W3C Working Draft

CSS Techniques for Web Content Accessibility Guidelines 2.0

W3C Working Draft 16 July 2001

This version:
http://www.w3.org/WAI/GL/WCAG20/WD-WCAG20-CSS-TECHS-20010716

Latest version:
http://www.w3.org/WAI/GL/WCAG20/WD-WCAG20-CSS-TECHS

Editor:
Wendy Chisholm, W3C

Abstract

This document describes techniques for authoring accessible Cascading Style Sheets (CSS). This is the first Working Draft associated with the latest Web Content Accessibility Guidelines 2.0 Working Draft.

Status of this document

This document is prepared by the Web Content Accessibility Guidelines Working Group (WCAG WG) to show how a CSS Techniques for WCAG 2.0 with more easily testable CSS-specific checkpoints **might** read. This draft is not based on consensus of the WCAG Working Group nor has it gone through W3C process.

Please refer to "Issue Tracking for CSS Techniques for WCAG 2.0" for a list of open issues related to this Working Draft. The "History of Changes to CSS Techniques for WCAG 2.0 Working Drafts" is also available.

This is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". A list of current W3C Recommendations and other technical documents is available.

Please send comments on this document to w3c-wai-gl@w3.org and prefix the subject line with [CSS-TECHS]. For example, "[CSS-TECHS] Question about section 2." The archives for this list are publicly available.

Table of Contents

---

1 Why CSS?

The term "style sheet" comes from the idea of a document that tells a human typesetter the house policies for how documents of particular types (not particular documents) should

appear.

David Woolley - 7 May 2001

1 Author benefits

- Increases consistency.
- Decreases maintenance.
- Separating markup for content from markup for presentation decreases clutter of markup.
- Able to create dynamic decorative effects.

To take advantage of these features in HTML:

- Identify the various terms included throughout a site and create a look for each in your style sheets.
- Mark up content properly according to its function, then create classes that further specify the function. For example, if you use the p element to set the byline on an article, create a class called "byline".
- Use the same "class" name for the same concept in all of the style sheets.
- Use external style sheets rather than embed style information.
- Use a minimal number of style sheets for your site.

Where style sheets have been optional to display HTML content, they are required to display information from XML applications. As explained in question C.22 of The XML FAQ by Peter Flynn:

> *In HTML, default styling is built into the browsers because the tagset of HTML is predefined and hardwired into browsers. In XML, where you can define your own tagset, browsers cannot know what names you are going to use and what they will mean, so you need a stylesheet if you want to display the formatted text.*

Therefore, the differences between CSS for XML and HTML are:

- define styles for elements and attributes (including class if that is defined in the language),
- content is marked up for function since there is no hardwired presentation,
- style information is usually required to be in an external style sheet (always?),

To apply the same external style sheet to both HTML and XML

- make the HTML class names the same as the XML element names,
- define each style twice: once with a period (HTML class name) and once without (XML element name). e.g., .copyright, copyright {font-size : .5em}

@@test and confirm these statements.

For more information about XML and CSS:

- Create Extensible Web Pages with XML and CSS
- The XML FAQ by Peter Flynn

For more information about creating accessible XML applications refer to the XML Accessibility Guidelines.

2 User benefits

- Ability to change presentation to suit preferences or needs such as:
  - higher contrast color combination,
  - increase font size,
  - present the information aurally with a preferred scheme (through a screen reader or over the phone),
  - present the information in Braille with a preferred scheme,
  - present the information on a smaller screen (mobile device with a small screen, such as a PDA).
- Ability to turn off style sheets if have a slow modem connection

In CSS2, if a user's style sheet contains "!important", it takes precedence over any rule in an author's style sheet. This is different from CSS1, where the author always has final say over styles. For more information on the !important selector refer to the CSS1 and CSS2 specifications. Also, note that since !important was introduced with CSS1 and is more widely implemented than CSS2, the author might have preference over the user in most of the existing user agents. However, it seems that !important is rarely used in author style sheets and therefore might work as defined in CSS2.

For examples of user's style sheets, refer to Appendix ?? - Example user's style sheets.

For more information on the accessibility of CSS refer to:

- CSS Design Principles in the CSS2 Spec
- Accessibility Features of CSS by Ian Jacobs and Judy Brewer

2 CSS Concepts - techniques and examples

@@ all of these need work. Mostly, all I did was reorder them, but I have added a few things.

- move the units of measure discussion up - first?
- for each section/technique point to browser support, if really buggy or inconsistent, give a warning of some sort.

1 Fonts

- Always specify a fallback generic font.
- Instead of using deprecated presentation elements and attributes, use the many CSS properties to control font characteristics: 'font-family', 'font-size', 'font-size-adjust', 'font-stretch', 'font-style', 'font-variant', and 'font-weight'.
- Use the following CSS2 properties to control font information
  - 'font', 'font-family', 'font-size', 'font-size-adjust', 'font-stretch', 'font-style', 'font-variant', and 'font-weight'

  instead of the following deprecated font elements and attributes in HTML:

    - FONT, BASEFONT, "face", and "size".
- If you must use HTML elements to control font information, use BIG and SMALL, which are not deprecated.

Example.

Always specify a fallback generic font:

```
BODY { font-family: "Gill Sans", sans-serif }
```

Example.

```
<STYLE type="text/css">
   P.important { font-weight: bold }
   P.less-important { font-weight: lighter; font-size: smaller }
   H2.subsection { font-family: Helvetica, sans-serif }
</STYLE>
```

2 Colors

2.1 Color Contrast

- Use numbers, not names, for colors.

Example.

Use numbers, not names, for colors:

```
H1 {color: #808000}
H1 {color: rgb(50%,50%,0%)}
```

Deprecated example.

```
H1 {color: red}
```

Use these CSS properties to specify colors:

- 'color', for foreground text color.
- 'background-color', for background colors.

- 'border-color', 'outline-color' for border colors.
- For link colors, refer to the :link, :visited, and :active pseudo-classes.

Ensure that foreground and background colors contrast well. If specifying a foreground color, always specify a background color as well (and vice versa).

9.2 Ensuring information is not in color alone

Ensure that information is not conveyed through color alone. For example, when asking for input from users, do not write "Please select an item from those listed in green." Instead, ensure that information is available through other style effects (e.g., a font effect) and through context (e.g,. comprehensive text links).

For instance, in this document, examples are styled by default (through style sheets) as follows:

- They are surrounded by a border.
- They use a different background color.
- They begin with the word "Example" or "Deprecated Example".
- They also end with the phrase "End example", but that phrase is hidden by default with 'display: none'. For user agents that don't support style sheets or when style sheets are turned off, this text helps delineate the end of an example for readers who may not be able to see the border around the example.

Quicktest! To test whether your document still works without colors, examine it with a monochrome monitor or browser colors turned off. Also, try setting up a color scheme in your browser that only uses black, white, and the four browser-safe greys and see how your page holds up.

Quicktest! To test whether color contrast is sufficient to be read by people with color deficiencies or by those with low resolution monitors, print pages on a black and white printer (with backgrounds and colors appearing in grayscale). Also try taking the printout and copying it for two or three generations to see how it degrades. This will show you where you need to add redundant cues (example: hyperlinks are usually underlined on Web pages), or whether the cues are too small or indistinct to hold up well.

For more information about colors and contrasts, refer to [LIGHTHOUSE].

3 Units of measure

> *Think about what needs to change in size and what doesn't. Raster images, for example, have fixed width and heights. We specify them in the img tag in pixels...Similarly, if I just want to move the text away from the border of the page, setting the padding on the body to 5 px shouldn't be a problem. Text, on the other hand, should always be free to change size (a good reason not to use text in images). But when my text doubles in size, do I need the gutter next to it to double in size as well? Probably not. In fact, doubling the gutter might even decrease the usability of the page.*

Charles Munat - 6 May 2001

- Use "em" or % for properties that need to change:
  - font size,

    - line height,
    - @@create exhaustive list. any exceptions?
- Use px for properties that do not need to change:
    - height and width of raster images,
    - margins,
    - @@create exhaustive list. any exceptions?

Example.

Use em to set font sizes, as in:

```
H1 { font-size: 2em }
```

rather than:

```
H1 { font-size: 12pt }
```

Example.

Use relative length units and percentages.

```
BODY { margin-left: 15%; margin-right: 10%}
```

Example.

Only use absolute length units when the physical characteristics of the output medium are known.

```
.businesscard { font-size: 8pt }
```

4 Text style effects

The following CSS2 properties can be used to style text:

- Case: 'text-transform' (for uppercase, lowercase, and capitalization).
- Shadow effects: 'text-shadow'
- Underlines, overlinks, blinking: 'text-decoration'. **Note.** If blinking content (e.g., a headline that appears and disappears at regular intervals) is used, provide a mechanism for stopping the blinking. In CSS, 'text-decoration: blink' will cause content to blink and will allow users to stop the effect by turning off style sheets or overriding the rule in a user style sheet. Do not use the BLINK and MARQUEE elements. These elements are not part of any W3C specification for HTML (i.e., they are non-standard elements).

5 Text instead of images

Content developers should use style sheets to style text rather than representing text in images. Using text instead of images means that the information will be available to a greater number of users (with speech synthesizers, braille displays, graphical displays, etc.). Using style sheets will also allow users to override author styles and change colors or fonts sizes more easily.

If it is necessary to use a bitmap to create a text effect (special font, transformation, shadows, etc.) the bitmap must be accessible (see the sections on text equivalents and alternative pages).

Example.

In this example, the inserted image shows the large red characters "Example", and is captured by the value of the "alt" attribute.

```
<P>This is an
   <IMG src="BigRedExample.gif" alt="example">
   of what we mean.
</P>
```

6 Text formatting and position

The following CSS2 properties can be used to control the formatting and position of text:

- Indentation: 'text-indent'. Do not use the BLOCKQUOTE or any other structural element to indent text.
- Letter/word spacing: 'letter-spacing', 'word-spacing'. For example instead of writing "H E L L O" (which users generally recognize as the word "hello" but would hear as individual letters), authors may create the same visual effect with the 'word-spacing' property applied to "HELLO". Text without spaces will be transformed more effectively to speech.
- White space: 'white-space'. This property controls the white space processing of an element's content.
- Text direction: 'direction', 'unicode-bidi'.
- The :first-letter and :first-line pseudo-elements allow authors to refer to the first letter or line of a paragraph of text.

The following example shows how to use style sheets to create a drop-cap effect.

Example.

```
<HEAD>
<TITLE>Drop caps</TITLE>
<STYLE type="text/css">
      .dropcap { font-size : 120%; font-family : Helvetica }
</STYLE>
```

```
</HEAD>
<BODY>
<P><SPAN class="dropcap">O</SPAN>nce upon a time...
</BODY>
```

Note. As of the writing of this document, the CSS pseudo-element ':first-letter', which allows content developers to refer to the first letter of a chunk of text, is not widely supported.

7 Rules and borders

Rules and borders may convey the notion of "separation" to visually enabled users but that meaning cannot be inferred out of a visual context.

Use these CSS properties to specify border styles:

- 'border', 'border-width', 'border-style', 'border-color'.
- 'border-spacing' and 'border-collapse' for tables.
- 'outline, 'outline-color', 'outline-style', and 'outline-width' for dynamic outlines.

Authors should use style sheets to create rules and borders.

Example.

In this example, the H1 element will have a top border that is 2px thick, red, and separated from the content by 1em:

```
<HEAD>
<TITLE>Redline with style sheets</TITLE>
<STYLE type="text/css">
     H1 { padding-top: 1em; border-top: 2px red }
</STYLE>
</HEAD>
<BODY>
<H1>Chapter 8 - Auditory and Tactile Displays</H1>
</BODY>
```

If a rule (e.g., the HR element) is used to indicate structure, be sure to indicate the structure in a non-visual way as well. (e.g., by using structural markup).

Example.

In this example, the DIV element is used to create a navigation bar, which includes a horizontal separator.

```
<DIV class="navigation-bar">
   <HR>
   <A rel="Next" href="next.html">[Next page]</A>
   <A rel="Previous" href="previous.html">[Prevous page]</A>
   <A rel="First" href="first.html">[First page]</A>
</DIV>
```

8 Generated content

- Provide a text equivalent for any important image or text generated by style sheets (e.g., via the 'background-image', 'list-style', or 'content' properties).
- Ensure that important content appears in the document object. Text generated by style sheets is not part of the document source and will not be available to assistive technologies that access content through the Document Object Model Level 1 ([[DOM1]).

CSS2 includes several mechanisms that allow content to be generated from style sheets:

- the :before and :after pseudo-elements and the 'content' property. When used together, these allow authors to insert markers (e.g., counters and constant strings such as "End Example" in the examples below) before or after and element's content.
- the 'cue', 'cue-before', and 'cue-after' properties. This properties allow users to play a sound before or after an element's content.
- List styles, which may be numbers, glyphs, or images (usually associated with the LI element in HTML). CSS2 adds international list styles to the styles defined in CSS1. See the 'list-style-type' and 'content' properties.

Generated content can serve as markers to help users navigate a document and stay oriented when they can't access visual clues such as proportional scrollbars, frames with tables of contents, etc.

For instance, the following user style sheet would cause the words "End Example" to be generated after each example marked up with a special class value in the document:

Example.

```
DIV.example:after {
  content: End Example
}
```

Users could also, for example, number paragraphs so that they could locate their current reading position in a document:

Example.

```
P:before {
  content: counter(paragraph) ". " ;
    counter-increment: paragraph
}
```

9 Providing contextual clues in HTML lists

Content developers are encouraged to use UL for unordered lists and OL for ordered lists (i.e., use markup appropriately) combined with CSS to provide contextual clues.

The following CSS2 style sheet shows how to provide compound numbers for nested lists created with either UL or OL elements. Items are numbered as "1", "1.1", "1.1.1", etc.

Example.

```
<STYLE type="text/css">
    UL, OL { counter-reset: item }
    LI { display: block }
    LI:before { content: counters(item, "."); counter-increment: item }
</STYLE>
```

Until either CSS2 is widely supported by user agents or user agents allow users to control rendering of lists through other means, authors should consider providing contextual clues in nested lists. The following CSS1 mechanism shows how to hide the end of a list when style sheets are turned on, and to reveal it when style sheets are turned off, when user style sheets override the hiding mechanism, or when style sheets aren't supported.

Example.

```
<STYLE type="text/css">
    .endoflist { display: none }
</STYLE>
<UL>
   <LI>Paper:
      <UL>
         <LI>Envelopes
         <LI>Notepaper
         <LI>Letterhead
         <LI>Poster paper
         <span class="endoflist">(End of Paper)</span>
      </UL>
   <LI>Pens:
      <UL>
         <LI>Blue writing pens
         <LI>whiteboard pens
```

```
                <span class="endoflist">(End of Pens)</span>
             </UL>
          <LI>Fasteners:
             <UL>
                <LI>paper clips
                <LI>staples
                <LI>Big lengths of rope.
                <span class="endoflist">(End of Fasteners)</span>
             </UL>
          <span class="endoflist">(End of Office Supplies)</span>
       </UL>
```

Note: This example does not help the case of wrapping list items. With some more effort, the author could put similar markup at the end of each list item.

10 Relative positioning

@@best title? want to distinguish from next section - absolute positioning.

Layout, positioning, layering, and alignment should be done through style sheets (notably by using CSS floats and absolute positioning):

- 'text-indent', 'text-align', 'word-spacing', 'font-stretch'. Each of these properties allows users to control spacing without adding additional spaces. Use 'text-align: center' instead of the deprecated CENTER element.
- 'margin', 'margin-top', 'margin-right', 'margin-bottom', 'margin-left'. With these properties, authors can create space on four sides of an element's content instead of adding non-breaking spaces ( ).
- 'float', 'position', 'top', 'right', 'bottom', 'left'. With these properties, the user can control the visual position of almost any element in a manner independent of where the element appears in the document. Authors should always design documents that make sense without style sheets (i.e., the document should be written in a "logical" order) and then apply style sheets to achieve visual effects. The positioning properties may be used to create margin notes (which may be automatically numbered), side bars, frame-like effects, simple headers and footers, and more.
- The 'empty-cells' property allows users to leave table cells empty and still give them proper borders on the screen or on paper. A data cell that is meant to be empty should not be filled with white space or a non-breaking space just to achieve a visual effect.

10.1 If you must use images as spacers

Provide text equivalents for all images, including invisible or transparent images.

If content developers cannot use style sheets and must use invisible or transparent images (e.g., with IMG) to lay out images on the page, they should specify `alt=""` for them.

Deprecated example.

Authors should **not** use spaces for the value of "alt" to prevent the words from running together when the image is not loaded:

```
   my poem requires a big space<IMG src="10pttab.gif"
alt="   ">here
```

In this next example, an image is used to force a graphic to appear in a certain position:

```
<IMG src="spacer.gif" alt="spacer">
<IMG src="colorfulwheel.gif" alt="The wheel of fortune">
```

11 Using absolute positioning accessibly

Using the positioning properties of CSS2, content may be displayed at any position on the user's viewport. The order in which items appear on a screen may be different than the order they are found in the source document. The following example demonstrates a few principles:

1. the text appears visually in the browser in a different order than in the markup.
2. CSS positioning may be used to create tabular effects. A TABLE element could have been used to create the same effect.

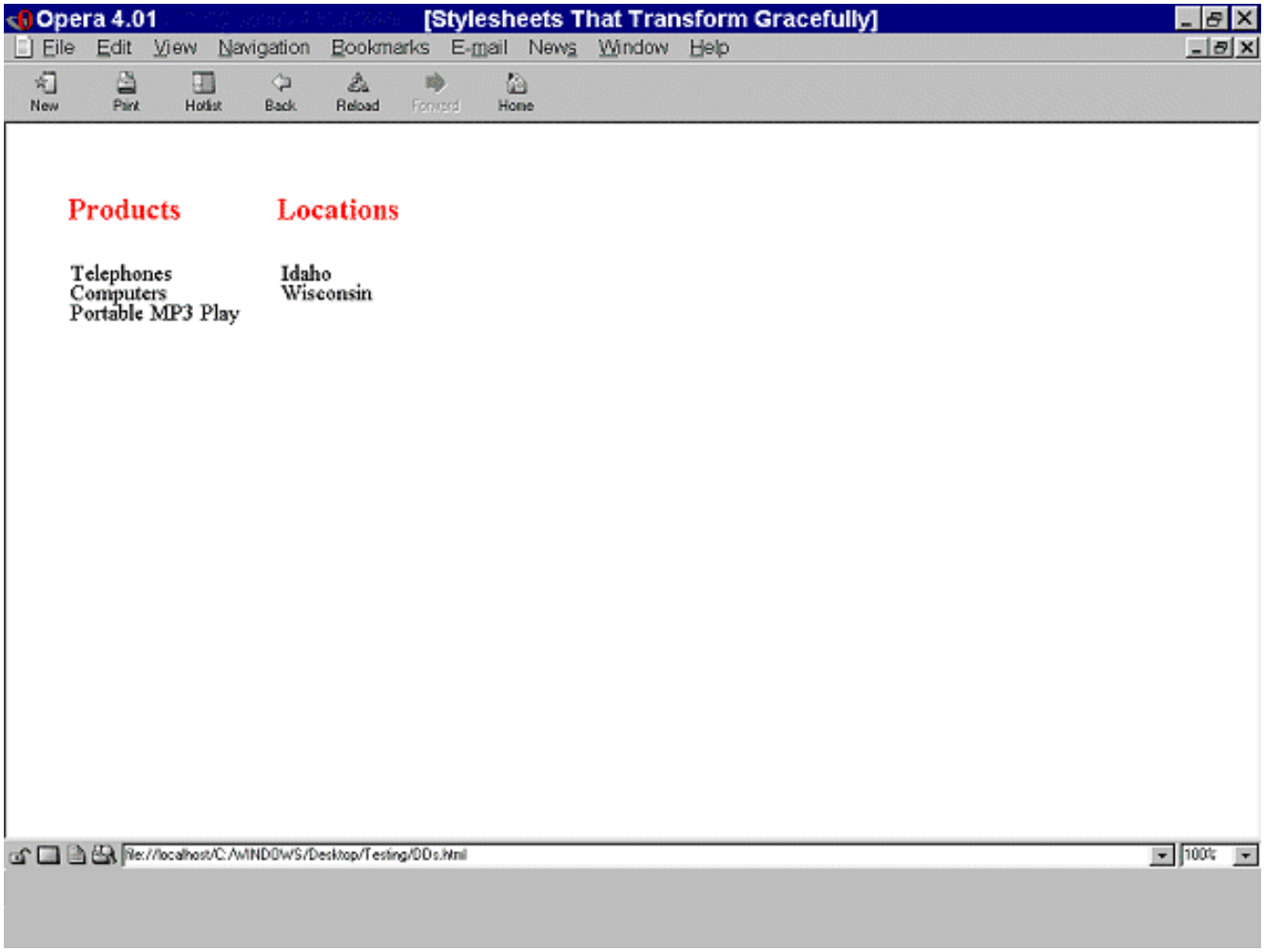
Note that a class is defined for each object that is being positioned. The use of "id" could be substituted for "class" in these examples. "Class" was used because in the live example, the objects are replicated and thus not unique.

Deprecated example.

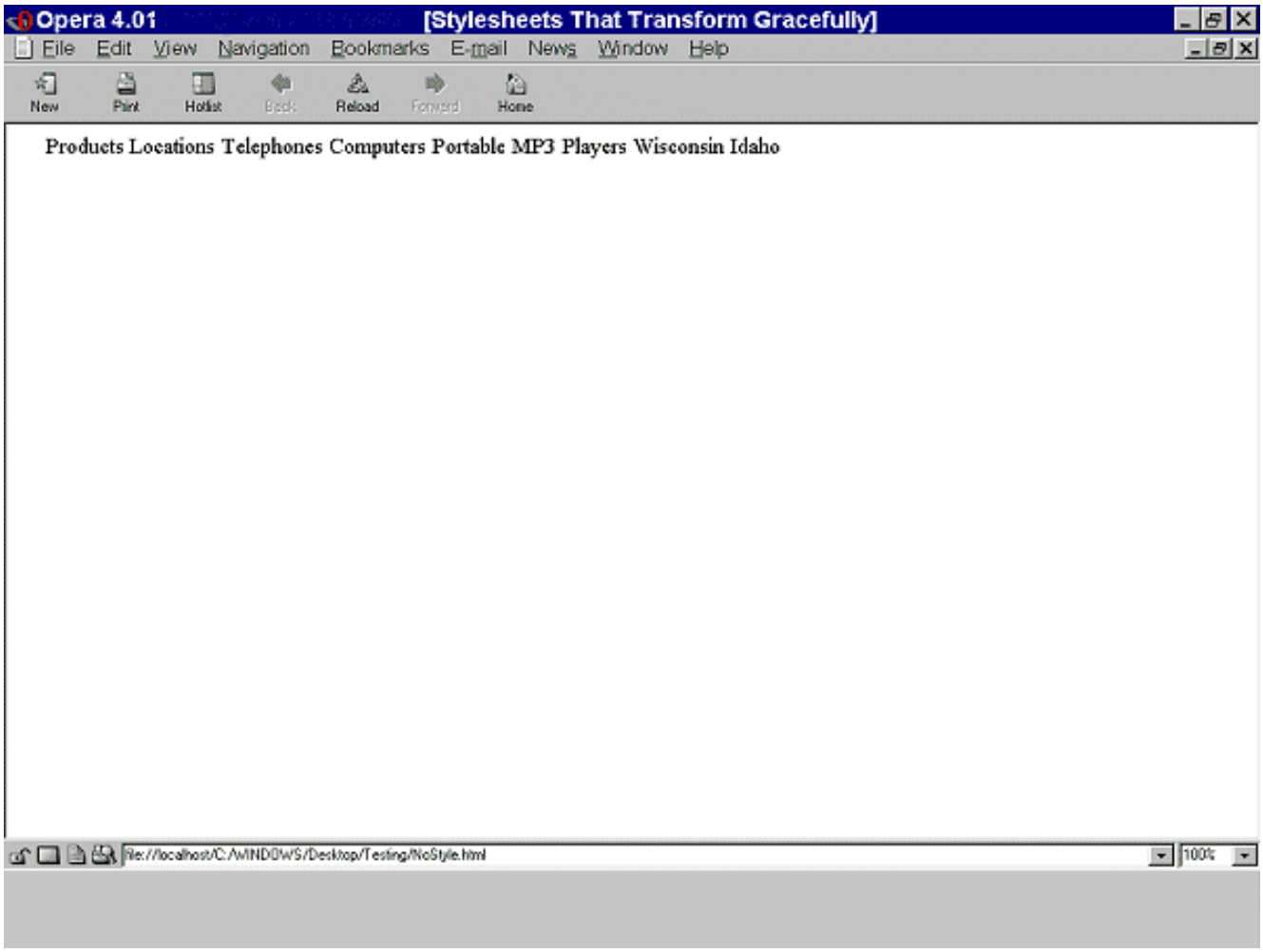
```
<head><style type="text/css">
  .menu1 { position: absolute; top: 3em; left: 0em;
           margin: 0px; font-family: sans-serif;
           font-size: 120%; color: red; background-color: white }
  .menu2 { position: absolute; top: 3em; left: 10em;
           margin: 0px; font-family: sans-serif;
           font-size: 120%; color: red; background-color: white }
  .item1 { position: absolute; top: 7em; left: 0em; margin: 0px }
  .item2 { position: absolute; top: 8em; left: 0em; margin: 0px }
  .item3 { position: absolute; top: 9em; left: 0em; margin: 0px }
  .item4 { position: absolute; top: 7em; left: 14em; margin: 0px }
  .item5 { position: absolute; top: 8em; left: 14em; margin: 0px }
  #box { position: absolute; top: 5em; left: 5em }
</style></head>
<body>
<div class="box">
  <span class="menu1">Products</span>
  <span class="menu2">Locations</span>
  <span class="item1">Telephones</span>
  <span class="item2">Computers</span>
```

```
        <span class="item3">Portable MP3 Players</span>
        <span class="item5">Wisconsin</span>
        <span class="item4">Idaho</span>
      </div>
      </body>
```

When style sheets are applied, the text appears in two columns. Elements of class "menu1" (Products) and "menu2" (Locations) appear as column headings. "Telephones, Computers, and Portable MP3 Players" are listed under Products and "Wisconsin" and "Idaho" are listed under Locations as shown in this screen shot:



When style sheets are not applied, all of the text appears in one line of words, "Products Locations Telephones Computers Portable MP3 Players Wisconsin Idaho". They appear in the order in which they are written in the source. Therefore, what appear as column headings when style sheets are applied are the first phrases since they were defined first in the source. The following screen shot illustrates this:



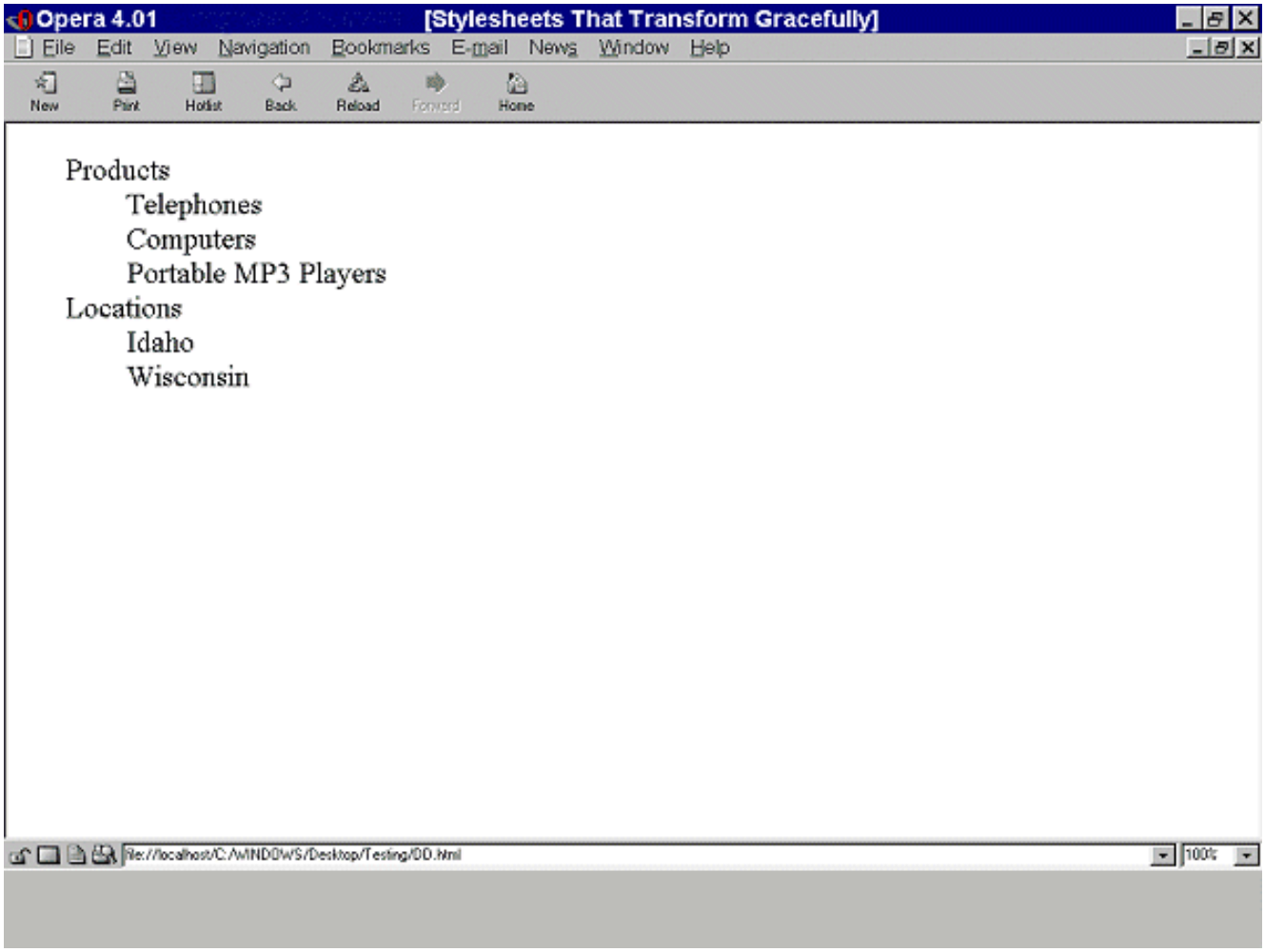
The following example shows that the same visual appearance may be created in a browser that support style sheets as well as creating a more meaningful presentation when style sheets are not applied. Structural markup (definition lists) have been applied to the content. Note that the margins have been set to 0 since in HTML browsers, definition lists are displayed with a margin set on the DD element.

Example.

```
<head><style type="text/css">
  .menu1 { position: absolute; top: 3em; left: 0em;
           margin: 0px; font-family: sans-serif;
           font-size: 120%; color: red; background-color: white }
  .menu2 { position: absolute; top: 3em; left: 10em;
           margin: 0px; font-family: sans-serif;
           font-size: 120%; color: red; background-color: white }
  .item1 { position: absolute; top: 7em; left: 0em; margin: 0px }
  .item2 { position: absolute; top: 8em; left: 0em; margin: 0px }
  .item3 { position: absolute; top: 9em; left: 0em; margin: 0px }
  .item4 { position: absolute; top: 7em; left: 14em; margin: 0px }
  .item5 { position: absolute; top: 8em; left: 14em; margin: 0px }
  #box { position: absolute; top: 5em; left: 5em }
</style></head>
```

```
<body>
<div class="box">
<dl>
  <dt class="menu1">Products</dt>
    <dd class="item1">Telephones</dd>
    <dd class="item2">Computers</dd>
    <dd class="item3">Portable MP3 Players</dd>
  <dt class="menu2">Locations</dt>
   <dd class="item4">Idaho</span>
   <dd class="item5">Wisconsin</span>
  </dt>
</dl>
</div>
</body>
```

When style sheets are applied, it looks as it did before. However, now when the style sheets are not applied, the text appears in a definition list rather than a string of words. What appear as column headings when style sheets are applied, appear as defined terms when style sheets are not applied as demonstrated in the following screen shot.



Note. Experience the difference between these examples for yourself: test file for style sheets that transform gracefully.

User Agent Note: At the current time (July 2001), user agent support for positioning CSS is spotty; it is most reliable on <browser> and <browser>. Version <n> of <browser> and earlier, and version <k> of <browser> and earlier do not support CSS-P at all. You may want to provide a legacy version [defn link] for these browsers. @@include summary of webreview info as well as the bugs and workarounds stuff - point to these as well.

13 Creating movement with style sheets and scripts

@@discuss Javascript, and style sheets, point to the MWC example or "the company" example. points to make/demonstrate:

- hide/show content,
- change presentation (movement and colors)

14 Aural Cascading Style Sheets

CSS2's aural properties provide information to non-sighted users and voice-browser users much in the same way fonts provide visual information. The following example show how various sound qualities (including 'voice-family', which is something like an audio font) can let a user know that spoken content is a heading:

Example.

```
H1 {
  voice-family: paul;
  stress: 20;
  richness: 90;
  cue-before: url("ping.au")
}
```

The following properties are part of CSS2's aural cascading style sheets.

- 'volume' controls the volume of spoken text.
- 'speak' controls whether content will be spoken and, if so, whether it will be spelled or spoken as words.
- 'pause', 'pause-before', and 'pause-after' control pauses before and after content is spoken. This allows users to separate content for better comprehension.
- 'cue', 'cue-before', and 'cue-after' specify a sound to be played before and after content, which can be valuable for orientation (much like a visual icon).
- 'play-during' controls background sounds while an element is rendered (much like a background image).
- 'azimuth' and 'elevation' provide dimension to sound, which allows users to distinguish voices, for example.
- 'speech-rate', 'voice-family', 'pitch', 'pitch-range', 'stress', and 'richness' control the quality of spoken content. By varying these properties for different elements, users can fine-tune how content is presented aurally.

- 'speak-punctuation' and 'speak-numeral' control how numbers and punctuation are spoken, which has an effect on the quality of the experience of aural browsing.

Furthermore, the 'speak-header' property describes how table header information is to be spoken before a table cell.

15 Access to alternative representations of content

CSS2 lets users access alternative representations of content that is specified in attribute values when the following are used together:

- attribute selectors.
- the attr() function and the 'content' property'
- the :before and :after pseudo-elements

In the following example, the value of the "alt" attribute for the IMG element is rendered after the image (visually, aurally, etc.):

Example.

```
IMG:after {
   content: attr(alt)
}
```

Note that the value of the attribute is displayed even though the image may not be (e.g., the user has turned off images through the user interface).

16 Media types

The CSS2 "media types" (used with @media rules) allow authors and users to design style sheets that will cause documents to render more appropriately on certain target devices. These style sheets can tailor content for presentation on braille devices, speech synthesizers, or TTY devices. Using "@media" rules can also reduce download times by allowing user agents to ignore inapplicable rules.

3 Assessing accessibility of CSS with HTML

@@Auto checks: fill in with list of tools that automatically provide results to these checkpoints. Also, in manual provide list of tools that will give warning for possible problem.

WCAG 1.0 checkpoints that are not easily identifiable in WCAG 2.0 checkpoints:

- units of measure
- color contrast

WCAG 2.0 checkpoint	Manual checks	Auto checks	Examples and Techniques
1.1 Provide a text equivalent for all non-text content.			Generated content
1.2 Synchronize text equivalents with multimedia presentations.	N/A	N/A	Refer to client-side scripting techniques
1.3 Synchronize a description of the essential visual information in multimedia presentations.	N/A	N/A	Refer to client-side scripting techniques
1.4 Use markup or a data model to provide the logical structure of content.	Generate outline and determine if structure is good		
1.5 Separate content and structure from presentation.	Increase font size in the browser settings. does all the text increase in size?	HTML Validator	
2.1 Provide more than one path or method to find content.	N/A	N/A	Refer to Core techniques
2.2 Provide consistent responses to user actions.	N/A	N/A	Refer to client-side scripting techniques
2.3 Give users control of mechanisms that cause extreme changes in context.	N/A	N/A	Refer to client-side scripting techniques
2.4 Give users control over how long they can spend reading or interacting with content.	N/A	N/A	Refer to client-side scripting techniques
2.5 Use device-independent event handlers.	N/A	N/A	Refer to client-side scripting techniques
3.1 Use consistent presentation.	Check for use of external style sheets. Check that classes are used for function rather than presentation (?)		
3.2 Emphasize structure through presentation, positioning, and labels.	Print on paper using only black and white ink. Is all the info present? i.e., is info presented not in color alone?		
3.3 Write clearly and simply.	N/A	N/A	Refer to Core techniques

3.4 Use multimedia to illustrate concepts.			
3.5 Summarize complex information.	N/A	N/A	Refer to Core techniques
3.6 Define key terms, abbreviations, acronyms, and specialized language.	N/A	N/A	Refer to Core techniques
3.7 Divide information into smaller, more manageable units.	N/A	N/A	Refer to Core techniques
4.1 Choose languages, API's, and protocols that support the use of these guidelines.	Use CSS1 or CSS2 with HTML 4.01 or XHTML 1.1		
4.2 Use languages, API's, and protocols according to specification.	check that link element in head of page references an external style sheet. ensure that no style attributes exist on elements, ensure that font element is not used. etc.	W3C CSS Validator W3C HTML Validator	
4.3 Design assistive-technology compatible user interfaces.	Does your site work with a variety of assistive technologies, such as screen readers, magnifiers, on screen keyboards, etc.?		
4.4 Design content so that when presentation effects are turned off or not supported the content is still usable.	Use a browser that does not support or that will turn off CSS to determine if content is still usable.		

4 Assessing accessibility of CSS with XML

@@Similar to above table, but XML-specific.

---

5 References

For the latest version of any W3C specification please consult the list of W3C Technical Reports at http://www.w3.org/TR.

[CSS1]
"CSS, level 1 Recommendation", B. Bos, H. Wium Lie, eds., 17 December 1996, revised 11 January 1999. This CSS1 Recommendation is http://www.w3.org/TR/1999/REC-CSS1-19990111. The latest version of CSS1 is available at http://www.w3.org/TR/REC-CSS1.

[CSS2]
"CSS, level 2 Recommendation", B. Bos, H. Wium Lie, C. Lilley, and I. Jacobs, eds., 12 May 1998. This CSS2 Recommendation is http://www.w3.org/TR/1998/REC-CSS2-19980512/. The latest version of CSS2 is available at http://www.w3.org/TR/REC-CSS2.

[WCAG10]
"Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, and I. Jacobs, eds., 5 May 1999. This WCAG 1.0 Recommendation is http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/.

[WCAG10-TECHS]
"Techniques for Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, I. Jacobs, eds. This document explains how to implement the checkpoints defined in "Web Content Accessibility Guidelines 1.0". The latest draft of the techniques is available at http://www.w3.org/TR/WCAG10-TECHS/.

6 Resources

Note: *W3C does not guarantee the stability of any of the following references outside of its control. These references are included for convenience. References to products are not endorsements of those products.*

6.1 Other guidelines

[UWSAG]
"The Unified Web Site Accessibility Guidelines", G. Vanderheiden, W. Chisholm, eds. The Unified Web Site Guidelines were compiled by the Trace R & D Center at the University of Wisconsin under funding from the National Institute on Disability and Rehabilitation Research (NIDRR), U.S. Dept. of Education.

6.2 Accessibility resources

[LIGHTHOUSE]
The Lighthouse provides information about accessible colors and contrasts.

NCI-NIH stuff
helpful, good intro.

IBM
also really good.

6.3 Tutorials and other help with CSS

WestCiv
Lots of links!

Basic "old timers" typesetting practices
Explains history of typesetting, fonts, em-dash, measures of units, etc.

CSS Bugs and Workarounds
Haven't thoroughly reviewed, but seems to be a good complement to the WebReview charts as to exactly what goes wrong in which browsers and how to work around.

7 Acknowledgments

Web Content Guidelines Working Group Co-Chairs:
Jason White, University of Melbourne
Gregg Vanderheiden, Trace Research and Development

W3C Team contact:
Wendy Chisholm

We wish to thank the following people who have contributed their time and valuable comments to shaping these guidelines:
Harvey Bingham, Kevin Carey, Chetz Colwell, Neal Ewers, Geoff Freed, Al Gilman, Larry Goldberg, Jon Gunderson, Eric Hansen, Phill Jenkins, Leonard Kasday, George Kerscher, Marja-Riitta Koivunen, Josh Krieger, Chuck Letourneau, Scott Luebking, William Loughborough, Murray Maloney, Charles McCathieNevile, MegaZone (Livingston Enterprises), Masafumi Nakane, Mark Novak, Charles Oppermann, Mike Paciello, David Pawson, Michael Pieper, Greg Rosmaita, Liam Quinn, Dave Raggett, T.V. Raman, Robert Savellis, Jutta Treviranus, Steve Tyler, and Jaap van Lelieveld

Appendix - User's style sheets

In CSS1, authors always had final say over styles. In CSS2, if a user's style sheet contains "!important", it takes precedence over any applicable rule in an author's style sheet. This is an important feature to users who require or must avoid certain color combinations or contrasts, users who require large fonts,

etc. For instance, the following rule specifies a large font size for paragraph text and would override an author rule of equal weight:

Example.

```
P { font-size: 24pt ! important }
```

The CSS2 'inherit' value - available for every property - leads to compact "!important" style rules that govern most or all of a document. For instance, the following style rules force all backgrounds to white and all foreground colors to black:

Example.

```
/*   Sets the text color to black
and the background color to
white for the document body.  */

BODY {
  color: black ! important ;
  background: white ! important
}

 /*   Causes the values of 'color' and 'background'
to be inherited by all other elements,
strengthened by !important. Note that this
may be overridden by other, more specific,
user styles.  */

* {
  color: inherit ! important ;
  background: inherit ! important
}
```

CSS2 also includes these user control features:

- System colors (for 'color', 'background-color', 'border-color', and 'outline-color') and system fonts (for 'font') mean that users may apply their system color and font preferences to Web documents.
- Dynamic outlines (the 'outline' property) allow users (e.g., with low vision) to create outlines around content that don't affect layout but do provide highlight information.

For example, to draw a thick black line around an element when it has the focus, and a thick red line when it is active, the following rules can be used:

Example.

```
:focus  { outline: thick solid black }
:active { outline: thick solid red }
```