STRATEGIC WEBSITE TECHNOLOGIES

LECTURE 4

(DYNAMIC HTML)

Introduction

"Dynamic HTML" is a term used by some vendors to describe the combination of HTML, style sheets and scripts that allows documents to be animated.

A style sheet is made up of style rules that tell a browser how to present a document.

By attaching style sheets to structured documents on the Web (e.g. HTML), authors and readers can influence the presentation of documents without sacrificing device-independence or adding new HTML tags.

Two main types of style sheets:

1.Cascading Style Sheets (CSS) is a style sheet mechanism that has been specifically developed to meet the needs of Web designers and users.

2.eXtensible Style Language (XSL): XSL builds on CSS and is primarily targeted for highly structured XML data.

There is also Document Style Semantics and Specification Language (DSSSL)

Cascading Style Sheets

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents.

There are various ways of linking these style rules to your HTML documents, but the simplest method for starting out is to use HTML's STYLE element.

This element is placed in the document HEAD, and it contains the style rules for the page.

Each rule is made up of a selector --usually an HTML element such as BODY, P, or EM-- and the style to be applied to the selector.

There are numerous properties that may be defined for an element. Each property takes a value, which together with the property describes how the selector should be presented.

Style rules are formed as follows:

selector { property: value }

Multiple style declarations for a single selector may be separated by a semicolon:

selector { property1: value1; property2: value2 }

As an example, the following code segment defines the color and font-size properties for H1 and H2 elements:

```
<HEAD>
<TITLE>CSS Example</TITLE>
   <STYLE TYPE="text/css">
       H1 { font-size: x-large; color: red }
       H2 { font-size: large; color: blue }
   </STYLE>
</HEAD>
```

The above style sheet tells the browser to show level-one headings in an extra-large, red font, and to show level-two headings in a large, blue font.

Some properties to play with

Background:

The *background-color* and *background-image* properties set the background color and image of an element, respectively. E.g.

```
BODY { background-color: white }
H1   { background-color: yellow }
P    { background-image: url(http://www.mysite.com/space.jpg) }
```

Text:

The *letter-spacing* property defines an additional amount of space between characters.

```
H1     { letter-spacing: 0.3em }
```

The *text-decoration* property allows text to be decorated through one of five properties: underline, overline, line-through, blink, or the default, none.

```
P {text-decoration: underline || overline }
```

The *text-indent* property defines the amount of indentation that the first line of the element should receive.

```
P { text-indent: 15mm }
```

The *line-height* property could be used to double space text:

```
P { line-height: 200% }
```

Box:

The *margin* property sets the margins of an element by specifying between one and four values, where each value is a length, a percentage, or auto.

```
BODY{margin: 1cm 2cm 3cm 4cm}/*top margin 1cm,
                               right margin 2cm,
                               bottom margin 3cm,
                               left margin 4cm */
```

The *border* property is a shorthand for setting the width, style, and color of an element's border.

```
H2{ border: groove 3em }
H3{ border: solid blue }
H4{ border: thin dotted #800080 }
H5{ border: thick double red }
```

Font:

The *font-style* property defines that the font be displayed in one of three ways: normal, italic or oblique (slanted).

```
H1 { font-style: oblique }
P  { font-style: normal }
```

The *font-weight* property is used to specify the weight of the font. Possible Values: normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900

```
H1 { font-weight: 800 }
P  { font-weight: normal }
```

The *font-size* property is used to modify the size of the displayed font.

xx-small | x-small | small | medium | large | x-large | xx-large | larger | smaller | <length> | <percentage>

```
H1     { font-size: large }
P      { font-size: 12pt }
LI     { font-size: 90% }
STRONG { font-size: larger }
```

The *font-family* property is used to select the font-family.

```
P { font-family: “Courier" }
```

The CLASS Attribute

The **CLASS** attribute is used to specify the style class to which the element belongs. For example, the style sheet may have created the **punk** and **warning** classes:

```
.punk {color: lime; background: red}
P.warning {font-weight:bolder; color: red; background: white}
```

These classes could be referenced in HTML with the **CLASS** attribute:

```
<H1 CLASS=punk>Proprietary Extensions</H1>
<P CLASS=warning>This is a warning paragraph…</P>
```

Linking to an External Style Sheet

The **<LINK>** tag is placed in the document **HEAD**. The optional **TYPE** attribute is used to specify a media type -- **text/css** for a Cascading Style Sheet -- allowing browsers to ignore style sheet types that they do not support.

External style sheets should *not* contain any HTML tags like **<HEAD>** or **<STYLE>**. The style sheet should consist merely of style rules or statements. A file consisting solely of

P { margin: 2em }

could be used as an external style sheet.

<LINK REL=StyleSheet HREF="style.css" TYPE="text/css" MEDIA=screen>
<LINK REL=StyleSheet HREF="color-8b.css" TYPE="text/css" TITLE="8-bit Color Style" MEDIA="screen, print">

Importing a Style Sheet

A style sheet may be *imported* with CSS's **@import** statement. This statement may be used in a CSS file or inside the **STYLE** element:

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

All **@import** statements must occur at the start of the style sheet.

Any rules specified in the style sheet itself override conflicting rules in the imported style sheets.

The order in which the style sheets are imported is important in determining how they cascade.

Imported style sheets are useful for purposes of modularity.

For example, a site may separate different style sheets by the selectors used. There may be a *simple.css* style sheet that gives rules for common elements such as **BODY**, **P**, **H1**, and **H2**.

In addition, there may be an *extra.css* style sheet that gives rules for less common elements such as **CODE**, **BLOCKQUOTE**, and **DFN**.

A *tables.css* style sheet may be used to define rules for table elements.

These three style sheets could be included in HTML documents, as needed, with the **@import** statement.

Inlining Style

Style may be inlined using the **STYLE** attribute. The **STYLE** attribute may be applied to any **BODY** element

```
<P STYLE="color: red; font-family: 'New Century
Schoolbook', serif"> This paragraph is styled in red
with the New Century Schoolbook font, if available.</P>
```

Inlining style loses many of the advantages of style sheets by mixing content with presentation.

By using *position*, *top*, *left*, and *visibility* properties it is possible to determine the exact location of an element on the page.

```
<H1 style="position:relative; top:5%; left:5%;“>
  A header that knows its location! </H1>
```

The SPAN Element

The **SPAN** element was introduced into HTML to allow authors to give style that could not be attached to a structural HTML element.

```
<STYLE TYPE="text/css" MEDIA="screen, print,
projection">  .firstwords { font-variant: small-caps }
</STYLE>
…
<P><SPAN CLASS=firstwords>The first few words</SPAN> of
a paragraph could be in small-caps.
Style may also be inlined, such as to change the style
of a word like <SPAN STYLE="font-family: Arial">  Arial
</SPAN>.</P>
```

The DIV Element

The **DIV** element is similar to the **SPAN** element in function, with the main difference being that **DIV** (short for "division") is a block-level element. **DIV** may contain paragraphs, headings, tables, and even other divisions. This makes **DIV** ideal for marking different classes of containers, such as a chapter, abstract, or note.

```
<DIV CLASS=note>
<H1>Divisions</H1>
<P>The DIV element is defined in HTML 3.2, but only the
ALIGN attribute is permitted in HTML 3.2. HTML 4.0 adds
the CLASS, STYLE, and ID attributes, among others.</P>
<P>Since DIV may contain other block-level containers,
it is useful for marking large sections of a document,
such as this note.</P>
<P>The closing tag is required.</P>
</DIV>
```

A selection of books on CSS:

- Keith Schengili-Roberts. Core CSS (Prentice Hall, 2000, ISBN 0-13-083456-4)
- Ian Graham. The XHTML 1.0 Language and Design Sourcebook (John Wiley and Sons, 2000, ISBN 0-471-37485-7)
- Eric Meyer: Cascading Style Sheets: The Definitive Guide (O'Reilly & Associates, 2000, ISBN 1-56592-622-6
- Håkon Wium Lie, Bert Bos: Cascading Style Sheets: Designing for the Web (2nd edition, Addison-Wesley 1999, ISBN 0-201-59625-3)
- Erik Wilde: Wilde's WWW, technical foundations of the World Wide Web. (Springer 1998, ISBN:3-540-64285-4)

Document Object Model (DOM)

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

All headers, paragraphs, tables, images, forms etc form the elements of a document. The properties of these elements can be reached and changed using *JavaScript* !

For instance, we have been using *JavaScript* to change the values of form elements.

Although there are some standards, there is no complete compatability between major browser vendors (ie Microsoft and Netscape) as far as the DOM is concerned.

In our course, we shall be focusing on the Microsoft DOM.

DHTML

In order to create dynamic HTML pages we use JavaScript to change the properties of elements in a document.

In this way, it is possible to create buttons, menus, animations and truly interactive pages.

To reach each element we can use the ID attribute in HTML tags and document.all object (for MS) in JavaScript.

ID Attribute

By using the **ID** attribute it is possible to give an ID for each tag. Then the tag can be reached from a scripting language and modified. In IE 4.0 the element can be reached using the document.all object (collection).

Examples:

```
<H1 ID="Header"
onMouseOver=“document.all.Header.style.color=‘red';"
onMouseOut="this.style.color='black';"> Turn me red! </H1>
```

```
<H2 ID=“hdr2" onClick=“this.innerText=‘Thanks!';"> Click on
me! </H2>
```

```
<H3 ID=“hdr3" onClick=“this.className=‘pastoral';"> Click on
me! To change my style class!</H3>
```

Using Layer Technology

- By using layers it is possible to create dynamic effects on web pages.
- Layers can be thought as transparencies put on top of each other. This brings a third dimension in design.
- By using layers it is possible to determine exact locations (in terms of pixels) of text and images on the browser.
- By hiding some of the layers or showing a part of them (clipping) and using a script language, it is possible to produce awesome effects.
- In order to use such technology efficiently, some basic animation knowledge is required.
- There are differences between the tags to be used for MS Internet Explorer and Netscape Navigator.

Creating Layers

Layers can be created by using the style sheets. The *z-index* property can be used to determine the order of the layers.

```
<div id="id1" title="Layer1" style="position:absolute;
left:10; top:10; z-index:2;">
<H1> Hi there </H1>
</div>
<div id="id2" title="Layer2" style="position:absolute;
left:40; top:10; z-index:1;">
<img src="ring.gif" width="50" height="50">
</div>
```

Animation using Scripts

It is possible to reach the style properties of an element from a Scripting language (e.g. JavaScript). Therefore it is possible to have dynamic effects (and animation).

```
function Animate(posx,posy)
{
  document.all.id1.style.top=posy;
  document.all.id1.style.left=posx;
  document.all.id2.style.left=400-posx;
  px=posx+2;
  py=posy;
  if(px > 400) {px=0}
  window.setTimeout(“Animate(px,py);",10);
}
```

There are differences between Netscape and MS IE.

The browser type is determined first for the script to run correctly.

Browser Sniffing

It is possible to learn the name and version of the browser used by the user in JavaScript by the help of the navigator object. This object comes with several properties including appName, appVersion, appCodeName and userAgent.

appName The name of the application in which the page is loaded represented as a string (i.e. "Netscape")

appVersion The version information of the current browser as a string in the form "2.0 (Win16; I)" where 2.0 is the version number, Win16 is the platform and I indicates the international version (as opposed to U for the domestic version).

appCodeName The code name of the current browser (i.e. "Mozilla").

userAgent The user agent for the current browser as a string in the form "Mozilla/ 2.0 (Win16; I)".

```
<html>
<head>
<script>
function checkBrowser() {
  if(navigator.appName.toLowerCase().indexOf("microsoft") <0)
{
    alert("This page is best seen by MS Internet Explorer!");
  }
}
</script>
</head>
<body bgcolor="red" text="white" onLoad="checkBrowser()">
<h1>Check Browser</h1>
This page warns Netscape users!
</body>
</html>
```