<span class=strut>
 <span class=anno><span style="font-weight: bold; font-size:
   10pt; color: red">
Replace:</span>  here is candidate replacement
   text</span>
</span><u>that is marked for replacement</u> as this example
   illustrates.
```

Example (con't)

- Will produce the following

This is some plain HTML text. We would like to place an external annotation on it, which is still hard to do, but we can at least make a span that is marked for replacement as this example illustrates.

Replace: here is candidate replacement text

See Examples

- Of boxes and positioning
- Of advanced use of absolute positioning
- Note: Browsers will disagree on how to render these.
  - Example more or less optimized for IE.

Page Boxes

- A page box is just a big box.
  - So, it has margins, etc., along with some page-specific properties.
- When set up, a page box will serve as the containing box for content occurring within page breaks.
  - E.g.,

    @page { size 8.5in 11in; margin: 2cm }

    sets up a page as you would expect.
- CSS2 doesn't say where to break.
  - It does say where breaks are allowed.
  - Users can stipulating where breaks go.
  - CSS2 has a recommended good breaking policy.

Usage in HTML

- Accommodated via general HTML style mechanisms mentioned above.
- Imported style sheets allowed inside elements. E.g.,

```
<STYLE TYPE="text/css">
<!--
@import url(http://www.theirsite.com/their-style.css);
H1 { color: blue }
-->
</STYLE>
```

results in a single style sheet in which the specification for **H1** is overridden locally.

Usage in XML

- Proposal is to link a style sheet to an XML document is using a processing instruction.
- E.g.:

```
<?XML:stylesheet type="text/css" href="some-
  style.css"?>
<our-tl-element>
  <sub-element>
    ...
```

- Simple illustration from spec. (works in IE ≥5).

How Applicable to General XML DTDs?

- Weak on
  - languages with difficult orthography.
- Not as powerful as DSSSL/XSL, but much simpler.
  - Don't really have to be a programmer to write a style sheet.
  - No ability to transform document.