Web Accessibility

Web Standards and Regulatory Compliance

Jim Thatcher
Michael R. Burks
Christian Heilmann
Shawn Lawton Henry
Andrew Kirkpatrick
Patrick H. Lauke
Bruce Lawson
Bob Regan
Richard Rutter
Mark Urban
Cynthia D. Waddell

Web Accessibility: Web Standards and Regulatory Compliance

ISBN-13 (pbk): 978-1-59059-638-8

ISBN-10 (pbk): 1-59059-638-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editor
Chris Mills

Development Editor
Adam Thomas

Technical Reviewers
Bruce Lawson
Gez Lemon

Editorial Consultant
Bruce Lawson

Editorial Board
Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade

Project Manager
Beth Christmas

Copy Edit Manager
Nicole LeClerc

Copy Editor
Marilyn Smith

Assistant Production Director
Kari Brooks-Copony

Production Editor
Laura Esterman

Compositor
Dina Quan

Artist
April Milne

Proofreader
Elizabeth Berry

Indexer
Michael Brinkman

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski

6 ACCESSIBLE CONTENT

by Jim Thatcher

The subject of accessibility on the Web can be divided into three main categories: accessible web content, accessible navigation, and accessible interaction. This chapter addresses the first of these: creating accessible content, including the text, images, and audio files that might be available on a web page. Accessible navigation is covered in Chapter 7, and accessible interaction is the subject of Chapter 8. In addition to these basic treatments of accessibility for the Web, see also the coverage of advanced topics, especially Cascading Style Sheets (CSS) in Chapter 9 and JavaScript in Chapter 10.

Content is accessible only if it can be "viewed" or accessed by people with disabilities. Besides getting the information from the page, people with disabilities must be able to use all the functions available to nondisabled users: links, buttons, form controls, and so on. Accessible content must be compatible with assistive technologies, particularly screen readers. There must be alternatives to pure visual content for people who can't see and alternatives to pure auditory content for people who can't hear.

To explain issues and solutions for accessibility, I will refer to four assistive technologies discussed in Chapter 5: one talking browser and three screen readers. The screen readers are Hal Version 6.5 (`www.dolphincomputeraccess.com/`); JAWS for Windows Version 7.0, by Freedom Scientific (`www.freedomscientific.com`); and Window-Eyes Version 5.5, by GW Micro (`www.gwmicro.com/`). The talking browser is IBM Home Page Reader Version 3.04 (`www.ibm.com/able/hpr.html`). All are intended to be used by people with very limited or no useful vision. As explained in Chapter 5, demonstration versions of these products are available from their respective websites.

Guidelines for Accessible Web Development

The two important sets of guidelines for accessible web development are the Web Content Accessibility Guidelines (WCAG) and the Section 508 Standards. In order to share in the wisdom of the corresponding community of experts and to give context to the discussion, each of the topics in this chapter will include a review of what the guidelines have to say on the subject. Here, I will provide an overview of these guidelines.

WCAG 1.0

WCAG 1.0, from the Web Accessibility Initiative (WAI, `www.w3.org/WAI`) of the World Web Consortium (W3C, `www.w3.org`) became an official Recommendation of the W3C on May 5, 1999. The WCAG consists of 14 guidelines, or principles of accessible design (`www.w3.org/TR/WCAG10`). Each guideline includes a set of checkpoints that explain how the guideline applies to web development. The checkpoints (`www.w3.org/TR/WCAG10/full-checklist.html`) are prioritized according to the following criteria:

> [Priority 1] A web content developer **must** satisfy this checkpoint. Otherwise, one or more groups will find it impossible to access information in the document. Satisfying this checkpoint is a basic requirement for some groups to be able to use web documents.

> [Priority 2] A web content developer **should** satisfy this checkpoint. Otherwise, one or more groups will find it difficult to access information in the document. Satisfying this checkpoint will remove significant barriers to accessing web documents.
>
> [Priority 3] A web content developer **may** address this checkpoint. Otherwise, one or more groups will find it somewhat difficult to access information in the document. Satisfying this checkpoint will improve access to web documents.

There are 65 checkpoints in all; 16 of them are Priority 1, 30 are Priority 2, and 19 are Priority 3.

WCAG 2.0

The 1999 WCAG is being revised by the WAI at the time of this writing. The WCAG revision process has taken about five years and entered "Last Call" on April 27, 2006. Last Call comments were allowed through late June. After processing Last Call comments, and assuming none are showstoppers, the WCAG Working Group will solicit and then verify WCAG 2.0 compliant websites. This testing phase will last about four months.

There are two dramatic differences between the first edition of these guidelines, WCAG 1.0, and the second edition, WCAG 2.0 (`www.w3.org/TR/WCAG20`). Like WCAG 1.0, the second edition is organized around guidelines. WCAG 2.0 includes 13 guidelines, and each contains a list of items similar to the checkpoints of WCAG 1.0 that are called *success criteria*. These are testable statements of what needs to be done to satisfy each guideline. These success criteria are technology-independent and thus very general. The generality is the first dramatic difference between WCAG 2.0 and WCAG 1.0 and it leads to wording like this:

> SC 2.4.4 Each link is programmatically associated with text from which its purpose can be determined.

Instead of this:

> 13.1 Clearly identify the target of each link. [Priority 2]

With more general constructs, the applicability is also generalized. Success Criterion 2.4.4 (as of Last Call) about link text also applies to server-side image maps requiring text links for each image map hotspot.

After you have become accustomed to the style and the vocabulary of the WCAG 2.0 document, the technology-neutral approach begins to make sense.

The second major difference between WCAG 1.0 and WCAG 2.0 is that the success criteria under each guideline are not prioritized by their importance or priority, as they were in

WCAG 1.0. Instead, they are ranked by the extent to which the web design and development process must be modified in order to meet the success criteria. The ranking is by levels, as follows (as defined at the time of this writing):

- Level 1 success criteria:
  1. Achieve a minimum level of accessibility.
  2. Can reasonably be applied to all web resources.
- Level 2 success criteria:
  1. Achieve an enhanced level of accessibility.
  2. Can reasonably be applied to all web resources.
- Level 3 success criteria:
  1. Achieve additional accessibility enhancements.
  2. Cannot necessarily be applied to all web resources.

Whereas Level A conformance with WCAG 1.0 required complying with all Priority 1 checkpoints, Level A conformance with WCAG 2.0 requires compliance with all Level 1 success criteria.

Read more about this very important set of guidelines in Chapter 14.

Section 508 Standards

The U.S. Access Board (`www.access-board.gov`) has issued access standards (`www.access-board.gov/508.htm`) for federal electronic and information technology as required under Section 508 of the Rehabilitation Act: *The Electronic and Information Technology Accessibility Standards, 36 CFR Part 1194, Web-based Intranet and Internet Information and Applications* (1194.22). The Access Board has also published an online guide (`www.access-board.gov/sec508/guide`) for all the standards. This guide site is the easiest route to view the 16 provisions of the Section 508 Standards for the Web.

The force of the Section 508 Standards is that electronic and information technology purchased by the U.S. federal government must comply with these provisions. Because of that force of law, these provisions are seen as playing an important role in defining accessibility, especially in the U.S. There are additional applications of the Section 508 Standards. Several states, including my home state of Texas, use the Section 508 Standards as at least a reference for state accessibility requirements.

Many of the Section 508 provisions correspond to Priority 1 WCAG checkpoints with minor changes for regulatory wording. Some of the Priority 1 checkpoints were deemed by the Access Board to be too restrictive on web development or too difficult to judge for compliance. In addition, the Section 508 Standards add provisions that combine WCAG 1.0 checkpoints of lower priorities, like the Section 508 provision for accessible forms.

The Association of Assistive Technology Act Projects (ATAP, `www.ataporg.org/`) sponsored a detailed side-by-side comparison (`www.jimthatcher.com/sidebyside.htm`) of the Section 508 provisions and the Priority 1 WCAG 1.0 checkpoints. In the same spirit, I have

compiled a mapping from the Section 508 Standards to the WCAG 2.0 success criteria, which you can find at `www.jimthatcher.com/508wcag2.htm`.

This chapter will address the individual WCAG guidelines and Section 508 web accessibility provisions, and show how each one can be met.

Using Text Equivalents for Images

The one accessibility issue that is probably more important than all others is providing text information for web content that is nontextual. Nontext items include images, image maps, image buttons, audio files, and multimedia files that provide both audio and video.

The reason text is so important is that people with sensory disabilities have ways of accessing text even if they are unable to access the nontext content. If there is a textual equivalent for an image, users who are blind will be able to listen to that text with their screen reader or talking browser. Most images on web pages are simple, and so their text equivalents are simple, too. Even when the images carry more information—as with charts or graphs—there are techniques for conveying that information textually.

A person with impaired hearing may find it difficult or impossible to get information from an audio file. The text equivalent of an audio file is a word-for-word transcript. The transcript can be read by itself or viewed while listening to the audio. In this way, the person who is hearing impaired can get the equivalent information. (See the "Using Text Equivalents for Audio" section later in this chapter.)

It would be convenient if there were a mechanism for creating some sort of word-for-word transcript of an image, but, of course, there is no such thing. Therefore, the process of providing adequate and useful alternative text for images—text that does not overburden the user—requires judgment and style.

Guidelines and Standards for Text Equivalents

One measure of the perceived importance of the text equivalents can be gleaned from the fact that the requirement for text equivalents is the first item in most guideline lists.

WCAG 1.0

The first checkpoint in the WAI WCAG deals with providing text equivalents for nontext elements.

> 1.1 Provide a text equivalent for every non-text element (for example, via "`alt`", "`longdesc`", or in element content). This includes: images, graphical representations of text (including symbols), image map regions, animations (for example, animated GIFs), applets and programmatic objects, ascii art, frames, scripts, images used as list bullets, spacers, graphical buttons, sounds (played with or without user interaction), stand-alone audio files, audio tracks of video, and video.

This checkpoint from the Guidelines Working Group of the WAI lists more types of nontext elements than we will be considering in this chapter. Applets, programming objects, and scripts can often be very textual. The issues with these kinds of content are much more serious than suggested by including them among the candidates for alternative text. Applets and programmatic objects, frames, scripts, and multimedia will be covered elsewhere in this book.

There are also duplications in the WCAG list of nontext elements. Images, graphical representations of text, animations, images used as list bullets, and spacers are all instances of uses of the `img` element. The classification of nontext elements I'll use in this chapter is discussed in the upcoming "Classification of Images" section.

Section 508

The first provision of the Section 508 Standards for web content accessibility is very similar to the WCAG, but without the "This includes" remarks.

> §1194.22(a) A text equivalent for every non-text element shall be provided (e.g., via `"alt"`, `"longdesc"`, or in element content).

The guide for this provision (`www.access-board.gov/sec508/guide/1194.22.htm#(a)`) includes mention of most of the examples included in WCAG Checkpoint 1.1. The Access Board guide also mentions "check boxes" in the discussion of nontext elements, which, I think, is an error.

WCAG 2.0

The WCAG 2.0 success criteria for text equivalents (version as of Last Call) help us understand how to write alt-text. Here is the main guideline and corresponding success criterion (SC):

> Guideline 1.1 Provide text alternatives for all non-text content.
>
> SC 1.1.1 For all non-text content, one of the following is true:
>
> - If non-text content presents information or responds to user input, text alternatives serve the same purpose and present the same information as the non-text content. If text alternatives cannot serve the same purpose, then text alternatives at least identify the purpose of the non-text content.
> - If non-text content is multimedia; live audio-only or live video-only content; a test or exercise that must use a particular sense; or is primarily intended to create a specific sensory experience; then text alternatives at least identify the non-text content with a descriptive text label. (For multimedia, see also, Guideline 1.2 Provide synchronized alternatives for multimedia.)

- If the purpose of non-text content is to confirm that content is being operated by a person rather than a computer, different forms are provided to accommodate multiple disabilities.
- If non-text content is pure decoration, or used only for visual formatting, or if it is not presented to users, it is implemented such that it can be ignored by assistive technology.

Note that the success criteria are numbered with the guideline and an index. So, SC 1.1.1 is the first success criterion for Guideline 1.1.

It is refreshing that these success criteria do a very good job in elaborating on and explaining how Section 508 §1194.22(a) and WCAG 1.0 Checkpoint 1.1 should be handled. SC 1.1.1 covers information-bearing images, for which the alt-text should convey the same information as the image. Here is a sample from `www.borders.com`.

But image buttons, image links, and image map hotspots are also covered by SC 1.1.1. If the image has text like About us or Go, then the alt-text should be the same. If the image is a question mark that opens a help screen, then the alt-text should be "help" (the purpose of the image link). WCAG 2.0 recognizes that there will be places where it does not make sense to try to provide a text equivalent, and SC 1.1.1 says to identify the purpose of that nontext content. The last clause of SC 1.1.1 covers formatting or spacer images, where `alt=""` is appropriate for XHTML.

Classification of Images

Most of the pictures we see on web pages are images, including animated GIFs. There are also images we do not see, so-called *spacer* images, which may be used for formatting purposes when tables are used for layout.

Text equivalents for images are provided through the `alt` attribute, which is a required attribute on the `<img>` tag. For example, consider the following image link:

Here is the code for this link:

```
<a href=". . .><img src= "go.gif" width="21" height="21" alt= "go" />
</a>
```

Every `img` element must have alternative text specified in the `alt` attribute. That text is referred to as *alt-text*. As specified in SC1.1.1, alt-text should convey the information in the image or the purpose (function) of the image.

Let's look at the different types of images.

Image Links

An image link is an image inside an anchor tag, <a>, like the following simple example of a logo at the top of a commercial page (www.corda.com).

Here is the code for this image link:

```
<a href="http://www.corda.com/" >
  <img src="images/logobutton2.gif" alt="Home"  ... />
</a>
```

In an earlier version of the page, the Corda logo had alt="Corda Logo", which I accepted without comment, but that alt-text is not really correct. The alt-text should specify the *function* of the link, and "Corda logo" is not the function of the link. It could be alt="Home page" or possibly alt="Corda home page". But "page" is redundant (we don't say "contact us page"), as is "Corda", because what other homepage would it be? And we just heard "Corda" from the page title. The bottom line is that alt="Home" is the correct choice for the alt-text.

Image Map Hotspots

A client-side image map is an image (img element) with the usemap attribute whose value is the name of a map element. Here is an example of a client-side image map with four hotspots:

The following is the code for this example:

```
<img src="banner.gif" width="500" alt="navigation banner"
     usemap="#banner" />
<map name="banner" id="banner">
   <area href="home.htm" alt="homepage"
         shape="rect" coords="0,0 110,24" />
   <area href="services.htm" alt="services"
         shape="rect" coords="111,0 215,24" />
   <area href="comntact.htm" alt="contact us"
         shape="rect" coords="216,0 326,24" />
   <area href="search.htm" alt="search"
         shape="rect" coords="327,0 435,24" />
</map>
```

Image Buttons

An image button is an `input` element with `type="image"`. The following shows a typical image button.

The source for the image is specified with the `src` attribute of the `input` element:

```
<input type="image" src= "llogon.gif" alt="log on" />
```

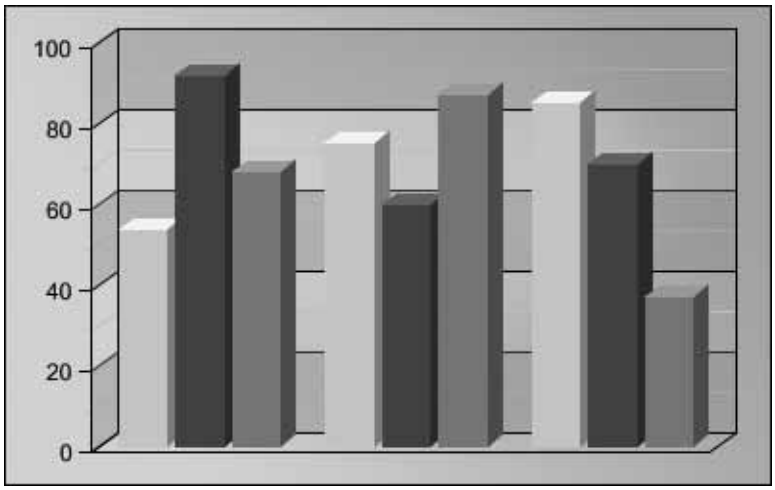
Decorative and Formatting Images

Some images are primarily for visual appeal, serving as decoration, or sometimes they carry absolutely no information, as is the case with spacer images used for formatting. For this type of image, `alt=""` is appropriate:

```
<img src="images/1-pix.gif" width="1" height="1" alt="" />
```


Charts and Graphs

Some images carry a lot of information—more than can reasonably be placed in alt-text—and they require special accommodation. Here is an example:



The following is the code used for the sample chart image:

```
<img src="images/bargraph.gif" width="400" height="250"
          alt="bar graph of results for three groups"
          longdesc="bargraph.htm" />
```

Accessible Image Links

Images that play some active role in the navigation of the site must have meaningful and simple alt-text. Without that text, users with disabilities will not be able to navigate or otherwise interact with your site. Since a user who is blind will be listening to the links, it is very important to make those alternative text strings simple and short while still clear.

Typical of this kind of image are the ones that look like buttons for navigation. Here is an example:

This image typifies groups of images that look like the horizontal tabs on file folders. Here is what the code for this image *should* look like:

```
<img src="video-tab.gif" width="39" ...alt="Electronics" />
```

I say "should" because this simple image actually does not have any alternative text at all on the site where it was found. Since there is no alt-text on the image on the subject site, a screen reader must try to hunt for some information about the image elsewhere in order to try to convey the target of the link.

When there is no alt-text, different assistive technologies look in different places: sometimes the `href` attribute of the anchor tag (`<a>`), and sometimes the `src` or the `name` attribute.

Here is the `href` attribute (in elided form) for the link in this case:

```
href="exec/obidos/tg/browse/-/172282/ref=tab_gw_e_4/104-9411283-476802"
```

Because values of `href` attributes are often very long and complicated (as in this case), with many initial segments that are redundant and do not carry information, IBM Home Page Reader tries to pick a part of the `href` that is shorter and will hopefully carry some information to the user. Here is the result in this case:

- **Home Page Reader:** "ref equal tab underscore g w underscore e underscore 4 divided by 104 minus 9411283 minus 476802"

I have spelled out the way Home Page Reader reads this `href` to underscore how ugly it is. The numbers are also spoken as numbers: "nine million, four hundred and eleven thousand, two hundred and eighty three." This is not only meaningless, it is painful. The visually impaired Home Page Reader user will not be able to participate in this navigation.

Hal users are better off because they only have to listen to the numbers!

- **Hal:** "104 minus 9411283 minus 476802 not visited link press insert to click"

Things are different for the Window-Eyes user, but they are not better. They get to hear the whole `href`, without punctuation and reading only digits instead of spelled-out numbers.

- **Window-Eyes:** "link exec obidos tg browse 172282 ref = tab g w e 4 104 9411283 476802"

JAWS for Windows takes a different approach, which, in this case, is more helpful. It tries to read a part of the `src` attribute of the image. Here is the `src` attribute (slightly elided):

```
src="http://.../G/01/nav/personalized/tabs/electronics-off-sliced.gif"
```

The part that JAWS reads contains valuable information but it is cluttered with meaningless stuff; however, the keyword, *electronics*, is present!

- **JAWS:** "Link graphic tabs slash electronics dash off dash sliced"

The fact that there is anything good about what JAWS speaks is the result of fortune rather than foresight, and it is still far from ideal! Accessibility should never rely on the use of comprehensible filenames in your web page (not that they are a bad idea, but clear and simple alt-text is the real answer). Because this page does not have alt-text where it needs it, the link becomes completely incomprehensible for the Home Page Reader, Hal, and Window-Eyes users, and even JAWS users are left guessing.

After pulling this terribly inaccessible example into pieces, let's look at a simple example of a nicely accessible image link. This is a common button-like navigation link (on `www.evite.com`), which is basically a picture of text tailored and refined by graphic designers and then sealed in pixels.

All of the images of text raise a serious problem. If an Internet Explorer user wants to adjust text size for more comfortable viewing (View ➤ Text size ➤ Largest)*, it will have no effect on these images. Opera's text enlargement* (View ➤ Zoom ➤ 200%) *does work with images. When you use images of text, be sure to use large fonts and adequate contrast. By the way, the sample image link here fails all the measures of contrast; see the "Color Contrast" section later in this chapter.*

This image link is correctly coded with the alt-text, `"Planning Ideas"`, which matches the text in the image and the function of the image link.

```
<a href="/app/home/viewPlanningHome.do">
   <img id="plan" src="/images/nav/orng.gif" ...alt="Planning Ideas" />
</a>
```

Each assistive technology handles this link in pretty much the same way—in the way that the web author intended and the way that the user can understand. Here are the results:

- **Hal:** "Planning Ideas, not visited link, press insert to click"
- **Home Page Reader:** "[Planning Ideas]"
- **Window-Eyes:** "Link Planning Ideas"
- **JAWS:** "Link Graphic Planning Ideas"

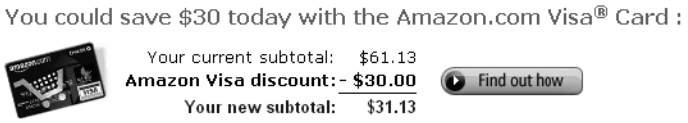
The fact that an item is a link is indicated with Home Page Reader speaking in a female voice (with default settings). In the text view, Home Page Reader displays alt-text in square brackets. Hal, Window-Eyes, and JAWS all speak some variation of the word "link" together with the alt-text, `Planning Ideas`.

This is why alt-text for images is so very important: because the necessary information is quickly and clearly conveyed to the user. It is also important for website owners, because without it, visitors who use screen readers or talking browsers will not be able to use or even understand their site.

The Role of Positional Information or Context

Often, context provides information that is not available to a screen reader user listening to links. (This topic is also addressed in the discussion of link text in Chapter 7 and the discussion of forms in Chapter 8.)

Consider the following part of a checkout procedure at Amazon.com.



The natural alt-text for the image link is `alt="find out how"` (the fact that Amazon has no alt-text on this image is not the point). If that were the case, when blind users listened to links, all they would hear is "Find out how." How to do what? This is a case where the alt-text on the image can easily and conveniently solve this context problem: `alt="Find out how to save $30 with the Amazon Visa Card"`.

Accessible Image Map Hotspots

The importance of text equivalents for image map hotspots is just as great as that for image links. The difference is the form that these client-side image maps take.

An Accessible Client-Side Image Map

Let's begin with an accessible client-side image map instead of an inaccessible one—it's more fun that way. The following is an image from the University of Arizona Web Resources Page (as it was in 2001).

This image is specified in the HTML code as an `img` element with the `usemap` attribute, as follows:

```
<img src=" . . ." alt="Web Resources at The University of Arizona" ...
          usemap="#banner" />
```

Notice that the image does have an `alt` attribute, but the image itself is not the important accessibility issue. In the University of Arizona website case, the image has alt-text that corresponds to the text on the image, which is good, in principle. But that text is repeated as image map hotspots. I tend towards the idea of using `alt=""` on the image itself, since it is not active and conveys no information beyond the hotspots.

Then the `<map>` tag with `name="banner"` specifies areas of the image that will be hotspots. In this example, all the areas are simple rectangles (`shape="rect"`):

```
<map name="banner" id="banner">
   <area shape="rect" alt="UA Web Resources Homepage"
         coords="10,05 90,75" href=" ..." />
   <area shape="rect" alt="UA Homepage" coords="95,05 360,35"
         href=" ..."  />
   <area shape="rect" alt="Search" coords="260,52 305,70"
         href=" ..." />
   <area shape="rect" alt="Comments & questions" coords="311,52 37,70"
         href=" ..." />
   <area shape="rect" alt="Web Resources Homepage"
         coords="385,52 502,70" href=" ..." />
   <area shape="rect" alt="UA Homepage" coords="515,52 572,70"
         href=" ..." />
</map>
```

> *A* `map` *element can be placed anywhere on the page. Its position is irrelevant to the position of the image, but the hotspot links will appear in the reading order where the* `area` *elements appear.*

The six hotspot rectangles specified by the `area` elements in the map surround the text headings in the image. Each rectangle is given by x/y pixel coordinates, where (0,0) is the top left and, in this case, (588, 81) is the bottom right.

If you had visited the University of Arizona site (the image map is no longer there) or if you visit any site with image maps using a traditional browser like Internet Explorer or Netscape, you may be able to see the `href` of the different areas in the browser status window as you move the mouse around the image. The tooltip also changes to the alt-text of the `area` element. In parts of the image that are not designated as hotspots, the status window goes blank and the tooltip changes to the alt-text of the image. (As discussed in Chapter 7, much more complicated client-side maps are possible, such as a map of the U.S.)

Assistive technology can cope with a correctly implemented image map just as well as it does with normal links (text or image). Home Page Reader does an even better job because only the number of links in the map is announced initially, so as not to burden the listener when the screen reader is just reading the page. If you want to hear the actual hotspots, you just stop and step through them with the arrow keys. This is the way they sound:

- **Home Page Reader:**
  - "(Start of map with 6 items.)"
  - "[UA Web Resources Homepage]"
  - "[UA Homepage]"
  - "[Search]"
  - "[Comments & Questions]"
  - "[Web Resources Homepage]"
  - "[UA Homepage]"
  - "(End of Map.)"

For sighted people using Home Page Reader as an analytical tool to check their websites for accessibility, it is convenient to see all *the text, including the links within image maps. You can do this by moving the focus to the text area and pressing Ctrl+A to highlight the entire text view. Home Page Reader will then load all of the text and highlight it. If you want to make a copy, you can press Ctrl+C to copy the entire text view to the clipboard, and then press Ctrl+V to paste it into another application. That is what I did here in order to display the links corresponding to the areas of the sample image map shown here.*

Using Home Page Reader, the value of the `alt` attribute is always displayed within square brackets, [ ]. So-called *metatext*, which is text that is generated by Home Page Reader and does not appear on the page, is enclosed in parentheses, as in "(End of Map.)".

Window-Eyes, Hal, and JAWS all treat image map links with alt-text just like any other kind of link. This makes sense because, in function, image map links are no different from any other kind of link, and so they should not be treated differently when spoken.

Style of Alternative Text

The choice of specific alternative text is a matter of style and judgment. For the six areas of the client-side image map shown in the previous section, I would use alt-text that is the same as that on the image, except where it makes sense to make it *shorter*. Here is a comparison of the alt-text that the developers at the University of Arizona used and what I would recommend instead:

Text in Image	Actual Alt-Text	Suggested Alt-Text
Web Resources	UA Web Resources Homepage	Web resources
The University of Arizona	UA Homepage	U of A
Search	Search	Search
Comments	Comments & questions	Comments
Web Resources Home	Web Resources Home	Web Resources
UA Home	UA Home	U of A

The reason I prefer shorter alt-text is that people using a screen reader must sit and listen to all this. When sighted people look at the text on an image, they have the ability to filter out the "visual noise" and focus on the main concept. It is true that this is still possible when you have to listen to the content, but it is more difficult.

I tend to avoid "homepage" when identifying links. If the link says "The University or Arizona" or "IBM," I know it is going to the respective homepage, so "homepage" is redundant.

Bearing in mind the importance of conciseness when considering alt-text, it is not surprising that my preference is "U of A" over "University of Arizona." But why, you may ask, "U of A" over "UA"? That comes from my experience with screen readers occasionally *pronouncing* abbreviations like UA (or IBM for that matter) instead of spelling them out. If "UA" is pronounced, it truly fails in conveying the desired information. In this case, it turns out that all of the assistive technologies discussed here spelled out "UA" rather than pronouncing it, probably because it is capitalized. However, since there are other voice-assistive technologies available, it is best to adopt a safe approach.

Sometimes web designers prefer to see longer mouse-over tooltip text than the alt-text so that the mouse user gets more information than that on the image. To do this, you keep the alt-text the same as the image text and put the longer text in the `title` attribute of the anchor element. With Internet Explorer, this trick works for links, but it does not work for the `area` elements. Only Internet Explorer displays the alt-text as a tooltip. Firefox and Opera display the `title` attribute; Opera adds the prefix Title:.

An Inaccessible Client-Side Image Map

As an example of an inaccessible image map, consider this Yahoo! banner image from late 2001.

> *You may be interested in how to find such older pages. Try the Wayback Machine (*`www.archive.org`*), where you can get archived copies of most sites. Unfortunately, the site seems to have stopped saving pages in mid 2005, but there is still an incredible amount of history available—approximately 40 billion pages!*

This banner image looks like it contains text—Finance, Messenger, Check Email, What's New, Personalize, and Help—but it doesn't. The text under each button-looking image is made to look like link text, but it is really part of the image. For mouse users, the text works like a link because the banner is an image map. Had the Yahoo! developers decided to add alt-text to the `area` elements as follows, the image map would work just like a list of text links for a blind user as well (in fact, that is exactly what they did in mid 2002).

```
<map name=m>
   <area coords="0,0,52,52" href=r/a1 alt="Finance" />
   <area coords="53,0,121,52" href=r/p1 alt="Messenger" />
   <area coords="122,0,191,52" href=r/m1 alt="Check Email" />
   <area coords="441,0,510,52" href=r/wn alt="What's New" />
   <area coords="511,0,579,52" href=r/i1 alt="Personalize" />
   <area coords="580,0,637,52" href=r/hw alt="Help" />
</map>
```

The Yahoo! page was designed with efficiency in mind. The web designers tried to minimize the download time by reducing the number of characters on the page; however, the additional text that makes the navigation banner solidly accessible amounts to only 186 bytes—less than 0.6 percent of the size of the page—and increases the download time by only about 0.05 seconds, even on a slow modem.

Without the alt-text, all the assistive technologies were stuck with the meaningless `href` information like `r/11`, `r/p1`, and so on. To make matters worse in this case, Hal adds the full URL `http://www.yahoo.com` in front of the `href`. But that has all passed; Yahoo! did it right eventually, and currently the site doesn't even use an image map for the banner.

(End of Excerpt)

9 CSS FOR ACCESSIBLE WEB PAGES

by Richard Rutter

Cascading Style Sheets (CSS) is a style sheet language that allows authors and users to attach styles to structured documents like HTML pages. Style sheets were designed to allow precise control of character spacing, text alignment, object position on the page, font characteristics, color, backgrounds, and so on.

When used to their greatest advantage, style sheets are defined in files completely separate from the HTML documents. By separating style from markup, authors can simplify and clean up the HTML in their documents. This means site-wide changes can be made in just one place, which should decrease maintenance and increase visual consistency across a website.

This chapter discusses the accessibility features of CSS and provides explanations and examples of how to use them. CSS benefits accessibility primarily by separating document structure from presentation, and we'll begin by looking at some examples of how that works.

How Style Sheets Benefit Accessibility

The CSS Zen Garden (`www.csszengarden.com/`) is a website designed to demonstrate the capabilities of CSS. It has a single HTML document for which hundreds of designers have created style sheets that radically change the layout and presentation. Figures 9-1 through 9-4 show some examples (remember the underlying HTML is identical in each).

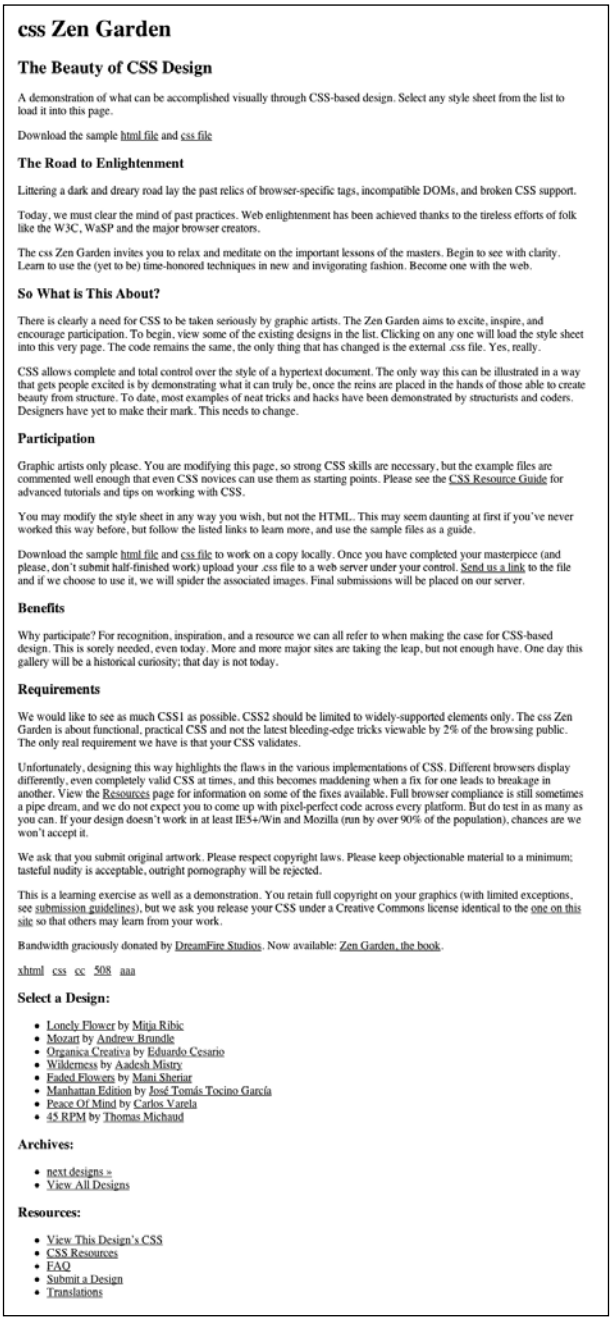
Figure 9-1. CSS Zen Garden design "Organica Creativa" by Eduardo Cesario

Figure 9-2. CSS Zen Garden design "Pinups" by Emiliano Pennisi

Figure 9-3. CSS Zen Garden design "Obsequience" by Pierce Gleeson

Figure 9-4. CSS Zen Garden design "Invasion of the Body Switchers" by Andy Clarke

Without a style sheet attached, the CSS Zen Garden HTML page looks like Figure 9-5.



css Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.

Download the sample html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

So What is This About?

There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The code remains the same, the only thing that has changed is the external .css file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. To date, most examples of neat tricks and hacks have been demonstrated by structurists and coders. Designers have yet to make their mark. This needs to change.

Participation

Graphic artists only please. You are modifying this page, so strong CSS skills are necessary, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.

You may modify the style sheet in any way you wish, but not the HTML. This may seem daunting at first if you've never worked this way before, but follow the listed links to learn more, and use the sample files as a guide.

Download the sample html file and css file to work on a copy locally. Once you have completed your masterpiece (and please, don't submit half-finished work) upload your .css file to a web server under your control. Send us a link to the file and if we choose to use it, we will spider the associated images. Final submissions will be placed on our server.

Benefits

Why participate? For recognition, inspiration, and a resource we can all refer to when making the case for CSS-based design. This is sorely needed, even today. More and more major sites are taking the leap, but not enough have. One day this gallery will be a historical curiosity; that day is not today.

Requirements

We would like to see as much CSS1 as possible. CSS2 should be limited to widely-supported elements only. The css Zen Garden is about functional, practical CSS and not the latest bleeding-edge tricks viewable by 2% of the browsing public. The only real requirement we have is that your CSS validates.

Unfortunately, designing this way highlights the flaws in the various implementations of CSS. Different browsers display differently, even completely valid CSS at times, and this becomes maddening when a fix for one leads to breakage in another. View the Resources page for information on some of the fixes available. Full browser compliance is still sometimes a pipe dream, and we do not expect you to come up with pixel-perfect code across every platform. But do test in as many as you can. If your design doesn't work in at least IE5+/Win and Mozilla (run by over 90% of the population), chances are we won't accept it.

We ask that you submit original artwork. Please respect copyright laws. Please keep objectionable material to a minimum; tasteful nudity is acceptable, outright pornography will be rejected.

This is a learning exercise as well as a demonstration. You retain full copyright on your graphics (with limited exceptions, see submission guidelines), but we ask you release your CSS under a Creative Commons license identical to the one on this site so that others may learn from your work.

Bandwidth graciously donated by DreamFire Studios. Now available: Zen Garden, the book.

xhtml css cc 508 aaa

Select a Design:

- Lonely Flower by Mitja Ribic
- Mozart by Andrew Brundle
- Organica Creativa by Eduardo Cesario
- Wilderness by Aadesh Mistry
- Faded Flowers by Mani Sheriar
- Manhattan Edition by José Tomás Tocino García
- Peace Of Mind by Carlos Varela
- 45 RPM by Thomas Michaud

Archives:

- next designs »
- View All Designs

Resources:

- View This Design's CSS
- CSS Resources
- FAQ
- Submit a Design
- Translations

Figure 9-5. CSS Zen Garden HTML without a style sheet attached

So, you can see that CSS is extremely powerful. It can provide layout and presentation entirely independent of the HTML document, which is extremely important. If you look at the unstyled version of the CSS Zen Garden page (Figure 9-5), it doesn't look very pretty, but it still makes sense. You can still scan the headings, read the text, and see the lists. This is because the underlying content is in a logical order and marked up with semantic—that is to say, structural and meaningful—HTML.

In the U.S. Access Board's Section 508 Standards, the applicable provision is as follows:

> §1194.22(d) Documents shall be organized so they are readable without requiring an associated style sheet.

Similarly, the Web Content Accessibility Guidelines (WCAG) 1.0 says this:

> 6.1 Organize documents so they may be read without style sheets. For example, when an HTML document is rendered without associated style sheets, it must still be possible to read the document.

To accomplish this, you can think of CSS as a presentation "layer" that you can remove and still be left with readable and understandable content. By moving presentational markup into style sheets, HTML documents are left with clean and semantic markup. This more readily enables people to use a website with devices such as text-only browsers, aural browsers, and screen readers.

CSS can provide a benefit only when used with semantic HTML. You should build the unstyled web page first, paying attention to the order and structure of the pages. As discussed in Chapter 6, remember to mark up all your headings with `<hn>` elements; all your lists using `<dl>`, `<ul>`, or `<ol>` tags; and so on. You can also wrap related content (such as sidebars) inside `<div>` elements.

In this chapter, you'll learn how to use various CSS techniques to improve the accessibility of a website. But first, let's review some CSS basics.

CSS Basics

Although many designers are familiar with the basic syntax used in CSS, this section provides a brief overview, with an emphasis on best practices for accessibility. If you need an in-depth guide to CSS, refer to a book dedicated to that subject, such as *CSS Mastery: Advanced Web Standards Solutions* by Andy Budd (1-59059-614-5; friends of ED, 2006).

Style sheets are made up of rules that apply to an HTML document (although style sheets can also be applied to XML documents, this chapter will cover HTML only). Style sheet rules are made up of three parts:

- The elements the rule applies to (called the *selector*)
- The property that is being set
- The value of the property

An example of a style sheet rule is shown in Figure 9-6.

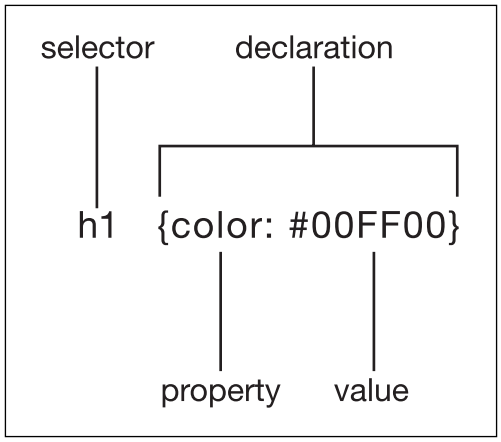


Figure 9-6. An example of a style sheet rule

In the example in Figure 9-6, the selector is `h1`, the property is `color`, and the value is `#00FF00` (the hexadecimal value for pure green). Together, the property and value form the declaration part of the rule. The selector links the style sheet rule to the HTML. In this case, all `h1` elements will be shown in this shade of green.

More than one declaration can be applied to a selector, and more than one selector can be given the same declaration, as in this example:

```
cite, blockquote {font-style:italic; font-family:'Myriad Pro',
Helvetica, Arial, sans-serif}
```

Here, all `cite` and `blockquote` elements will be displayed in italic Myriad Pro (or Helvetica or Arial, depending on which fonts are available on the user's computer).

Style sheet rules can be applied to HTML documents in three ways:

- **Inline:** Declarations can be attached inline to individual elements using the `style` attribute.
- **Embedded:** The rules can be embedded into the HTML document using the `<style>` element.
- **Linked:** The CSS rules can be written to a separate file, usually with a `.css` extension, and linked to the HTML document using the `<link>` element or an `@import` rule.

Let's look at when it's appropriate to use each of these methods, and then at when user and browser style sheets are applied.

Inline Styles

Inline styles should be used only when the designer needs to specify a particular style in one place in a single document. Here is an example:

```
<p style="border:1px solid purple ">
```

In this piece of code, content (the <p> element) and presentation (the `style` attribute) are both present. The implication is that inline styles do not separate presentation from content. Therefore, you should use inline styles only in exceptional circumstances.

Even if you require a single element in a document to be given a style, there is an alternative to using an inline style: using an `id` selector. Give the element a unique `id` attribute, as in this example:

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

You can then use the `id` attribute as a selector in your style sheet, like this:

```
#special { border:1px solid purple }
```

If you find yourself repeating the same inline style several times within a document, you should use a `class` selector. This works in a similar way to the `id` selector, except it uses a `class` attribute, like this:

```
<p class="special">
```

Use the `class` selector in your style sheet like this:

```
.special { border:1px solid red }
```


The main difference between `id` selectors and `class` selectors is that an `id` can be applied only to a single element within an HTML document, whereas `class` selectors can be applied to as many different elements as you like, although you should limit the use of `id` and `class` selectors where possible. One further difference between `class` and `id` selectors—from a CSS perspective—is that `id` selectors are considered to have a higher *specificity*. An `id` is considered more important than a class, so where rules contradict each other, the `id` will override the `class`. For example, consider this HTML code:

```
<p id="veryspecial" class="special">
```

And the corresponding CSS rules:

```
#veryspecial {background: purple}
.special {background: red; font-weight: bold}
```

The paragraph will be displayed in bold, as specified in the `class` rule. The `class` rule also specifies a red background, but the `id` rule says that the background should be purple. As `id` selectors are more specific than `class` selectors, the `id` overrides the `class`, and the background of the paragraph will be displayed in purple.

Embedded Style Sheets

With the embedded method of adding CSS to an HTML document, the CSS rules are enclosed in a <style> element, which should be included inside the <head> element. Valid HTML requires that you should always include the type attribute with the <style> element. Here is an example:

```
<style type="text/css">
h1 {
color: #00FF00;
}
#veryspecial {
background: purple;
}
.special {
background: red;
font-weight: bold;
}
cite, blockquote {
font-style:italic;
font-family:'Myriad Pro', Helvetica, Arial, sans-serif;
}
</style>
```

Embedding style sheets in this manner goes one step further in separating content from presentation, as all the style rules are containing within a single element and not intermingled with the HTML. However, embedded style sheets can apply rules to only a single HTML document, so in a site-wide context, the styles are still not very separated from the content. The implication is that embedded style sheets should be used only when rules need to be applied to a single page within a website. For styles that need to be applied to more than one web page (as is often the case), a linked style sheet should be used.

Linked and Imported Style Sheets

By importing or linking to an external style sheet file, the designer can control the presentation of a whole website from one or more style sheets. A style sheet file would normally be named with a .css extension and would include style sheet rules like this:

```
h1 {
color: #00FF00;
}
#veryspecial {
background: purple;
}
.special {
background: red;
font-weight: bold;
}
```

```
cite, blockquote {
font-style:italic;
font-family:'Myriad Pro', Helvetica, Arial, sans-serif;
}
```

Notice the rules are exactly the same as with the embedded style sheet, but there is no need for the <style> element. The style sheet file is linked to an HTML page using the <link> element in the <head> of the document, like this:

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

An alternative method is to use @import from within a <style> element, like this:

```
<style type="text/css">
@import url(mystyles.css);
</style>
```

The <link> element is supported by all browsers that support style sheets. However, the @import method is not supported by some older browsers, including Internet Explorer 4 and Netscape Navigator 4 and older. This is often used to the designer's advantage, because the older browsers have very poor support for CSS. Linking to style sheets using the @import method means only capable browsers will be able to use your style sheets. The older browsers will display the unstyled pages, which should still be usable, assuming you have used HTML correctly and marked up the content in a meaningful way. Hence, @import is usually the method of choice for linking to style sheets.

User Style Sheets

So far, I have discussed the use of style sheets created by the author. However, one of the major accessibility features of CSS is that style sheets can be created by users, too. User style sheets are files created by users and stored on their computer. The browser is then configured to apply the user style sheet to all websites that the user visits.

For example, a visually impaired reader may specify a style sheet like this:

```
body {
  color: yellow;
  background: black;
}
```

In this style sheet, the user has set a black background and yellow text for the web page (perhaps because the combination of color makes the text easier to read). The rules in the user style sheet will override rules of an equivalent specificity set by the author. However, if the author sets a more specific rule like this:

```
p.special {background: green;}
```

Then the author's rule will override the user's rule. In this case, the result will be yellow text displayed on a green background for all paragraphs with a class of special, which is clearly undesirable and not at all helpful for the user.

In order to ensure that users can control styles as they wish, CSS2 defines the `!important` operator. If a user's style sheet contains `!important`, it takes precedence over any applicable rule in an author's style sheet. For instance, the rule in the following user style sheet will ensure that every paragraph on the web page is set to the desired colors:

```
p {
  color: yellow ! important ;
  background: black ! important ;
}
```

Authors can also use `!important` in their style sheets, but it will always be overridden by a user style sheet (although this is only true in Internet Explorer if the user-defined style sheet also contains the `!important` keyword).

The CSS2 `inherit` value, which is available for most properties, enables `!important` style rules that can govern most or all of a document. For instance, the following rules force *all* backgrounds to black and *all* foreground colors to yellow:

```
body {
  color: black ! important ;
  background: white ! important ;
}

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

The first rule sets the colors of the <body>, and the second rule uses the * selector to target all elements on the page and the `inherit` property to inherit the colors from the <body>.

CSS2 also enables further control over the display, including these features:

- System colors (for the `color`, `background-color`, `border-color`, and `outline-color` properties) and system fonts (for the `font` property) mean that users may apply their system color and font preferences to web pages.
- Dynamic outlines (using the `outline` property) allow users to create outlines around content in order to highlight certain information.

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

```
:focus  { outline: thick solid blue }
:active { outline: thick solid green }
```

These style rules could help some readers with visual difficulties more readily see which element has focus (in a form, for example).

Browser Style Sheets

Even when no user or author style sheet has been specified, browsers still use a style sheet to render HTML with default styling. For example, Firefox uses a file called `html.css` for the default rendering of HTML. Here is an extract showing how Firefox styles level 1 headings:

```
h1 {
  display: block;
  font-size: 2em;
  font-weight: bold;
  margin: .67em 0;
}
```

Browser style sheets have a lower specificity than author and user style sheets.

Color and Backgrounds

CSS allows users with particular presentation requirements to override author styles. This is very important to users who have problems with certain colors and fonts, as CSS allows users to view documents with their own preferences by specifying them in a user style sheet.

Colors are probably the easiest properties to set using CSS, but they can also be the most problematic. In this section, I will highlight the potential problems and provide solutions to specifying color with CSS.

Background and Text Colors

As explained in the "User Style Sheets" section, it is possible for author style sheets and user style sheets to clash. The example I provided had a user specifying yellow text on a black background, with the author using a more specific rule to set the background to green. This resulted in the user seeing green text on a yellow background, which is difficult to read. It is therefore vital that authors specify both a foreground color and a background color so that text will always remain readable.

Color problems can also arise with certain operating system "themes." For example, a designer may want all the form controls to appear with a gray background:

```
input, textarea {background: gray}
```

In most situations, this would be fine. However, form control colors tend to be inherited from the operating system, so an operating system theme with gray text might exist, meaning that the text on the form controls would be invisible. To solve this problem, the designer should specify the foreground color of the form controls as well:

```
input, textarea {
  background: gray;
  color: black;
}
```

Background Images

CSS also allows designers to set background images. However, it's important to remember that if you do this, you should always set a background color as well:

```
P .special {
  background: red url(bricks.gif);
  color: white;
}
```

By choosing a suitable background color, you can ensure that the text is always readable, regardless of browser settings or any bandwidth problems delaying the downloading of images.

Foreground and Background Contrast

People with low vision often have difficulty reading text that does not contrast enough with its background. This can be exacerbated if the person has a color vision deficiency that lowers the contrast even further. Another group of users affected by low contrast are people using monitors suffering from reflections (perhaps from a window) or screens that have poor contrast. So, following accessibility advice could also benefit people without disabilities.

Designers should ensure that foreground and background colors contrast well. On this subject, WCAG 1.0 states

> 2.2 Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen. [Priority 2 for images, Priority 3 for text].

A very basic test for determining whether color contrast is sufficient to be read by people with color vision deficiencies or by those with low-resolution monitors is to print pages on a black-and-white printer (with backgrounds and colors appearing in grayscale). The important thing to remember is to rely on lightness differences between foreground and background, not to rely on color differences.

For more specific guidance on color contrast, see "Effective Color Contrast" by Aries Arditi, Lighthouse International (`www.lighthouse.org/color_contrast.htm`), and "Type and Colour" by Joe Clark (`joeclark.org/book/sashay/serialization/Chapter09.html`). Jonathan Snook has developed an online Color Contrast Checker at `www.snook.ca/technical/colour_contrast/colour.html`.

Other Means of Conveying Information

I have just stressed the importance of color contrast. It can be inferred that relying on color *alone* is not a reliable method of conveying information and meaning. Indeed, Section 508 states

> §1194.22(c) Web pages shall be designed so that all information conveyed with color is also available without color, for example from context or markup.

As a designer, you should ensure that information and meaning is also conveyed through means that do not depend on the user's ability to differentiate colors. For example, when asking for input from users, do not write "Please select an item from those listed in green." Similarly, do not turn off the underlining of links without providing some other method of visually determining a link. Instead, make sure that information and meaning are available through other style effects, such as bold text or grouped under a heading.

This does not mean that you should not use color to enhance identification of important features. Indeed, color is an extremely important weapon in the designer's arsenal. It does, however, require that some other method of identification, such as text labels, be combined with the use of color.

Text and Fonts

CSS has properties to control font appearance, so authors can avoid using images of text. Using style sheets to style text, rather than creating images of text, makes textual information and meaning far more accessible. Text as text (as opposed to images of text) makes content available to people who use speech synthesizers and Braille displays. It can also more readily be resized and reformatted for visually impaired users or those using alternative devices, such as handhelds and projectors. Another benefit is that the text can be read by other software, such as search engine robots.

In the past, HTML markup was heavily abused and the accessibility implications were ignored in the pursuit of presentational effects. CSS provides many properties for styling text beyond just size, and this opens the door to enabling well laid out pages combined with meaningful, accessible HTML.

Text Sizing

Text size is probably the first font style a designer will need to set. In CSS, this is accomplished with the `font-size` property:

```
P {font-size:12px}
```

This sets all paragraphs to display at a font size of 12 pixels, what the designer may consider the perfect size for paragraph text. However, not all users will agree with that design decision—perhaps they are short-sighted, doing a presentation, using a very-high resolution screen, or simply have tired eyes. The beauty of CSS (and the Web in general) is that users can change settings in their browser to make all the page text proportionately bigger or smaller as they require.

> *As a designer, it is important to create web pages that will allow text to be resized without the layout breaking or the design being otherwise compromised. Users can, and will, change the size of text to suit their requirements, and this should always be taken into account.*

There is, however, a complication: Internet Explorer for Windows (up to version 6 at the time of writing) will *not* resize text that has been defined in pixels. You could reasonably argue that if users require text to be resized, they should use software that is capable of doing so, such as Firefox and Opera (both of which are free), but not everyone has that choice. Furthermore, WCAG 1.0 has this to say on the subject:

> 3.4 Use relative rather than absolute units in markup language attribute values and style sheet property values. [Priority 2]

In this instance, pixels should be considered absolute units, although they are actually relative to the screen resolution. The alternative to absolute units is relative units, which include percentage, keywords (such as `bigger`), and for text in particular, `ems`. Sizing text using `ems` is slightly more complicated than using pixels, and so warrants further explanation.

The *em* unit of measure is so-called because it approximates the size of an uppercase letter *M*, although 1 em is actually significantly larger than this. In *The Elements of Typographic Style* (0-881-79132-6; Hartley and Marks, 2001), the typographer Robert Bringhurst describes the em as follows:

> The em is a sliding measure. One em is a distance equal to the type size. In 6 point type, an em is 6 points; in 12 point type an em is 12 points and in 60 point type an em is 60 points. Thus a one em space is proportionately the same in any size.

To illustrate this principle in terms of CSS, consider these styles:

```
#box1 {
     font-size: 12px;
     width: 1em;
     height: 1em;
     border:1px solid black;
}
```

```
#box2 {
     font-size: 60px;
     width: 1em;
     height: 1em;
     border: 1px solid black;
}
```

These styles will render as shown in Figure 9-7.

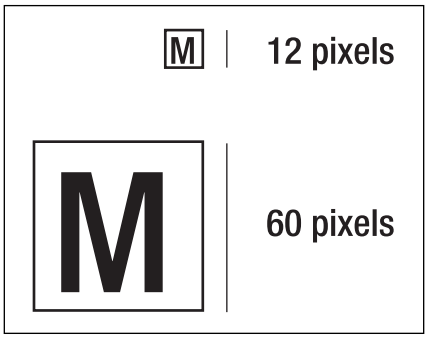


Figure 9-7. Sizing with ems

Note that both boxes have a height and width of 1 em, but because they have different font sizes, one box is bigger than the other. Box 1 has a `font-size` of `12px`, so its width and height are also `12px`. Box 2 has a `font-size` of `60px`, and correspondingly, its width and height are also `60px`.

To explain how sizing text using `ems` actually works, consider this extract from a short web page:

```
<body>
<h1>My heading</h1>
<p>A paragraph of text</p>
</body>
```

In this example, the designer wants the heading to be `24px` and the paragraph to be `12px`. In most browsers, with text set to the default medium size, the font size of the `<body>` will be `16px`. To set the font size of the heading to `24px` using `ems`, you need to apply a little mathematics:

child pixels / parent pixels = child ems

24 / 16 = 1.5

That is to say, the heading is 1.5 times the size of its parent, the body, so you use this rule:

```
h1 {font-size:1.5em}
```

Applying the same logic to change the font size of a paragraph, you have 12 / 16 = 0.75, so you use this rule:

```
p {font-size:0.75em}
```

One added complication is that of nested elements. A list might be added to the page like this:

```
<ul>
  <li>Fruit
    <ul>
      <li>orange</li>
      <li>apple</li>
    </ul>
  </li>
  <li>Vegetable
    <ul>
      <li>potato</li>
      <li>carrot</li>
    </ul>
  </li>
</ul>
```

If you also want the list of items to be 12 pixels, you could add the same style sheet rule that you did for paragraphs:

```
li {font-style:0.75em}
```

This will work for `Fruit` and `Vegetable`, but `orange`, `carrot`, and the other nested list items will be displayed at 9 pixels. Why? Because the rule actually says that any list item should be 0.75 times the size of its parent. So you need another rule to prevent this inherited shrinkage:

```
li li {font-size:1em}
```

This says that any list item inside another list item should be the same size as its parent (the other list item).

For a more detailed explanation of sizing text using ems, see `www.clagnut.com/blog/348/`.

Text Margins and Indentation

Often, the `blockquote` element is misused to indent a block of text. This is an unreliable and sometimes harmful approach. It can be harmful because `blockquote` adds meaning to the text: It tells the browser that the enclosed text has been quoted from a source. If this is not the case, the `blockquote` would be misleading. It can be unreliable because the indentation may not always be applied. Indentation is simply a common style applied by most browsers, but not necessarily all of them.

Excerpt from Chapter 14

RETROFITTING CASE STUDY: REDESIGN OF A UNIVERSITY WEBSITE

by Patrick H. Lauke

My first real contact with accessibility came when I started in my position as Web Editor at the University of Salford in January 2001. As part of the responsibilities of my new job, I started researching some of the legal requirements involving educational websites in the UK, inevitably coming across the Special Educational Needs and Disabilities Act (SENDA) and the wider-reaching Disabilities Discrimination Act (DDA). From there, I delved further into the area of web accessibility. It soon became obvious that the University's site, which had undergone a year-long redesign process just before I started, presented some fundamental accessibility concerns that urgently needed to be addressed.

This chapter covers the process I went through in redesigning the University of Salford's website, with an eye on increased accessibility and the adoption of web standards. These techniques should be of use to you if you're faced with the challenge of retrofitting, or completely rebuilding, a site in order to make it more accessible.

The Original Site

Figure 15-1 shows the original homepage for the University of Salford. Purely from a graphic design point of view, the site was compliant with the University's corporate identity guidelines. However, the additional visual elements of the design—the heavy use of large, rounded corners and the complementary blue/orange color scheme (which, to this day, prompts people to refer to it as the "blueberry-and-custard" site)—were based on the University's recent designs for its print campaign.

The Problems

Subjectively, I would say that the rounded corners, both in the overly large header and footer, were too strong a design element in their own right. Because of their size, they often overpowered the actual content presented on each page. In addition, many people (including myself) often criticized the look of the template for being fairly dated; even in 2001, the "rounded-corners" trend had been gone for a few years.

The design choices for any further visuals on a page were severely restricted. Images with straight edges looked out of place, often requiring them to be "rounded off" first, simply to make them visually fit within the overall feel of the template. An extreme case in point: When asked to include a permanent new feature on the homepage (to advertise the fact that the University had just been awarded a prestigious prize), I ended up having to construct a whole new rounded section as part of the page footer, as shown in Figure 15-2.

Figure 15-1. The University of Salford homepage in 2001

Figure 15-2. More rounded edges to make additional graphics fit with the overall design

As for usability, one of the major pieces of feedback received from users in the weeks following the launch was that the site proved to be very confusing. This sort of problem would have been identified quite early on in usability testing, but unfortunately, no tests were carried out before the launch in December 2000 (apart from a series of focus groups, mainly concentrating on the visual aspects of the site). Because of the drop down–based navigation and the absence of any indication as to where the current page was situated within the context of the overall site (apart from a simple image in the top-right corner with the generic name for the current section), users were simply lost and often forced to return to the homepage just to orient themselves. This was particularly true of visitors coming to the site through a search engine.

Under the hood, the markup was what, even back in 2001, could be described as "old school." Any visual aspects were handled directly in the HTML, with no attempt on the part of the original developers to carry out some basic separation of content and presentation. Here is an excerpt:

```
<TABLE WIDTH="98%" BORDER="0" CELLPADDING="0" CELLSPACING="0">

<TR>
<TD ROWSPAN="4" WIDTH="1%">
<IMG SRC="/images /dot.gif" WIDTH="30" HEIGHT="1" ALT="-" BORDER="0">
</TD>
<TD WIDTH="99%">
<P><FONT FACE="arial,helvetica" SIZE="4" COLOR="#0000CC">
Academic enterprise</FONT></P>
</TD>
<TD ALIGN="RIGHT" VALIGN="TOP" WIDTH="35%"></TD>
<TD ROWSPAN="4" WIDTH="5%"> </TD>
</TR>

<TR>
<TD COLSPAN="2">
<!--MAIN BODY OF PAGE-->
<TABLE WIDTH="100%" BORDER="0" CELLPADDING="0" CELLSPACING="0">
<TR>
<TD VALIGN="TOP"><FONT FACE="arial,helvetica" SIZE="2">
Salford has always had a reputation for enterprising initiatives
... </FONT>
...
```

With no use of Cascading Style Sheets (CSS) in sight, each table cell, paragraph, list item, and so on featured a <font> tag to set the basic typeface, size, and color.

The entire layout was based on a series of convoluted tables, using rowspan/colspan and set cell dimensions (although, to the designers' credit, they did use a combination of pixel and percentage values, making the layout halfway elastic). Due to their construction, these tables also didn't linearize in a sensible way, which I swiftly demonstrated to management by running the site through the Lynx text-only browser. (See Chapter 6 for an explanation of table linearization.)

Plain HTML does not lend itself well to a design based predominantly on rounded corners. To "fake" the rounded-design elements demanded by the University's print campaign, a lot of extra markup had been added to the already complex table layout, as shown in Figure 15-3.

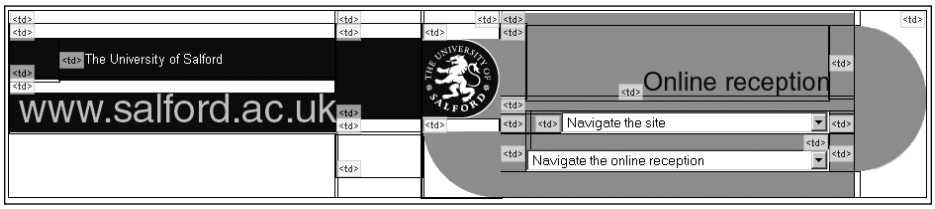


Figure 15-3. Faking rounded corners with tons of table-based markup

Because of the cluttered table markup and the abundance of table cells used purely for spacing and presentation, the resulting code was unnecessarily cumbersome to work with.

This layout was not just used on the University's core site, but also employed across the whole range of departmental sites. With the decentralized web management structure at the University, individual departments have their own web authors looking after their particular site. The experience and skill level of these web authors can vary considerably across the institution, ranging from fairly web-confident technicians to administrative staff and lecturers with little or no knowledge of HTML. Particularly for this latter group, the overly convoluted code of the templates proved difficult to understand. This can often lead to a vicious circle in development. With an increasingly complex maze of markup, web authors who are not too confident in working directly with HTML will rely more and more on the use of what you see is what you get (WYSIWYG) editors. Although these editors hide the complexity of a page's underlying markup, they often introduce even further bloated code, thus compounding the problem. Then, when things go wrong and for some reason a page's layout falls apart at the seams, it's increasingly difficult to fix the problem at the HTML level.

To achieve the rounded-edges look, the original designers used a mixture of table cell backgrounds for solid areas of color and image elements for the corners. This presented an interesting problem when pages were printed.

By default, browsers will print images present in the markup, but omit any background colors (unless users explicitly set this option in their print settings). The result, as indicated in Figure 15-4, was that our pages looked very odd on paper, with a mixture of text content and isolated images of rounded corners set against the white background of the page.

Figure 15-4. Print preview of a typical page from the old site, with rounded-corner images clearly visible

For this reason, many of the core pages (for instance, all of our online course information) needed a separate "printer-friendly" version, as in the example in Figure 15-5. Luckily, many of the pages were already database-driven, so I only had to create a new page that pulled the content from the database and presented it without any of the surrounding site template that was causing the problems. But this solution was inelegant and still left the remainder of static pages on the site looking shabby when put on paper.

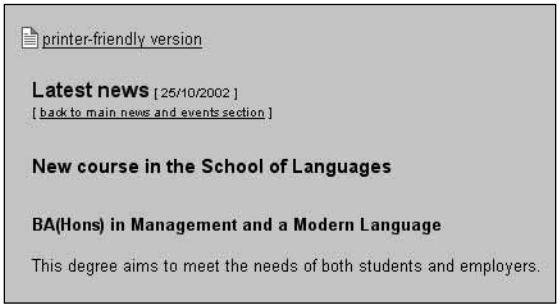


Figure 15-5. Detail of an events page, showing the printer-friendly version link

Another problem area was the site navigation. As shown in the following code, classic Dreamweaver "jump menu" drop-downs were used to provide the navigation for all major site areas and subpages within the current section.

```
<SCRIPT LANGUAGE="Javascript">
<!--
function gotoPage(number) {
  i=document.navform.elements[number].selectedIndex;
  parent.location.href=➥
  document.navform.elements[number].options[i].value;
}
// -->
</SCRIPT>


<SELECT NAME="choose" onChange="gotoPage(0)">
  <OPTION VALUE="#">Navigate the site</OPTION>
  <OPTION VALUE="#">.....................................</OPTION>
  <OPTION VALUE="http://www.salford.ac.uk/">Homepage</OPTION>
  <OPTION VALUE="http://www.salford.ac.uk/reception/">Online
reception➥
  </OPTION>
  <OPTION VALUE="http://www.salford.ac.uk/choose/">Choose Salford➥
  </OPTION>

...

</SELECT>
```

In the very first version of the site, these navigation menus were completely reliant on client-side scripting. Simply marked up as a couple of `<select>` elements, not contained inside any `<form>` and certainly lacking a conventional submit button, these menus relied on the `<select>` element's onchange event to load a new page whenever the user changed the current selection.

Apart from making the navigation unusable when client-side scripting is unavailable or disabled, this makes keyboard access (with or without assistive technology) tricky. Using default cursor keys to move through the available options triggers the script. Effectively, this makes it impossible for keyboard users to move beyond the first `<option>` unless they are aware of work-arounds built into their browser or assistive technology.

An example of a work-around for keyboard users in Internet Explorer is using Alt+cursor keys to "open" the `<select>` *menu—a concept that certainly doesn't translate well to the nonvisual realm. This suppresses any* onchange *event until the* `<select>` *menu is closed by either pressing the Enter key or tabbing away from it.*

The worst use of these jump menus was the Online Reception page, shown in Figure 15-6. This was a one-stop-shop page used in lieu of a proper index or site map, which had no less than six `<select>` elements to provide quick links to the majority of the University's top-level pages and departmental sites.

Figure 15-6. The University's Online Reception page, featuring a total of six JavaScript-driven jump menus (excluding the main navigation)

Some of the drop-downs on the Online Reception page also nicely demonstrate another shortcoming of `<select>` elements: the difficulty with providing effective levels of grouping and subcategorization. Although HTML does offer the ability of grouping individual `<option>` elements inside an `<optgroup>`, support for this element was (and still is, in certain browsers) flaky or nonexistent. Therefore, the designers used a variety of visual tricks to "fake" the semblance of hierarchy and structure within the `<select>` elements by introducing empty spacer `<option>` elements and prefixing subpages with a simple dash—a purely visual subterfuge. Figure 15-7 shows an example.

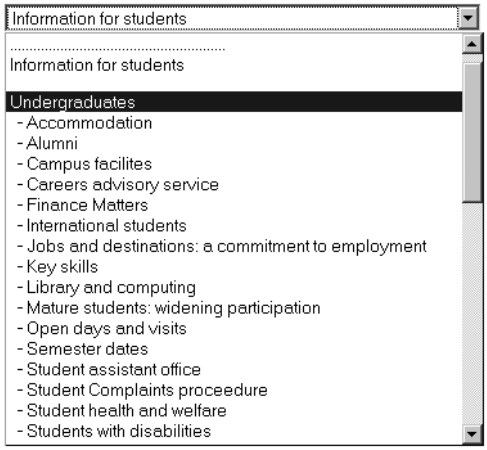


Figure 15-7. Faking a structural hierarchy with spacer <option> elements and dashes

The following is the code that creates the menu in Figure 15-7:

```
<SELECT NAME="choose" onChange="gotoReceptionPage(2)">
    <OPTION VALUE="..." SELECTED>Information for students</OPTION>
    <OPTION VALUE="#">.....</OPTION>
    <OPTION VALUE="...">Information for students</OPTION>
    <OPTION VALUE="#"></OPTION>
    <OPTION VALUE="...">Undergraduates</OPTION>
    <OPTION VALUE="...">  - Accommodation</OPTION>
    <OPTION VALUE="...">  - Alumni</OPTION>
    <OPTION VALUE="...">  - Campus facilites</OPTION>
    <OPTION VALUE="...">  - Careers advisory service</OPTION>
    <OPTION VALUE="...">  - Finance Matters</OPTION>
...
</SELECT>
```

Other than inconveniencing, or outright excluding, certain users, this type of navigation also had a noticeable impact on the University's search engine rankings. With hardly any real links in the markup, content was effectively hidden from search engine spiders. Even our own internal search engine needed to be explicitly pointed to specific URLs to index in order to keep search results relevant.

Along with these problems were the usual suspects of inaccessible sites: a proliferation of click here links, extensive use of graphical buttons, inappropriate use of markup to achieve a certain type of visual presentation, and so on.

In short, the site that I had inherited was a mess. Unfortunately, the University had just launched its new design (after a considerable investment of money and time), and many departments across the institution had already started the process of painstakingly transferring their content into the new cumbersome layout. It was therefore decided that, rather than embarking on yet another redesign so soon after the last one, just the most obvious accessibility issues should be fixed.

The Initial Fixes

A small retrofit in its own right, my initial "damage limitation" exercise concentrated mainly on the two biggest problems: making the table-based layout linearize properly and making the drop-down navigation at least workable for non-JavaScript users.

Paradoxically, to fix some of the linearization issues of the template, the tables themselves had to be broken down into further nested tables, thus adding even further "scaffolding" markup to the already convoluted code.

To make the navigation work even without client-side scripting, I implemented a rudimentary server-side fallback mechanism. The two separate `<select>` elements in the header (as well as the six jump menus on the Online Reception page) were each wrapped in a proper `<form>` element, and a discreetly designed image-based submit button was included. When submitted, the destination's URL was passed to a simple PHP script, whose only purpose was to redirect the user's browser to the desired target page. This approach was not very elegant, as it added an unnecessary round-trip to the server and made it impossible to gather any useful page-referrer information within the site (although, admittedly, the PHP code could have been further expanded to separately log this type of information). Nonetheless, this fallback at least meant that users without JavaScript would be able to navigate the site. Figure 15-8 shows the new menu.

Figure 15-8. Tweaked jump menu, now wrapped in a form element with an unobtrusive image-based submit button to the right

The following is the HTML that creates the tweaked jump menu shown in Figure 15-8:

```
<form action="/redirect.php" method="post">
    <select name="select" onchange="MM_jumpMenu('document',this,1)">
        <option value="#" selected="selected">Navigate the site➥
        </option>
        <option value="#">_____</option>
        <option value="http://www.salford.ac.uk/">Homepage</option>
        <option value="#">_____</option>
        <option value="http://www.salford.ac.uk/about/">➥
        About the University</option>
        <option value="http://www.salford.ac.uk/faculties-schools/">➥
        Faculties and Schools</option>
```

```
        ...

    </select>
    <input type="image" border="0" name="submit"➥
    src="/images/common/go.gif" alt="Go" width="11" height="11">
</form>
```

And the following is the corresponding server-side code to handle the redirection:

```
<?php
// expects a value for 'select' either GET or POST
if (isset($_REQUEST['select'])&&(!($_REQUEST['select']=="#")))
{
    // redirect user to selected page
    header('Location: '.$_REQUEST['select']);
} else {
    // default behaviour - go back from whence you came
    header('Location: '.$_SERVER['HTTP_REFERER']);
}
?>
```

Initially, to address the keyboard access problems for users that *did* have JavaScript, I also removed the client-side scripting completely. However, this change immediately prompted a barrage of messages from users and management, reporting that suddenly the navigation was "broken." Grudgingly, I had to take the pragmatic decision to reintroduce the JavaScript-based functionality (on top of the server-side fallback) in order to make the fundamentally flawed navigation scheme usable for the majority of users, while acknowledging that keyboard users would still be faced with some difficulties.

The Redesign

At the start of 2003, the University undertook a review of its corporate identity guidelines. A new design agency was tasked with expanding the existing guidelines (which were quite minimal, mainly centered around the appropriate use of the University logo, our two core corporate colors, and the right choice of typeface) to provide additional color combinations and a modular set of templates for print and electronic formats. Although the agency didn't specifically look at guidelines related to the Web, I worked closely with the designers to ensure that it would be possible to interpret and adapt the revised identity for online use.

In contrast with the comparatively lax previous corporate identity, the new guidelines introduced a set of tighter restrictions. For instance, they prescribed the top-left position of the logo (which was also slightly altered to now include the University name and the "A Greater Manchester University" slogan in a set configuration next to the original logo roundel) and a limited palette of primary and secondary colors that could be used.

As these new guidelines applied to our site as well, it was clear that the blue-and-orange color scheme and the general page header would have to be revised. So, after two years

of "blueberry-and-custard," we were faced with a fundamental choice: Should we simply apply a new facade to the University's web presence, changing only visual aspects of the markup while maintaining the rest of the underlying, table-based structure? Or should we learn from past mistakes, be mindful of the fundamental accessibility problems present in the current site, and make a fresh start?

Clearly, a complete rebuild would have involved a considerable amount of work, with every page of the site having to be recoded. However, remember the complete absence of separation of content and presentation? Because the colors and general layout were hard-coded into the presentational HTML, even superficial changes would have required work on each page. I therefore decided to seize the opportunity of this corporate identity review to radically change the University site for the better, rather than awkwardly carrying on with small fixes and tweaks to the existing markup.

Unlike many other web managers in charge of corporate sites, I was fortunate that management allowed me a certain level of flexibility in making these kinds of decisions autonomously. Apart from a series of briefings to key groups of internal stakeholders, centered mainly on the need to redesign from a corporate identity point of view, I did not have to do a "hard sell" to convince the University to move toward a web standards–based approach. None of these briefing sessions ever actually mentioned web standards or layouts without tables in any technical detail. I merely put the point across that the redesign would follow modern web development best practice. Similarly, for the accessibility side of the redesign, I did not enter into any lengthy explanation of the Web Content Accessibility Guidelines (WCAG), assistive technology, or people with disabilities. I merely reassured management that, in adopting the proposed best practice strategies, the University's site would fall in line with legal requirements with regard to SENDA.

> *Web managers who are having more difficulty in convincing their bosses or clients of the value of a web standards–based approach may be interested in the documents put together by the Making A Commercial Case for Adopting Web Standards (MACCAWS) research group, at* `www.maccaws.org`.

Decisions in Early Planning Stages

A number of decisions had to be made during the early planning stages of the redesign. Some of the important ones were to design to web standards, to use XHTML, and to use CSS rather than table-based layouts. The following sections look at these decisions in some detail.

Why Web Standards?

Nowadays, the majority of web professionals should at least have heard of web standards, thanks in no small part to the ongoing efforts of grassroots movements such as the Web Standards Project (`www.webstandards.org`), the increased availability of valuable online and paper-based resources on the subject, and excellent examples of large corporate sites that have made the switch to valid, table-free, structural markup.

At the time of the University of Salford website redesign, however, these resources were few and far between. Designing with web standards, thoroughly separating content from presentation, and abandoning tables in favor of pure CSS solutions were still the exceptions. However, the fundamental benefits of a web standards–based approach were already becoming evident:

- Coding to official World Wide Web Consortium (W3C) specifications and abandoning proprietary markup (more often than not aimed squarely at Internet Explorer) ensures that pages work in the majority of browsers, without the need for separate versions or the risk of shutting out a percentage of your visitors.
- The separation of content and presentation results in "lighter" web pages, devoid of presentational clutter, that load and display considerably faster and are easier to maintain.
- Thinking beyond this particular design, with an eye on the possibility that the University's branding or visual guidelines may change in future, having all aspects of the presentation neatly centralized in a handful of CSS files allows for a quick way to radically alter the look and feel of an entire site, without needing to amend each individual page.

As at the time the University's central web team was composed of only myself (a team of one), all of these benefits certainly would have helped in "working smarter, not harder" and giving me better control over the institution's site.

Why XHTML Rather Than HTML?

Generally, many developers assume that "designing with web standards" implies the need to switch to XHTML. Of course, that's not necessarily true. It is perfectly possible to use HTML 4.01 (preferably following the Strict `DOCTYPE`) to create lean, semantically structured sites. With enough "personal discipline"—consciously avoiding presentational language elements such as bold, italic, and `<font>` tags in favor of structural equivalents and CSS—you can follow the principles of web standards (valid, semantic code and the separation of content and presentation) without the need for XHTML.

> *Purists will argue that the only real benefits of XHTML can be reaped when combining it with other XML-based technologies (for example, mixing XHTML and MathML). But in order to do this, pages need to be sent with an XML MIME type (such as* `application/xhtml+xml`*), which Internet Explorer does not understand. This is a fact that purists will often use as proof positive that, for general-purpose web pages that do not mix XML technologies, there is no reason to use anything other than plain HTML.*

The decision to move the University site to XHTML was dictated not so much by any intrinsic advantage of XHTML over HTML, but by the need to make life easier for myself when it comes to testing and quality assurance. Because the majority of presentational markup has been removed from XHTML (with a few debatable exceptions like the `sub`/`sup` elements), running pages through the W3C markup validator quickly brings to light any use of presentational elements and attributes. This is a particularly useful first test when checking pages developed by departmental web authors or third-party suppliers. Of course, it's still

possible to create completely nonsemantic and inaccessible documents that are completely valid XHTML/CSS, so this sort of testing needs to be followed up by a proper look at the markup. Nonetheless, it can be helpful as a very superficial initial assessment.

Another thought at the back of my mind when choosing XHTML was the planned adoption of a content management system (CMS) by our IT department a few years further down the line. With the possibilities opened up by simple, readily available technologies such as XSLT and the use of XML-based languages for data import into such systems, it would be possible to repurpose web pages by transforming XHTML into a format suitable for inclusion into the CMS. Further potential applications, such as print-on-demand systems where XML content can be loaded directly into a desktop publishing package such as Adobe InDesign or Quark Xpress, further influenced this decision.

Why Move Away from Tables?

The syntax of XHTML still allows the use of tables, and rightly so, as the table construct is ideally suited to mark up the complex relationships of headings and values of tabular data. In that respect, it's obvious that tables are not purely presentational. However, web designers have traditionally relied on table markup for exactly that: defining a visual layout. Even stretching the definition, there is usually no relationship being defined in a table-based layout other than "I want these two columns of text to sit next to each other."

> *If you're ever in doubt about whether or not something could be considered "tabular data," ask yourself if you would put this information in an Excel spreadsheet. This method isn't foolproof though, as over the years, I've come to know many managers who treat Excel like a poor man's layout tool. Incidentally, these are usually also the people who think PowerPoint is a web design tool in its own right and proudly hand over their mock-ups with an "I've already done the layout of my pages for you as well" remark.*

The original definition for tables from the HTML 4.01 specification (`www.w3.org/TR/html401/struct/tables.html#h-11.1`) is slightly vague on the subject.

> The HTML table model allows authors to arrange data -- text, preformatted text, images, links, forms, form fields, other tables, etc. -- into rows and columns of cells. [...] Tables should not be used purely as a means to layout document content as this may present problems when rendering to non-visual media. [...] authors should use style sheets to control layout rather than tables.

Even WCAG 1.0 mentions the use of tables for layout, but fails to actually prohibit their presentational use (`www.w3.org/TR/WCAG10/wai-pageauth.html#gl-table-markup`).

> 5.3 Do not use tables for layout unless the table makes sense when linearized. Otherwise, if the table does not make sense, provide an alternative equivalent (which may be a linearized version). [Priority 2]
>
> 5.4 If a table is used for layout, do not use any structural markup for the purpose of visual formatting. [Priority 2]

Regardless of wooly definitions in the HTML specification and get-out clauses so readily provided in the WCAG, staying true to the idea of separating content from presentation means abandoning table-based layouts in favor of CSS.

There are obvious advantages to CSS-driven layouts. Externalizing all of a site's presentation (layout, colors, typeface definitions, and so on) to style sheets provides an excellent way of keeping control over the look and feel of a site at a central location. When rigorously applied to all pages, CSS provides an excellent means of maintaining a consistent layout and presentation across a site. It also gives a certain measure of future-proofing. For example, if the University's corporate colors or choice of overall design were to change, a large part of the redesign would involve nothing more than a change to the styles, rather than requiring a complete recoding of each individual page.

Another advantage of CSS is the ability to define different presentation styles for a variety of delivery channels. Through the use of `@media` directives or the media attribute for style sheet `<link>` elements you can specify how a document should look on the screen or in print, making a separate printer-friendly version unnecessary.

> *There are still edge cases in which a printer-friendly version may be required. One example would be a lengthy document that was split into separate pages for easier reading. A designer may want to provide the same document as a single, albeit long, page.*

See Chapter 9 for details on using CSS for accessible web pages.

Site Structure—Taking Stock

A complete redesign is a good opportunity to not only change the appearance of a site, but to also take a critical look at a site's structure and content. Sites can grow in a very organic fashion. Despite an initially defined information architecture, new pages and sections that don't quite fit the original structure are added at a later date. It's often surprising how much legacy content is still present, often carried from one redesign to the next.

Before embarking on a redesign, it is therefore useful to start with a completely blank slate. For the University of Salford website redesign, the first phase involved a meticulous inventory of any content available on the old site. Working with the respective content owners, outdated content was pruned and amended. To work out the new site's overall structure, I enlisted the help of various stakeholders across the University, including a sample of current students, in order to carry out simple taxonomy and card-sorting.

Card-sorting is a user-centered design method in which users sort a series of cards, each representing a particular piece of content or functionality of a site, into groups that make sense to them. Card sorting can help in understanding the users' mental models, thus allowing you to organize your content in a way that matches their expectations.

It's usually possible to make smaller design or structural changes directly on a live site. However, as this redesign involved a complete restructuring and rebuilding, I requested a new development server to be set up centrally by the University's IT department. Even if, as was the case with this redesign, you're the only developer working on a new site, there are many advantages to this approach over simply setting up a local development server on your own workstation. Chief among them is the ability to let management and other interested parties across the institution keep an eye on the work in progress. Coupled with a simple feedback form, this can give them the opportunity to have their say and effectively take a certain amount of ownership of the redesign (be careful to manage their expectations, though; otherwise, they will expect any single suggestion to be implemented immediately).

Based on the new information architecture, I prepared the development server by basically setting up an equivalent directory structure. I then proceeded to copy the relevant existing pages, plus any additional pieces of amended information (mostly provided by the content owners as Word documents) into their respective new folders.

So, with a new streamlined site structure and up-to-date content, it was time to get down to the nitty-gritty of the actual build.

Building Page Templates

In an ideal web standards workflow, the task of creating a site template can be broken down into two distinct phases:

- Content—creating the page structure in (X)HTML
- Presentation—applying a style and layout to the structure (CSS)

As shown in Figure 15-9, these phases would normally be carried out in sequence, practically in isolation. Reality, however, is never quite as structured as this ideal model.

Figure 15-9. Ideal web standards workflow as a one-way process: first create the structure, then the presentation

(End of Excerpt)