The Top 5 Roadblocks to Accessibility

Karl Groves, Director of Web Development
User-Centered Design, Inc.
20548 Deerwatch Place, Ashburn, VA 20147
Phone/Fax: (703) 729-0998

Introduction

Accessibility is often the last thing on a web designer's mind when creating a website. This is not a trait unique to newbies or people working on a personal page. It is also a trait common to professional web designers (large and small) and even multinational corporations. In fact, most web designers have no clue about what accessibility is.

Many who do know what accessibility is will often treat it as though it is brain surgery. Nothing could be further from the truth. In fact, all that it takes to create an accessible site is some forethought and understanding of the kinds of mistakes you're likely to make so you know how to avoid them.

What is Accessibility?

For those with no knowledge of accessibility, I usually like to use the following analogy: The building you work in probably has at least one handicapped parking space. If it has more than one floor, it probably has an elevator or escalator (or both). It has railings on the stairs, and probably has a dip in the curb. Your workplace has these items in order to facilitate access to the business by disabled customers and/or employees. Accessible web design is nothing more than an electronic equivalent of this effort toward equal access to your resources.

> ***The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect*** **- Tim Berners Lee, inventor of HTML and the HTTP protocol.**

Why A Complete Focus is Needed

I will not venture into any discussion of the morality of why you should make your site accessible. Simply put, any argument (other than sheer ignorance) against accessible design is simply not worth having. Instead, let's focus on what sorts of special needs a user may have that can create problems in interacting with a website:

- **Visual impairments**, which can manifest themselves in color blindness, poor eyesight or complete blindness
- **Hearing impairments** in a wide range of potential severities
- **Mobility impairments** ranging from arthritis, multiple sclerosis, Parkinson's disease, paralysis, or other motor-control disorders
- **Cognitive impairments** such as cerebral palsy, Downs syndrome, Alzheimer's, dyslexia, or learning disorders
- **Seizure disorders** like epilepsy

Ultimately, a complete approach is what is needed in order to create an accessible site. You cannot predict the user's needs or the severity of those needs. Misguided approaches such as browser detection or text-only alternatives betray a lack of understanding this fact. Text-only alternatives can only provide an accessibility solution for those with vision disorders, thereby disregarding millions of other disabled users with different needs. Browser detection is destined to fail, as disabled visitors use a wide range of adaptive technology, including user-agents that identify themselves as one of the major brands. Moreover, there is no way to detect the user's pointing device or any of the other items in the wide range of settings used to compensate for their special needs.

Avoid The Most Common Mistakes

In their ignorance and disregard, web designers are most likely to commit five mistakes that are roadblocks to accessibility. Avoiding these mistakes will take you far down the road toward accessible design. Your site won't be perfect just by avoiding these five mistakes, but committing these mistakes will likely mean the site is completely unusable a wide range of users with special needs.

1. Dependence upon client side scripting to present navigation or important content

Among the ways you can make a website completely inaccessible to many types of impairments at once is to use client-side scripting in such a way that the entire site fails to work for users who have client side scripting turned off or who are using adaptive technology that does not recognize client side scripting at all.

Such items would be things like:

- Event handled dynamic content (processed client side)
- Fly-Out (aka DHTML) menus
- Drop down menus that require the onChange event handler to operate
- Popup windows that do not work without JavaScript

Reliance on scripting will have one of four possible results:

1. The site will load but navigation will be impossible (it will either display and not work, or often just not display)
2. Substantial portions of content will not display
3. The site will load but absolutely no content will display
4. In an attempt to circumvent "usability" problems created by their use of scripting, they will cluelessly detect the presence of scripting support by the browser and redirect the user to a dead end page, which attempts to teach you how to upgrade your software.

I consider any of these results to be a complete accessibility failure. Unfortunately, even websites for multinational corporations and government entities fail under such criterion.

The following sites lead to dead end pages asking you to upgrade your software

- New York Stock Exchange
- Time Magazine
- H&R Block
- Travelocity
- General Motors
- Toyota
- Porsche

The following sites result in completely inoperable navigational elements

- Washington DC Water and Sewer Authority
- Volvo
- 1-800-Contacts
- The American Automobile Association
- Verisign

The following sites will load, but the content will actually not display at all

- Chrysler
- Ford
- Chevy
- Visa

The following sites have a Swiss-cheese effect with content sporadically missing on screen

- Dodge
- Jeep
- Baltimore Ravens Football Team

The solution for these problems is extremely simple. **Do not make anything completely dependent upon scripting.** Interactive elements should add to the enjoyment of the website, not detract from it as often happens.

I recommend against using "fly-out" (aka DHTML) menus and dropdown menus for primary navigation.
While it is possible to create them so that navigation works with scripting turned off, that will only solve one set of problems. As a general usability issue, DHTML navigation is often described as "slippery" by able-bodied users, and can certainly create frustration among those with motor impairments.

At the very least, you'll want to ensure two things when using fly-out menus for navigation:

- The menu (or a <noscript> alternative) actually displays when scripting is off/ unrecognized
- The first link in the menu functions and leads to an actual destination

Create drop downs that function without scripting.
You can choose one of about three approaches to this:

- **First,** it is OK to operate from an onChange event on these, provided of course, you supply a submission method that doesn't rely on scripting - in other words, a button and a server-side script
- **Second,** if you really want to get tricky, write the button based upon detection of scripting support. It's a bit too much needless effort in my opinion, but some might not think so.
- **Third,** write the list of links in your <noscript> element that have the same destination as the DHTML menu. The majority of users will get the nifty dynamic menu and those without scripting support will at least get an operational list of links that go to the same place.

Other navigation
I have even seen plain text links rendered inoperable without scripting support. The natural solution is just to not do that. There are no neat workarounds necessary.

Event Handled Content
Event-handled content is a bit trickier. In general, my argument is that there is nothing that client-side scripting can do that cannot be done with server-side scripting such as PHP. For small websites, this is the best approach. For a large website getting thousands of hits a minute, the back & forth trips to the server for each little request can become an overwhelming burden on the server. In the case of large sites, developers need to work with the person responsible for interaction design to come up with a way to avoid such a reliance on client-side scripting.

Unfortunately, event-handled content is also likely to result in a blank, or Swiss-cheesed screen if handled client-side. In such a case, you're probably better off dodging the issue altogether by any means necessary. I'd rather have a website that worked properly for everyone than a website that had interactivity dependent upon something that completely failed for a potentially large number of people.

Popup Windows
For popup windows, you will need to make sure the link operates regardless of scripting support. If you're using popup windows for ads, they're probably tied to an onload (or onunload) event. That's fine, leave it there. Nobody except you and your advertiser likes them anyway; so don't bother making them accessible. But if you're using popup windows for supplementary content, you should not use "#" or "javascript:" as your hypertext reference. Use a real link, and set your JavaScript code to "return false". The link will operate properly regardless of scripting support. (See my article - "A More Accessible Pop-up Window")

2. Improper use of markup/ Invalid markup

Much is thrown around in discussion groups about the value of valid markup - even by yours truly. Valid markup is easy to create, despite the fact that most people manage not to achieve it. The validity of the page's markup is important - from an accessibility standpoint - because adaptive technology may not be as forgiving of botched markup as the major browsers. What is more challenging, even among those who have valid pages, is the *proper* use of markup.

Web designers say much about interactive, dynamic, and database-driven content. Ultimately, it doesn't matter one bit about how amazing the programming is that drives the site - what gets sent to the visitor's browser is HTML.

HTML is a markup language

Simply put, it was created with the express purpose of describing the structure of the information, NOT the presentation. Many web designers will use HTML to dictate the presentation by using deprecated or proprietary attributes, or by improperly using some elements to dictate presentation:

- Deprecated elements include , <font>, <menu>, <s>, <strike>, <u>
- Deprecated attributes include align (in many cases), alink, background, bgcolor, height and width (for table cells), hspace, vspace, and many more
- Improper use of elements to dictate presentation would include use of tables for layout, use of <tbody> in layout tables, using <pre> or <blockquote> to control text positioning, or not associating data table cells with headers and defining relationships between headers and data rows/ columns.

CSS is for presentation

In every case where presentational effect is needed, you should use CSS to achieve the presentational effect. I cannot think of many presentational effects achieved by HTML attributes that do not have CSS equivalents. Some CSS properties have spotty support by major browsers, and for this reason you should always keep this issue in mind. This spotty support does not mean you should lazily resort to HTML's presentational attributes. Rather, it means you should either come to grips with the myth of "looking the same" to everyone or find another presentational effect to chase after. In my opinion, resorting to a workaround means the design is flawed, not the spec.

Use appropriate markup

Use appropriate markup for the document, before even involving yourself in presentation. This includes things like headings, paragraph tags, and break tags. Are you using <h1> to make a large, bold word? Are you using <b> to create a heading? Are you using to create a break between paragraphs? Are you using <blockquote> to indent text? If so, you're using these elements improperly. Each of these elements has a purpose in dictating the structure of the document. For instance, <h1> is the element meant to dictate the heading of the document.

<h2> is for sub-headings, such as main sections of text. <h3> is meant as a subheading of a section. <p> is meant to mark a paragraph, and so on. For more information on what the purpose is of the various HTML elements, visit the spec yourself. http://www.w3.org/TR/html401

To ensure greater accessibility:

- Use HTML to define *structure* only
- Use CSS to define *presentation*
- Always use the most appropriate element for the content. Do not misuse elements or attributes just to achieve your aesthetic vision.

3. Device dependence

While many people tend to think of accessibility as efforts to make a site usable for the blind, this is simply not true. Creating a site that is dependent upon any piece of hardware is destined to fail for users with a wide range of potential special needs - most notably, the blind and persons with motor impairments. Device dependence is a bad practice from a general usability standpoint as well. More and more users are accessing the Internet with devices such as cell phones and PDAs. *Creating device dependence can serve to alienate far more people, disabled or not.*

Device dependence can be seen as:

- onClick/ onMouseover/ onMouseout or other event handlers that rely on the user having a mouse in order to operate a link or form control.
- Server-side image maps that do not have a corresponding set of redundant links
- Navigation or forms that do not have a logical tab order

I don't necessarily frown on device dependence in cases where the dependence is only an issue for things like rollovers, image highlighting, text color changing, or some other decorative change. The issue comes when the device dependence combines with a reliance on scripting (#1, above) in that the event handler is tied to something like form submission, content presentation, or a calculation. In such cases, as I also said above, these things would probably be best-handled server-side. At the same time, a more accessible event handler choice might be best as well, using handlers like onFocus, onBlur, onChange and onSelect.
In cases where device-dependent event handlers are needed, you should use redundancy to handle the interaction so that it at least works for people using either a mouse or a keyboard. When you use onMousedown, combine it with onKeydown. When you use onClick, combine it with onKeypress: `<button onClick="window.alert('See? Isnt it Cool?')" onKeypress="window.alert('See? Isnt it Cool?')">Click The Button</button>`

Ultimately, you would still be far from fully accessible, so long as there's any level of device dependence for access to any important content, navigation, or interactivity. A far better approach would be to refrain from creating site features that require the user having any specific piece of hardware. Remember one thing above all others: For every specific special need there is just as wide of a variety of methods used by the visitor to compensate for it. This includes input devices just as much as it does output devices. Dodge the bullet altogether by creating the site so that it doesn't matter what the visitor uses to interact with the site.

Some creative and knowledgeable designers also use JavaScript's document.write() function to write these features to the browser. In these cases, they also wisely provide a redundant text alternative via the <noscript> element.

4.Lack of/ improper use of alternative text for graphic and multimedia elements

Images, movies, sounds, and Flash are inherently inaccessible. The presentation of any of these items is likely to result in your site being completely useless to the disabled unless you supply an alternative text equivalent.

- The blind cannot see images or Flash. Using a picture or Flash to present important content will mean that a blind user cannot experience the same content.
- Deaf visitors will be unable to hear the sound that accompanies an MPEG or AVI movie.
- Those with motor impairments may be unable to interact with Flash.

Circumventing any of these issues is painfully simple, really. An intelligent application of alternative text to any graphical/ multimedia item is all that is needed. The issue comes into what is defined as an "intelligent" application.

As simple as it is, it seems to me that a lot of people really make a mess of their use of alternative text. So let's backtrack to the purpose of alternative text: To present a textual alternative to those who cannot see the picture or movie or hear the sound. But it is important that we make sure that the text alternative is meaningful to the overall understanding of the page content.

Images

If the image presents no contribution to the content of the page for those who can see it, then its description won't either. But, this doesn't mean that the image doesn't need an "alt" attribute at all. Some text-only browsers will show [image] on the screen for an "alt-less" image, or [link] on the screen for an image that is used as a link. In these cases, the aural equivalent is basically the same - sometimes reading aloud the image name or link destination as a way to compensate for the basically useless image.

In cases where an image is only decorative, the alt attribute should be empty. If the image is a link but provides no real context (like a button or something) then

the alt attribute should list the destination and possibly a description of the destination.

In cases where an image does provide important content, it is important to have an alternative text equivalent. Often this is simply an alt attribute. However, if the alt attribute will be long, it may be most appropriate to use the *longdesc* attribute. Longdesc is most often a link to a specific page that contains the description. Sometimes people create a page for all descriptions and the longdesc will be a named anchor in that page.

In any case, the alternative text must be meaningful and take into consideration the person who is likely to need it. I am of the opinion that the vast majority of images should have empty alt attributes, as most descriptions will provide no additional understanding of the content.

Movies (MPEG, AVI, etc)
Because movies are inaccessible to both those with vision impairments and hearing impairments, it is especially important to provide a text alternative. This will often mean captioning the file and/ orproviding a text transcript.

"A transcript is a text or HTML file with all of the important dialogue and/or narration. Captions are a part of the video file which synchronizes the dialogue and narration with the video itself." (WebAim)

Doing so properly will require extensive forethought to make it right including choosing between which format you'd like to use. There are also choices between whether you'd like to use a closed or open caption.

Web accessibility resource WebAim has such a good overview of Multimedia Accessibility that I would rather send you there (http://webaim.org/howto/captions/) than get into too much detail here. You're best advised to have a good understanding of multimedia accessibility *before* creating the movie file. Choose the format wisely based upon the technology and the requisite effort to ensuring accessibility. Attempting to make a movie file accessible after the fact is doomed to leave some visitors still unable to use it.

Flash
Macromedia Flash has been one of the coolest multimedia tools to ever hit the Internet. But up until recently, it was riddled with accessibility, general usability, and even development issues that prevented it from growing to its full potential in the marketplace. With its recent ability to integrate with databases and server-side scripting, as well as recent efforts toward greater accessibility, Flash is destined to continue growing in presence among the web. Yet despite the

advancements of the most recent version of Flash, it is still not a product I'd use for creation of anything other than multimedia presentations.

- Flash still requires the presence of the Flash plug-in. Users without the plugin will get nothing
- Interacting with navigational elements is often problematic with Flash designs (i.e. determining what is a link/ icon, links moving around, etc.)
- Inability to present an alternative text equivalent such as a caption
- Unfamiliar interface; inoperable browser controls.

For these reasons, use of Flash can be an out-and-out accessibility failure despite using the most recent version with its "Accessibility Features". At most, I would use Flash as a lightweight alternative to MPEG/ AVI/ Real Media. At that point, it would require the same efforts toward captioning and transcription as those movie formats require.

Sounds

Ultimately, sounds can be made more accessible in the same way as images. Sound is often used by newbies for background sound or for interactivity (i.e. rollover sound). Most professionals have realized that such practices are not user-friendly and avoid them. But, sound files such as MP3 are sometimes used to provide supplementary content.

Sound is an especially useful addition to content that covers information that cannot be accurately presented without supplemental sound. For example, a sample pronunciation of a foreign word on a site that has foreign language tutorials, or a sample sound of a piece of music being played for a guitar tablature website.

Like images, providing a plain text alternative to sound must be weighed against the potential usefulness of the particular text alternative. Is a deaf user likely to benefit from a description of a guitar being played? No. Will they benefit from a transcript alternative to Martin Luther King's "I Have A Dream" speech? Definitely. Like anything else, you should make every reasonable and intelligent effort toward providing an equal understanding of the content.

5. Improper creation of forms

Forms are utilized by companies to allow a site's users to submit applications, register for things like announcements and members' areas, or to actually contact the company. Any inability to interact properly with the form's elements is an accessibility failure in my opinion. Forms are often designed with no regard for people with special needs. For example:

- Use of JavaScript for validation or even submission means that the form does nothing for people with browsers that don't recognize JavaScript.

- Visual reference to required entry: "Fields in red are required," means nothing to the blind and colorblind
- Not associating elements with their labels will often mean confusion for those who cannot see the form elements.
- Not placing the label adjacent to its element will have the same effect.
- Using tables to position form elements can magnify some of the above problems.

Because of the potential return from using forms (i.e. sales leads, registrations, applications, etc.) it is vitally important that you ensure that all people are able to interact with the forms on your site. The greatest issue faced by persons using screen readers is in knowing how to interact with the form's elements.

> *When we talk about the accessibility of forms, we are usually referring about their accessibility to screen readers and the visually impaired. People with other types of disabilities generally are less affected by "faulty" forms that are missing some of the HTML accessibility features. I should note, however, that everyone benefits from a well-organized form, especially those with cognitive disabilities. Visual layout can be important to those who have sight. The less explanation that a form needs, the better.* - WebAIM

Place form labels next to their corresponding form controls
Simply put, if you're after the user's name, then place "Name" next to the input element for the name value.

Next, use the <label> tag to explicitly identify what is the label

```
<label for="name">Enter Your Name Here</label><input
type="text" id="name" name="visitor_name" />
```

In the case of multiple-choice items such as radio buttons and checkboxes, you should group them using the <fieldset> tag.
Then use a "legend" for each field set using the <legend> tag and provide a <label> for each element.

```
<fieldset><legend>What Operating System Do You Like
Best?</legend>
<input type="radio" id="win"
name="windows"><label>Windows</label>
<input type="radio" id="mac"
name="macintosh"><label>Macintosh</label>
<input type="radio" id="lin"
name="linux"><label>Linux</label>
<input type="radio" id="uni"
name="unix"><label>Unix</label>
</fieldset>
```

For select menus, use the <optgroup> element to group related choices. Again, explicitly defining the label.

```
<label for="breed">What kind of dog do you have</label>
<select id="breed" name="kind_of_dog">
<optgroup label="Sporting Group">
<option value="1">German Shorthaired Pointer</option>
<option value="2">Chesapeake Bay Retriever</option>
<option value="3">Golden Retriever</option>
<option value="4">Labrador Retriever</option>
<option value="5">Cocker Spaniel</option>
</optgroup>
<optgroup label="Working Group">
<option value="6">Boxer</option>
<option value="7">Bullmastiff</option>
<option value="8">Mastiff</option>
<option value="9">Rottweiler</option>
<option value="10">Saint Bernard</option>
</optgroup>
<optgroup label="Non-Sporting Group">
<option value="11">Boston Terrier</option>
<option value="12">Bulldog </option>
<option value="13">Dalmatian</option>
<option value="14">French Bulldog </option>
<option value="15">Poodle </option>
</optgroup>
</select>
```

Always provide a button to submit forms.

Don't use JavaScript to automatically submit forms via an onChange event unless you also provide a functioning button so that those without JavaScript can submit the form as well.

Ultimately, it is the information that matters

What it all boils down to is ability to access information. Accessibility is not "in addition to" what you already have, and it should not take away from it either. A complete focus on the user (all users) stems only from an understanding of why people are coming to your website: Information. Accessibility is the effort toward providing equal access to the information to all - regardless of the methods they use to access it. It requires nothing more than an understanding of the potential roadblocks and how they can be avoided.