

Accessibility and Universal Design
Methods and Considerations for Web Developers

Katherine Irvine
CPSC 299
Office of Learning Technology, UBC
May 13, 2004

May 13, 2004

Katherine Irvine
Web Application Developer, Office of Learning Technology
LSK 209
University of British Columbia

Gwen Litchfield
Computer Science & Alternate Routes to Computing Coordinator
CISR 253
University of British Columbia

Dear Gwen,

Please find attached the work term report for my first Co-op term, during which I worked at the Office of Learning Technology at UBC. This report was prepared for the office, as a guide for developing fully accessible websites.

Sincerely,

Katherine Irvine

Accessibility and Universal Design
Methods and Considerations for Web Developers

For:

Gwen Litchfield

Department of Computer Science, Faculty of Science

By:

Katherine Irvine, 38933024

May 13, 2004

Office of Learning Technology, UBC

Summary

Accessibility is a big issue in web development, a lot bigger than many web designers may realize. A website designed with accessibility in mind will assist disabled users, overcome most browser-related issues, and enable access from devices ranging from cell phones to laptops to printers. There are many methods available to address the challenges presented by each of these users. This report outlines which group of users each of these techniques can help, and how to use them to enable those users to access your website. The techniques discussed in this report include several ways of identifying the audience, from sniffing the browser to having the user choose to access a different version of the site, server-side scripts and includes, hidden elements, stylesheets, and “accessible” markup.

Table of Contents

Introduction.....	1
Defining Universal.....	2
Report.....	4
Disabled Users	4
Browser Compatibility.....	6
Non-Standard Browser Settings.....	7
Device Compatibility	8
Conclusion	10
Appendix I: Case Study	11
Appendix II: Interview.....	16
Appendix III: Stylesheets.....	18
Appendix IV: Creating a Second Version of Your Site.....	21
Appendix V: PHP functions.....	23
Bibliography	26

List of Tables

Mapping the Audience to the Technology.....	3
---	---

Introduction

For many web developers, including those that work within large organizations and institutions, the concept of web accessibility means adding “alt” tags to images, using contrasting colors, and making fonts big enough; the rationale simply being to make a website readable by the visually impaired. Accessibility, however, should be about more than just “alt” tags, and will be appreciated by more people than one would think at first glance. Consider the following:

A blind person using a screen reader finds that the drop-down menus are impossible to use, preventing her from accessing any of the content buried in the site.

Someone at home attempts to view a really cool site shown to him by a co-worker, only to find that the site appears broken on his older browser, with the footer floating over the main text and other elements not appearing at all.

Someone on a 56k telephone modem is faced with unbearably long page load times on a graphics-rich site.

A student, attempting to access the online library catalogue from her cell phone, is frustrated because the tables used for layout have jumbled all the information on the smaller screen.

Someone who has disabled JavaScript on their browser finds he cannot submit a form, nor can he print it out and mail it in, as the left-hand navigation bar causes the page to be far wider than his paper.

Only one of these people is likely to be considered when the concept of accessibility is discussed. Yet all of them will benefit from accessible websites, and should be considered when creating a site that is dedicated to universal design. Universal design might simply be called accessibility taken to its logical conclusion. However, doing so would be misleading. Universal design requires a shift in thinking, from “fixing” a website to accommodate people with problems, to enabling everyone to access the website. The result is that the website is easily viewed anywhere, by anyone, through any media. Universal design is closely linked with information architecture, functionality, and site maintenance. Broken links, confusing layouts, and improperly set-up databases are the worst kind of accessibility flaws; they render your site unusable to everybody, not just a disadvantaged few.

Creating a universally accessible site means putting in some extra effort and ensuring things are done properly, but it is not nearly as daunting a task as it may first appear. Nor does it require abandoning a graphics-rich site or an ornate, detailed layout. By showcasing various methods and techniques and explaining how each is related to the concept of universal design, this report will show how current design standards can easily be used to improve accessibility.

Defining Universal

According to Section 508, the United States law that sets minimum standards for online accessibility, universal design is “[t]he concept or philosophy for designing and delivering products and services that are usable by people with the widest possible range of functional capabilities. This includes products and services that are directly usable (without requiring assistive technologies) and those that are made compatible with assistive technologies.” (508 Universe Glossary)

“Everyone, everywhere, from every media,” as it was summarized in the previous section, is too vague to translate into a piece of code. Unless you have a text-only website, which is rather rare these days, the following list of specific audiences and situations should be kept in mind. Explaining web design techniques for accommodating each of these audiences is the focus of this document.

- Disabled users
 - Blind
 - Visually impaired
 - Colorblind
 - Difficulty using a mouse
- Browser Capability
 - New Netscape
 - Old Netscape (Version 4)
 - Internet Explorer
 - Safari (Macintosh browser)
 - Mozilla (and spin-offs)
 - Gecko and Opera
 - Text-only browsers (eg. Lynx)
- Non-standard browser settings
 - JavaScript disabled
 - Images disabled
 - Stylesheets disabled or custom stylesheets
 - Cookies disabled
 - High security settings
 - Slow connection
- Device Capability
 - Cell phones
 - PDAs
 - TVs
 - Printing

Each of these situations has specific needs and challenges; however, many of the needs, and therefore many of the solutions, are shared between several situations. For example, the kind of stripped-down, low-graphics site that fits a traditional description of “accessible” and would be appreciated by those with impaired vision generally renders quite well on a cell phone or a text-only browser, and both those using screen readers and

those with images disabled on their browsers will appreciate meticulous and consistent use of “alt” tags.

Table 1 illustrates the relationship between the methods and solutions which will be discussed in this article and the specific accessibility issues which they are capable of addressing.

	Identifying the Audience				Addressing Accessibility Issues					
	Sniff Browser	Media Types	Cookies	User action (eg. clicking a link)	Server-side Scripts and Includes	Styles and Stylesheets	Hidden Elements / Customizing Content	Flexible Layouts	Low-graphics Version of the Site	“Accessible” markup
Disabled users ~Blind ~Visually impaired ~Colorblind ~Difficulty using a mouse		●	○	○		●	●	○	○	●
Browser Capability ~New Netscape ~Old Netscape ~Internet Explorer ~Safari ~Mozilla ~Gecko and Opera ~Text-only browsers	● ● ● ● ● ● ●					● ● ● ● ● ● ●	○	●	●	●
Different browser settings ~JavaScript disabled ~Images disabled ~Stylesheets disabled ~Custom stylesheets ~Cookies disabled ~High security settings ~Slow connection	○ ○ ○ ○		○	● ● ● ●	● ●				● ● ● ●	● ●
Device Capability: ~Cell phones ~PDAs ~TVs ~Printing	○ ○ ○	● ● ● ●		● ●		● ● ● ●	● ● ● ●	● ● ● ●	● ●	

Table 1 – Mapping the Audience to the Technology

Black circles represent those areas where there is a strong correlation; white circles represent those areas where the correlation is weaker or intermittent.

Report

This report is intended for web designers who are already at least passably familiar with Cascading Stylesheets, Server-Side Includes, and at least one server-side scripting language, such as PHP, which is the language used in the examples.

Cascading Stylesheets, or CSS, is a method for defining styles in a web document. External stylesheets, which are discussed in this article, are separate files from the webpage itself, and consist of a list of attributes to assign to particular tags and classes, which are a way of attributing a specific style to an element. The stylesheet is then called from the webpage, and the browser applies those styles when it renders the page. A good CSS tutorial can be found at <http://www.w3schools.com>.

Server-side includes are a way of including the text of another file in webpages. For those running Apache, information on enabling and calling includes can be found at <http://httpd.apache.org/docs-2.0/howto/ssi.html>. Server-side includes can be called using a server-side scripting language, such as PHP.

Server-side scripting languages, such as PHP, are used to write scripts that are interpreted and executed by your server before it sends the webpage to the browser. Client-side scripting languages such as JavaScript, on the other hand, are interpreted by the browser after it receives the web page. Server-side scripting languages tend to be more powerful than client-side scripting languages, and can even do things that would seemingly be easier to do client-side, such as identifying the browser that requested the page (often called sniffing the browser). PHP documentation can be found at <http://ca3.php.net/manual/en/index.php>.

For a real-world application of many of the accessibility recommendation included in this report, a case study of UBC's Teaching and Academic Growth website revision is included as Appendix I. This study will be published as a separate paper at a later date.

Disabled Users

Disabled users, especially those who are visually impaired or blind, are often the first users who come to mind when accessibility is mentioned. They may use assistive technology, such as screen readers and screen magnifiers. An interview with someone who uses a screen reader can be found in Appendix II.

Often this group cannot be singled out by the webpage or server, except for those who use screen readers. Screen readers, which read web pages out loud, are categorized as an aural media, and therefore can be identified by a stylesheet. The "media" attribute of a stylesheet is a way to link a specific stylesheet with a certain audience, identifying separate styles for screen, aural, and other audiences. A complete list of available media types for stylesheets can be found in Appendix III.

There are two ways to link a certain style with a specific media (Box 1). The first specifies a media for an entire stylesheet, while the second specifies a media for a single rule. Not only is it possible to load a different stylesheet for each media type, but there are also media-specific styles which can be applied. One example of a style available for the aural media is changing the pitch of the speaking voice.

```
Box 1
1. <link href="styles3.css" rel="stylesheet"
   type="text/css" media="aural" />
2. @aural h1 { ... }
```

Of course, you can still use all of the common styles in a stylesheet dedicated to aural media. Some, like font color, will simply be ignored. Others will have more meaning, making an element hidden or visible, for example.

“Visibility” is an attribute used to make elements visible to certain audiences and hidden from others. An example of this is a link to the accessible or text-only version (if there is one) or to the full-graphics version (from the accessible version). This should be the first item on the page when viewed with a screen reader, but that may not be a good place to put it in terms of the visual design of your site. Therefore, you should put this link in the correct place for the screen reader, hiding it in the regular stylesheet and making it visible in the aural stylesheet. Similarly, a “skip to content” link should be the second item visible to aural medias. Top and left navigation elements should be located before the body of the page in your code, so that they appear in that order to screen readers and in a text-only version of the webpage.

A text-only or low-graphics version of your website with a table-less layout, no Flash or JavaScript, scalable fonts, and well-contrasting colors will be easier to access for those using screen readers, for those with impaired vision or colorblindness, and for those with difficulties using a mouse. However, if the main site is well designed, with a flexible layout (using “div” tags rather than tables) and all the styles in external stylesheets, a separate “accessible” site may be a bit of overkill. Instead, the site may simply offer the user a choice of skins or styles. Information on creating a separate version of a website can be found in Appendix IV, while information on switching stylesheets can be found in Appendix III. If the user is offered a choice of stylesheets, the developer may wish to identify returning users’ preferences with the use of cookies.

Finally, there is the issue of accessible markup. These are html elements such as “label” and “alt” which were designed to provide accessibility solutions. They should definitely be used consistently. Every image should have a descriptive “alt” tag, even if the description is merely “clear gif used for formatting,” and every form element should have a label, even obvious things like submit buttons. If a label looks awkward when the page is viewed in a browser window, simply set the visibility to “hidden” in the style sheet dedicated to the screen media.

Browser Compatibility

Determining which browser is being used is fairly straightforward. It can be done in whichever scripting language you prefer (Box 2). However, it is generally better to use server-side programming rather than client-side if possible on your web site because of differences in browser settings.

```
Box 2
PHP:
$_SERVER['HTTP_USER_AGENT']
JavaScript:
navigator.appName
navigator.appVersion
```

It will probably be necessary to load a different stylesheet based on the browser is being used; while most browsers handle basic CSS well, many have quirks and the webdesigner must compensate for them. When adding styles to your website, an external CSS stylesheet is the most accessible and the most flexible. It is also the most efficient when it comes to keeping styles consistent across an entire website, even when updating or changing the look and feel of your site. External stylesheets also allow you to overwrite your own styles. This may be done for various reasons, one of which is to compensate for browser discrepancies.

Among others, nested floating elements, padding/width, and pseudo-classes are handled slightly differently in different browsers. This means that the styles must be defined differently for each browser, to account for side-effects and quirks and achieve an identical appearance in each browser. These differences, and the ways around them, are very well documented on various sites on the Internet. Especially important is to have flawless presentation on Internet Explorer, since it is the most widely used browser. Sometimes these differences can be dealt with using work-arounds in a single stylesheet, but sometimes it is easier to simply load an appropriate “corrections” stylesheet on a per-browser basis, using server-side scripting to choose the stylesheet (Box 3).

```
Box 3
if ($client_browser == "NS4") {
echo '<link href="stylesNS4.css"
rel="stylesheet" type="text/css" />';
}
```

An example of a function returning `$client_browser` information is available in the Appendix V, and an example list of stylesheets is in Appendix III. When using multiple stylesheets, remember that the stylesheets are applied in the order that they are listed; in other words, the last stylesheet will overwrite the first if they both mention the same aspect of the same element.

Finally, it is sometimes the case that adjusting your styles for a certain browser is either not worth the effort, or is simply impossible. In these cases, it is necessary for the site to degrade gracefully. For a site that has been designed entirely in CSS with a table-less layout, it is generally quite easy to pass a specific browser a stylesheet which will render a text-only version of the page. If you have already created a second version of the site for accessibility reasons, or to render on mobile devices, it can be passed to incompatible browsers. Text-only browsers can be treated similarly, or they can be the regular stylesheet, as long as that version of the page has “alt” tags for every image and does not use tables for layout.

Non-Standard Browser Settings

As mentioned before, when creating dynamic content, server-side scripting languages such as PHP and ASP are better for accessibility than client-side languages, such as JavaScript. Because server-side languages do not depend on the client’s browser capabilities, they still work if the end user has JavaScript disabled, is using a screen reader, or is accessing your page from a cell phone. As well, JavaScript may not work with high security browser settings, which can block the execution of client-side scripts and executables.

A second issue when considering browser settings is images. People who have disabled automatic downloading of images need “alt” tags for all images. They will appreciate the option of viewing a text-only version of the site, as will people who have slow Internet connections. This can be achieved either by having a separate parallel version of the site or by offering the user a choice of stylesheets. More information on these techniques can be found in the section **Disabled Users**, and also in Appendices III and IV.

In the case of the user with the slow connection, it bears noting that setting the visibility of an image to hidden does not cut down on the loading time of a page. However, if many background images are being set by an external stylesheet, changing the stylesheet to one with no background images will minimize the loading time of the page. Another option is to use different server-side includes if your main site has a graphics-rich header and footer.

Server-side includes are useful for website maintenance, which in turn is good for accessibility: no-one will be able to access a webpage if the link to it is broken, and server-side includes, especially of standard navigation elements, help cut down the number of broken links in your site. Standard navigation elements also help cut down on confusion of people trying to find their way around your site.

Because includes can be called from anywhere, it is necessary to have all links inside an included page begin from the site root. It is also possible to have includes embedded in each other; these should also be called from the site root, rather than a relative path. PHP has some quirks in the way includes are handled; see Appendix V for tips. Server-side includes can also be used to include functions and lists of variables; this is useful if a

function is used frequently (such as to create breadcrumb navigation), or some information is accessed often, but might eventually change (such as the email address in a “contact us” link). Storing such variables in an external file, which is then included in every page, is less likely to result in broken links and misdirection later, since the information needs only be changed in one place.

Finally, there are different settings regarding stylesheets beyond simply how different browsers interpret CSS. Some users have their own custom stylesheet, which is loaded to overwrite the styles of the page that is being loaded. It is impossible to check how the page will render with a client-side stylesheet, but it is best to use only external stylesheets so that the page will render the way they expect, perhaps with enlarged font sizes. The other setting to consider is stylesheets being disabled entirely. This can and should be checked, to ensure that the site is still usable at least, even if it is not quite as elegant.

Device Capability

Using `<div>` tags and CSS in conjunction is the most likely method to give you a well designed, flexible, accessible layout. Flexibility is important because a website can be viewed on a 4-inch cell phone screen, a 24-inch flat screen TV, and everything in between. A flexible layout will accommodate all those users, while a more traditional tables-based layout is likely to break on a monitor much smaller than the one it was designed on, or if someone has increased or decreased their default font size.

Having said that, it is a good idea for a website to be no wider than 750 pixels in the standard stylesheet. First of all, people with 24-inch monitors will probably not have their browser windows maximized. Secondly, it is difficult to read something wider than 750px.

Furthermore, separate stylesheets may need to be loaded for different media types, especially to distinguish between screen, print, and handheld. There are several media-specific styles available. For example, there is a set of styles for printing, which can control where page breaks occur and other print-specific issues.

Hidden elements can be used to give custom content to certain users, depending on which stylesheet is being used. Three examples of custom content follow, two for printing, and one for handheld media.

1. Link URL – Put the URL of the link after the link, in parentheses, visible only in the print media. It has been recommended elsewhere that one use the `:after` pseudo class with `content: attr(href)` for this, but this is not supported in many browsers.
2. Print message – This gives the URL of the page, the site root, and other information, so people can remember where they printed it from. This should be visible only to the print media and may be located at either the top or bottom of the page. An easy way to do this is to use `$_SERVER['PHP_SELF']`; which in PHP holds the current URL from the site

root (or the equivalent in another language) and include this message in your header or footer server-side include.

3. Telephone number – Most webpages have a link (often somewhere in the header) labeled “contact us,” to send an email. When viewing a page on a cell phone, it seems more natural to be given a phone number than an email address, so make that visible beside the contact us link for the handheld media.

A note about hidden elements: Netscape 4 does not recognize the visibility style. Furthermore, if relative positioning is being used, most browsers will display a blank space the size of the hidden element. Box 4 shows the solution to both problems. More information on stylesheets and media types can be found in Appendix III.

```
-- Box 4 --
.hidden {
visibility: hidden;
position: absolute;
}
```

Conclusions

Many developers may not realize the importance of accessibility on the web. A website that does not acknowledge and accommodate the diversity of today's web users may be inaccessible to a large portion of its potential user base. Those wishing to create a well designed website must keep in mind disabled users, those using any of the many web browsers now accessible, with a variety of settings, and those using different devices, from cell phones to laptops to printers.

The best way to address issues in accessibility is by making a shift in perception, from adding accessibility fixes to a webpage, to designing the page following the ideals of universal design right from the beginning. While this may entail a larger amount of effort and difficulty level when first designing a website, the cost is not as high as some might fear, and it will ensure a decreased cost of fixing things down the road. The information in this report and other articles on accessibility should help you to design a website that is usable by the most number of people in the most number of situations possible.

Appendix I: Case Study – TAG Website

The Teaching and Academic Growth website, which I co-developed with Novak Rogic, is universally accessible, with an online registration system and a randomized featured project article on the home page. I used PHP as the server-side scripting language.

The TAG Home Page design is shown in Fig. 1.



Fig. 1 – TAG website design

Until it goes live at www.tag.ubc.ca, it can be found at www.tag.ubc.ca:888.

Below, I've documented how we accommodated each of the audiences mentioned in **Defining Universal**.

Disabled Users

The TAG site is AAA Bobby Approved, and was given the thumbs up by Michelle Creedy, whom I interviewed for Appendix III. Font sizes are specified in percent rather than pixels, which allows users to increase and decrease the font size using their browser settings. As well as making the text easier to see, the clickable link area can be increased by this method, since nearly all the links are text rather than images. The site was also tested for color contrast at www.vischeck.com.

Browser Compatibility

It was fairly straightforward to get the page set up in Netscape, Mozilla, and Firefox, on both Mac and PC. (Fig. 2 and 3)

Using various stylesheet hacks, I managed to get it to look the right way in Internet Explorer, and even IE 5 for Mac. I used both the box hack and the backslash-comment hack, which are documented in Appendix I. (Fig. 4 and 5)

I used a separate stylesheet to accommodate Safari and Opera, since they deal with floated elements differently. I had to give up on the multi-columned text in the main content area, and specify a margin for those div tags to get them clear of the left navigation (even while using `position: relative`). (Fig. 6 and 7)

Camino (a Mac browser) and Netscape four were beyond my willingness to hack, so I simply passed them stylesheets for text-only versions of the site. I used a separate stylesheet for Netscape 4, because it still required a bit of hacking. (Fig. 8 and 9)

Text-only browsers were passed the same stylesheet as Camino. In fact, all unrecognized browsers are given this stylesheet. Images, of course, all have “alt” tags. The final problem with text-only browsers is tables; this was circumvented by using div tags for nearly all the formatting. The site was tested in Lynx and had no problems.



Fig. 2 - Mozilla (PC)



Fig. 3 - Firefox (Mac)



Fig. 4 - Explorer (PC)

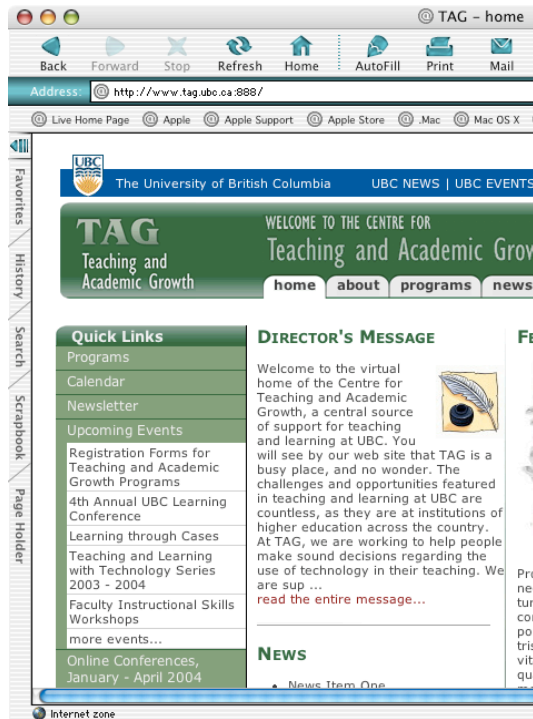


Fig. 5 - Explorer (Mac)



Fig. 6 - Netscape(PC)

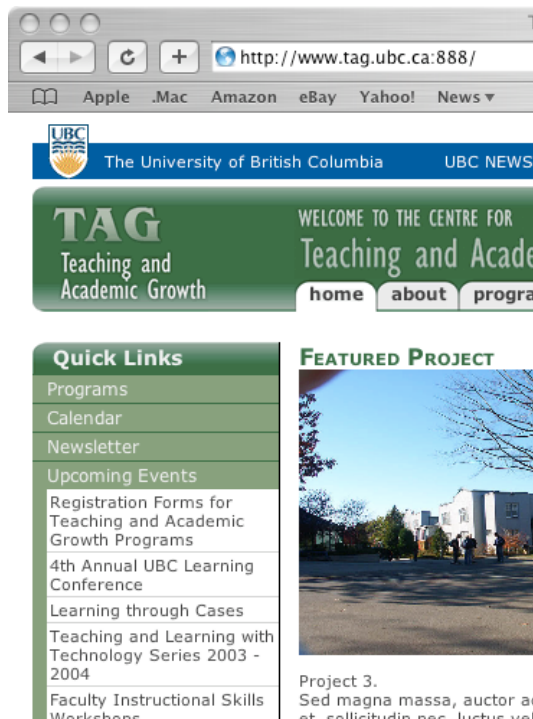


Fig. 7 - Safari (Mac)



Fig. 8 - Netscape 4 (PC)

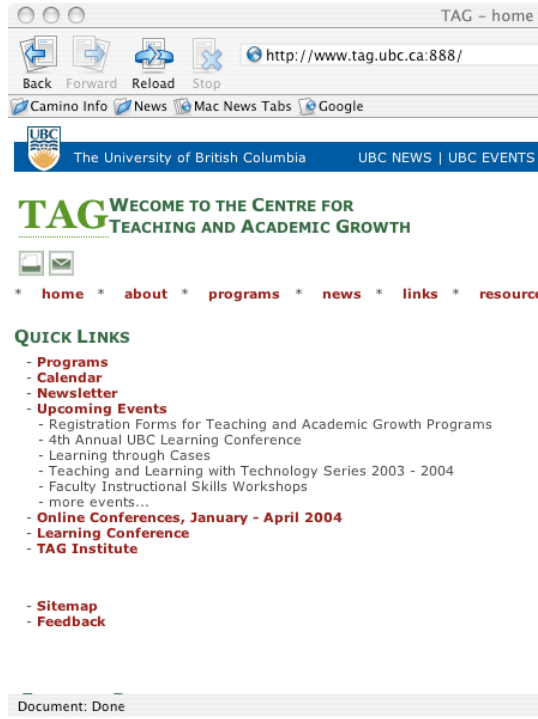


Fig. 9 - Camino (Mac)

Non-standard browser settings

The site uses no JavaScript. All coding is done server-side, with PHP. Images all have “alt” tags. Viewed without stylesheets, the site looks a little odd, but is still perfectly usable. Since all the styles are in external stylesheets, they can be overwritten by people who need custom styles.

Device Compatibility

Mobile devices get the text-only stylesheet. (Fig. 10)

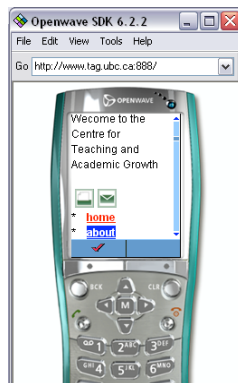


Fig. 10 - Cell Phone Simulation

The print stylesheet (Fig. 11) removes the left navigation, renders the top navigation and footer the same as the text-only page, and reveals the following message:

Our website is available at: <http://www.tag.ubc.ca:888>

The accessible version is available at: <http://access.tag.ubc.ca>

Thanks for printing this page from: <http://www.tag.ubc.ca:888/index.php>



Fig. 11 – Print Preview

Appendix II: Interview

I asked Michelle Creedy, a student at UBC who uses a screen reader, about her experiences using the web.

1. How much time do you spend online, and for which reasons/tasks?

I do most things online when possible. I send assignments, do research, and anything else that I can.

Which assistive technologies do you use?

JAWS 5.

When did you start using a computer and surfing the web?

1995

2. How accessible is the internet at the moment, approximately? ie, what portion of the content is fully accessible, partly accessible, not at all accessible?

It is relatively accessible. PDF is not accessible and nor is JAVA.

Do you think this is acceptable?

No, everything should be accessible.

Have you noticed an improvement since you started going online?

somewhat

3. I understand you're a student at UBC. How accessible is the university's web sites?

There are no standards for accessibility at UBC. There is no consistency.

WebCT?

Again, it changes a lot so there is no consistency.

Individual professor's homepages/course pages?

These are the worst! They use a lot of PDF. They also do not have guidelines to follow.

4. What is the most important aspect of accessibility that web designers should consider, or the first thing someone updating a website should fix?

Have other options besides PDF such as word or HTML. Label graphics and put labels on the buttons that explain what they are for. Avoid labels such as "more" or "click here". Have buttons on the page so people can enlarge text or change contrast. Avoid JAVA. Avoid flash pop-ups. Make sure site is Bobby-approved.

5. Can you give me worst-case and best-case examples of pages currently on the web?

Best - www.guidedogs.com

Worst are sites where people can download things such as music.

What features and aspects of the sites make them eligible for those labels?

Guide dogs has a fully accessible site with labels and a text-only site. The music or other download sites have pop-ups and sometimes JAWS can't even read the buttons.

6. If you have a choice between doing something online or offline (in person, on the phone, etc), which are you more likely to do? How much does this depend on accessibility?

I would do it online if the site was accessible or in person or on the phone if not.

Appendix III: Stylesheets

A list of possible media types (taken from the W3 standards website):

all

Suitable for all devices.

aural

Intended for speech synthesizers. See the section on aural style sheets for details.

braille

Intended for braille tactile feedback devices.

embossed

Intended for paged braille printers.

handheld

Intended for handheld devices (typically small screen, monochrome, limited bandwidth).

print

Intended for paged, opaque material and for documents viewed on screen in print preview mode..

projection

Intended for projected presentations, for example projectors or print to transparencies.

screen

Intended primarily for color computer screens.

tty

Intended for media using a fixed-pitch character grid, such as teletypes, terminals, or portable devices with limited display capabilities. Authors should not use pixel units with the "tty" media type.

tv

Intended for television-type devices (low resolution, color, limited-scrollability screens, sound available).

An example list of stylesheets (code from the TAG website – see Appendix IV):

stylesNS4.css – for Netscape Navigator 4

stylesPrint.css – for “print preview” pop-up window and printing

styles3.css – for text-only version of the site, screen readers, cell phones, text-only browsers, and unknown browsers

styles2opra-saf.css – for Opera and Safari browsers

styles2.css – for Netscape Navigator and IE (with IE hacks)

```
<?php
if ($client_browser == "old") {
    echo '<link href="/shared/css/stylesNS4.css"
rel="stylesheet" type="text/css" />';
} elseif ($display_mode == "print") {
    echo '<link href="/shared/css/stylesPrint.css"
rel="stylesheet" type="text/css" media="screen" />';
} elseif ($display_mode == "access"){
    echo '<link href="/shared/css/styles3.css"
rel="stylesheet" type="text/css" />';
} elseif ($client_browser == "alt") {
    echo '<link href="/shared/css/styles2opra-saf.css"
rel="stylesheet" type="text/css" media="screen" />';
} elseif ($client_browser == "new") {
    echo '<link href="/shared/css/styles2.css"
rel="stylesheet" type="text/css" media="screen" />';
} else {
    echo '<link href="/shared/css/styles3.css"
rel="stylesheet" type="text/css" media="screen" />';
}

echo '<link href="/shared/css/styles3.css" rel="stylesheet"
type="text/css" media="handheld, aural" />';
echo '<link href="/shared/css/stylesPrint.css"
rel="stylesheet" type="text/css" media="print" />';

?>
```

Ideally, it should have been created with 6 stylesheets; the 6th being called first and containing the basic elements common to all situations, with the other stylesheets containing additions and corrections. However, this structure, while less efficient, nevertheless causes the desired effect.

Switching stylesheets: (taken primarily from the A List Apart article Switching Stylesheets)

There are three kinds of external stylesheets: persistent, default, and alternate.

Persistent stylesheets are called with `rel="stylesheet"` and no title:

```
<link href="styles01.css" rel="stylesheet" type="text/css"
/>
```

Default stylesheets are called with `rel="stylesheet"` and a title:

```
<link href="styles02.css" rel="stylesheet" title="normal
styles" type="text/css" />
```

And alternate stylesheets are called with `rel="alternate stylesheet"` and a title:

```
<link href="styles03.css" rel="stylesheet" title="other
styles" type="text/css" />
```

When a page is first loaded, the persistent and default stylesheets are loaded, and the alternate stylesheets are ignored. When the user switches stylesheets, all stylesheets with the same title are loaded at the same time. However, the process of switching stylesheets is inherently a browser compatibility issue. Some browsers (like Mozilla) provide a drop down menu for switching between styles. Others (like IE) do not. To allow users in all browsers access to your alternate stylesheets, you will either need to create a script which will switch the stylesheets, or you can use the code that someone else already came up with. One such is available for free at <http://www.alistapart.com/articles/alternate/>, along with an explanation of how it works.

Appendix IV: Creating a Second Version of Your Site

Should you have two versions of your site? Well, that depends. If the layout of your site is simple enough, and you follow the guidelines laid out in this paper and elsewhere on accessibility, you may not need a second version. If, however, your site has a specific mood which is dependent on strange title fonts rendered as images and on low-contrast colors, an accessible version of the site will probably be necessary. If you determine that you will need a second version of the site, the following advice may be helpful.

~Don't duplicate content

It's hard enough keeping one website updated, let alone two identical sites. If you can make the transition from full-graphics site to accessible site simply by switching stylesheets, then simply put the second stylesheet in the header of your regular pages. This is the easier method, and should certainly be possible if you are already using a well-designed, CSS-rich site, with a flexible div-based layout. Of course, you only want to use this stylesheet some of the time. There are several ways of dealing with this. One example:

```
if ($accessible_mode) { echo "<link  
href="/shared/css/styles_access.css" rel="stylesheet"  
type="text/css" />"; }
```

`$accessible_mode` can be determined by parsing the current URL to see which version of the site is being browsed. There are also some tricks that can be pulled using `rel="alternate stylesheet"` and media types. More ideas and code for switching stylesheets can be found in Appendix I.

If you can't do the switch simply with CSS, then you will want to pull the existing content into the accessible pages. If you are already pulling content from a database, then this is fairly straightforward. Otherwise, you can use whatever server-side scripting language you prefer to parse your full-graphics pages for their content.

An example of a function to do this is in Appendix II.

~Use a second site root.

`http://access.mysite.com/path/to/page.php`

is much easier on your audience than:

`http://www.mysite.com/path/access/to/page.php`

`http://www.mysite.com/path/to/page.php?version=accessible`

`http://www.mysite.com/accessible.php?file=/path/to/page.php`

~Switch layout, not location

The link to switch site versions should take you to the mirror of the current page, not the site root. Often, your visitors will have come to your site through an external link, or a URL forwarded from a friend, or otherwise have no clue how to get back where they

were from the site root. Even if they do know, or manage to figure it out, they won't appreciate it. If your link is in a server-side include, or you just don't want to type out all those URLs, there is an easy solution: use server-side scripting. In PHP, for example, `$_SERVER['PHP_SELF'];` will give you the path of the current page from the site root.

Also on the subject of this link, it should appear twice. Once, in a logical place on the page for people using a browser to find, and once as the very first thing on the page for those using screen readers and cell phones to find. See **Hidden Elements – Customizing Content** for more information.

~Two words: “universal” and “accessible”

This *is* the accessible version of your site. You have no excuse whatsoever. Find a couple of accessibility checkers online, and make it pass everything. Don't cheat on the user checks, either.

Appendix V: PHP functions

How to deal with PHP's include function:

PHP has a function `virtual(path)`, which will include the file at *path* location, either relative or from the site root. However, this only works under Apache, and if the package is included in the PHP setup. Otherwise, you have to use `include(path)`.

The include function does not calculate absolute paths (eg. `/home/page.php`) from the site root, it interprets them as being from the root of the file structure on your hard drive. You could pass it a relative path name, or you could pass it a URL beginning with `http://`. However, there are potential problems with both those options. If you are calling the include function from within a server-side include, or if you want to copy-paste chunks of code without bothering to calculate relative paths each time, you don't want to use a relative path. If you call the include function with a URL, you may run into scoping errors with variables, as all the PHP in the other page will be executed first, and then plain html will be included in your page. This makes it impossible, for example, to include a file of PHP functions you wish to call on your page.

The solution: copy the following code into the top of each webpage, which will automatically calculate the relative path to the site root.

```
function siteRoot () {
    $numberOfSlashes = substr_count($_SERVER['PHP_SELF'],
"/");
    $retval = "";

    for ($i=1; $i < $numberOfSlashes; $i++) {
        $retval .= "../";
    }
    return $retval;
}
```

You can then call server-side includes like the following:

```
include(siteRoot()."path/from/site/root.php")
```

An example of a function returning `$client_browser` information from parsing `$_SERVER['HTTP_USER_AGENT']`.

```
function browser()
//returns a string depending on the type of browser: "new"
if equivalent to new Netscape, "alt" if equivalent to opera
and "old" if equivalent to old Netscape
{
$browser_info_string = $_SERVER['HTTP_USER_AGENT'];

if (strpos($browser_info_string, "Netscape")) {
    $browser_name = "Netscape";
    $browser_version = substr($browser_info_string,
    strpos($browser_info_string, "Netscape")+9);
}
else if (strpos($browser_info_string, "Safari")) {
    $browser_name = "Safari";
}
else if (strpos($browser_info_string, "Camino")) {
    $browser_name = "Camino";
}
else if (strpos($browser_info_string, "Opera")) {
    $browser_name = "Opera";
}
else if (strpos($browser_info_string, "MSIE")) {
    $browser_name = "Internet Explorer";
}
else {
    $browser_name = strtok($browser_info_string, '/');
    $browser_version = strtok(' ');
}

if (($browser_name == "Netscape" && $browser_version < 6)
|| ($browser_name == "Mozilla" && $browser_version < 5)) {
    return "old";
} else if (($browser_name == "Safari") || ($browser_name ==
"Opera")) {
    return "alt";
} elseif (($browser_name == "Internet Explorer") ||
($browser_name == "Netscape" && $browser_version >= 6) ||
($browser_name == "Mozilla" && $browser_version >= 5)) {
    return "new";
} else if ($browser_name == "Camino") {
    return "unknown";
} else {
    return "unknown";
}
}
```

If you then put the following line either at the top of your page, or in an included file with all your variable declarations, you can, for example, use `$client_browser` to determine which stylesheet to load.

```
$client_browser = browser();
```

An example of a PHP function to parse a file, and include the main content. It assumes that the beginning of the content is marked by a named anchor (which would also be used for the “skip to content” link – see Hidden Elements – Customizing Content), and that the end of the main content is marked by a comment “end content”.

```
function getPageContent ($url) {
    $file = fopen ($url, "r");
    $pageContent = "";
    if (!$file) {
        echo "<p>Unable to open the associated file at
http://$rootVar$url.</p>";
        exit;
    }
    while (!feof ($file)) {
        $line = fgets ($file, 1024);
        $pageContent .= $line;
    }
    fclose($file);
    $startpos = strpos($pageContent, '<a name="content" />');
    $endpos = strrpos($pageContent, '<!-- end content -->');
    $pageContent = substr($pageContent, $startpos, $endpos -
    $startpos);

    echo $pageContent;
}
```

Bibliography

Accessibility Articles and Tutorials: A List Apart. May 4, 2004. A List Apart.
<<http://www.alistapart.com/topics/accessibility/>>.

Apache Tutorial: Introduction to Server Side Includes. May 12, 2004. The Apache Software Foundation. <<http://httpd.apache.org/docs-2.0/howto/ssi.html>>.

CSS Articles and Tutorials: A List Apart. May 4, 2004. A List Apart.
<<http://www.alistapart.com/topics/css/>>.

A List Apart. May 4, 2004. A List Apart. <<http://www.alistapart.com/>>.

Stig Sæther Bakken, et al. *PHP Manual*. May 12, 2004. The PHP Group.
<<http://ca3.php.net/manual/en/index.php>>.

Vischeck: Home. May 4, 2004. Vischeck. <<http://www.vischeck.com/>>.

W3Schools Online Web Tutorials. May 9, 2004. W3Schools.
<<http://www.w3schools.com/>>.

Welcome to Bobby Worldwide. May 4, 2004. Watchfire.
<<http://bobby.watchfire.com/bobby/html/en/index.jsp>>.

References

508 Universe Glossary. May 13, 2004. Federal IT Accessibility Initiative Training.
<<http://section508.gov/508/asp/glossary.asp>>.

Media Types. May 4, 2004. W3C World Wide Web Consortium.
<<http://www.w3.org/TR/REC-CSS2/media.html>>.

Sowden, Paul. *Alternative Style: Working With Alternate Style Sheets*. May 4, 2004. A List Apart. <<http://www.alistapart.com/articles/alternate/>>.

TAG Home. May 4, 2004. Teaching and Academic Growth, UBC.
<<http://www.tag.ubc.ca:888/>>.

508 Universe Glossary. May 13, 2004. Federal IT Accessibility Initiative Training.
<<http://section508.gov/508/asp/glossary.asp>>.