

Not everyone using the Internet today enjoys the same kind of user experience as does the average person. Consider the following:

1. An estimated 20 percent of the U.S. population has some level of disability.
2. In 2006, the National Federation of the Blind (NFB) sued Target Corporation for failing to provide accessibility on their consumer shopping Web site.
3. All government funded Web sites, and many companies that follow good corporate citizenships, mandate that Web development follow standards protocols and section 508c of the U.S. Rehabilitation Act compliance.

So traditionally, accessibility refers to persons with physical disabilities. In the web development world, accessibility also refers to the **availability** of Web content in much broader terms, including

but not limited to all the various Internet enabled devices on the market, all the older (and future) versions of these devices, and systems that catalog and index Web content.

Unfortunately, many site owners and developers regard accessibility as a cost and inconvenience that does not necessarily translate into more return on investment (ROI). But what if designers, developers, and project managers approached Web site development in a way that **can** provide universal content accessibility without spending too much extra time or money? Would it be worth it? After all, why lock out any potential consumer because of a physical impairment, or a technology device constraint, or a site's presentation requirements? As a society, are we not responsible to each other in terms of universal and equal access to

Problem:

Creating "accessible" Web presences is typically regarded as a secondary need because many clients and developers believe that it will increase cost without necessarily improving ROI (return on investment).

Solution:

Implementing simple yet effective techniques, such as:

- writing semantic, hierarchical mark-up structures
- controlling text zooming options
- implementing basic keyboard navigation rules

will make content more universally accessible without significantly adding to the cost of development.

Benefit:

Creating a software development standard that accommodates all users and user agents on any device will increase content availability, improve bottom line ROI, and help secure Web properties under Section 508c of the U.S. Rehabilitation Act.

information? Philosophical arguments aside, this white paper will show, using real examples, how a modest level of accessibility can be achieved using simple techniques. In addition to the basic accessibility features that should always be present on any Web page (for example the screen reader friendly “skip to main content” option and ALT attributes for images), the techniques illustrated below will address both interpretations for accessibility in ways that improve ROI for any Web-based property.

Semantic, Hierarchical Mark-Up Structures

One of the most important client-side development techniques for achieving accessibility “enlightenment” involves writing clean, semantic HTML structures. This is easier said than done of course, given that tables-based HTML grids are still pervasive despite a concerted effort towards using CSS layouts.

Essentially, assistive technologies such as screen readers ignore* the typical references for style sheets, take a snapshot of the document object model (DOM), and read the content in order, from top to bottom. Thus a well constructed content hierarchy will present a more logical user experience for visually impaired users. The key is to wrap content with semantically meaningful HTML tags that describe what the content’s role is (descriptive tagging) and to avoid using the most convenient HTML tags to create a desired look or position.

* The W3C is currently working on a standard for a screen reader friendly CSS keyword reference. See: <http://www.w3.org/TR/css3-reader/>.

For example, in the development of Washington Mutual’s (WaMu) new transactional banking Web site, our task was to develop a forward thinking user interface that provided all the bells and whistles for normal Web users while paying close attention to building accessible HTML structures. Notice in the example to the right (which shows the header rendered with and without CSS styles), we used descriptive tagging to produce clean, unordered lists for groups of links and semantic form tag constructs for site search.



Semantic Submission Forms

Properly built submission forms come with their own set of useful HTML tags. Unfortunately they are rarely implemented because of their relative obscurity. They do, however, provide a wealth of information for accessibility needs. We used the following three tags to improve semantic meaning for WaMu form fields:

FIELDSET

The **fieldset** tag, which should only be used in conjunction with form elements, groups similar sets of input tags under the same wrapper. It is especially useful for long forms with multiple sets of field elements. Since the **fieldset** tag is a block level element, it can also be used as an attach point for CSS styling and JavaScript behaviors, thus avoiding such problems as *divitis* (the practice of using too many DIV tags to render a user interface component).

LEGEND

The **legend** tag, which should only be used in conjunction with the fieldset tag, defines the caption for a fieldset. For the WaMu redesign, we used CSS to hide the **legend** tag so we could design custom labeling for forms, knowing that it would properly define the fieldset for non-CSS devices.

LABEL

The **label** element is arguably the most important accessibility related tag for forms. It binds a text description to an input element through matching attribute values. In terms of user interaction, it gives the added advantage of applying focus to the input when the label is clicked.

In the case of the account login console for WaMu, the normal Web user interacts with a DHTML enhanced form (top right image), while the screen reader still receives an understandable interaction model (bottom right image), all from the same mark-up. The final HTML for WaMu’s login form includes a single **fieldset**, a **legend** describing the contents of the **fieldset**, and **label** tags paired with their respective inputs.

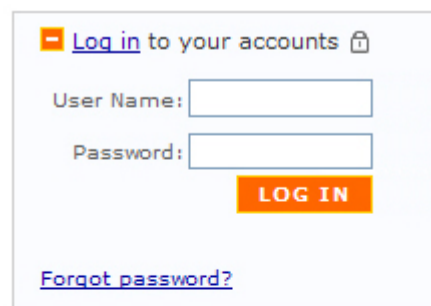
Benefits

In addition to providing assistive technologies with logical content structures, writing semantic HTML also improves SEO (Search Engine Optimization) by providing content that is formatted according to its functional role. For example, H1, H2, H3 header tags should describe content section headers so that search spiders can easily identify and index their relevance.

Other beneficiaries of semantic markup include various types of Internet enabled devices, like mobile phones and PDAs, that don’t necessarily interpret the presentation layer the same way a traditional desktop Web browser does.

Text Zooming

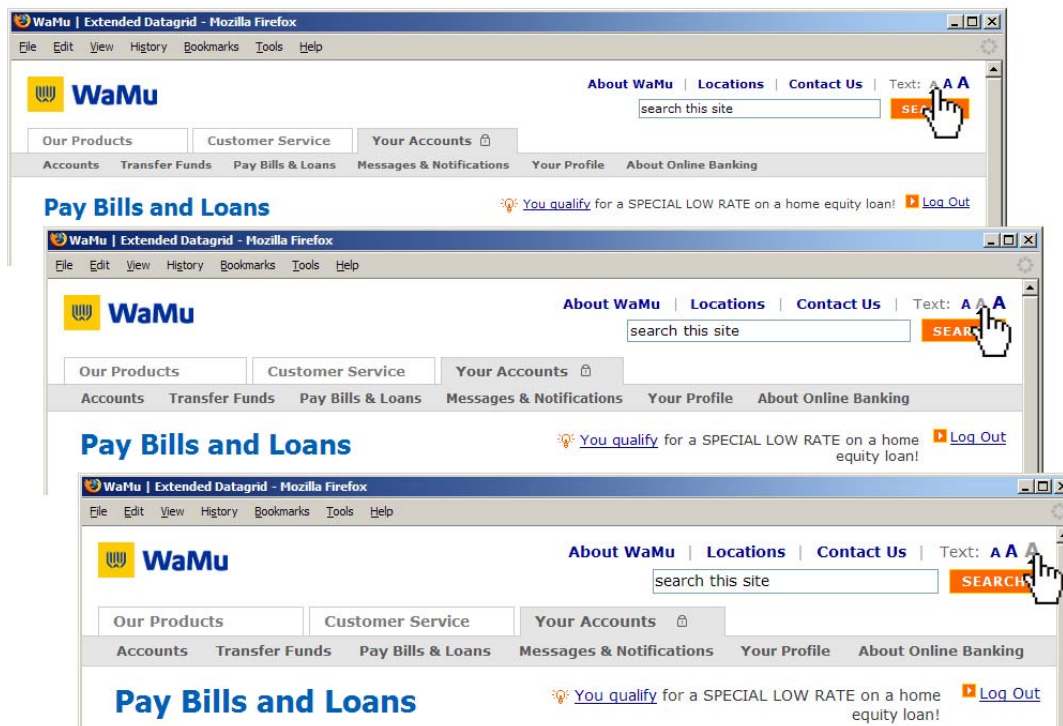
Text zooming is essential in creating readable content for people with visual impairments. Most people



characterized as visually disabled are usually not completely blind but need to adjust text to a readable size.

Most modern Web browsers come equipped with a text zooming feature as part of the native browser engine. The biggest challenge facing user interface designers and developers is in creating a flexible UI with components that can react and adjust as font sizes increase or decrease. Additionally, browser manufacturers have thus far handled text zooming differently, making it more difficult to control. Gecko-based browsers (Firefox, Netscape, etc) by default allow limitless text zooming in either direction, while Internet Explorer (version 6) allows only two levels of zooming in either direction (and then only if the font values have been set using relative measurements). Internet Explorer (version 7) has tried to alleviate the problems by using a bi-cubic interpolation algorithm that not only increases the font size but everything else on the page as well. The idea is to maintain the intended design of the Web page. Although not perfect (the horizontal scroll bar often appears), IE7 has come a long way towards addressing accessibility needs for its users.

For the WaMu redesign, the client wanted to augment the default browser-based text zooming with an embedded text zooming widget found in the upper right corner of the user interface. This approach seems redundant since people with visual impairment requirements know how to use the browser's text zooming feature. Nonetheless, the widget shows up prominently on every page and allows for a controlled approach by defining which parts of the page will react to text zooming, thereby preserving the intended UI design:



Benefits





Obviously the main benefit of text zooming is readability. According to Jakob Nielsen*, a well known usability guru, seniors are one of the fastest growing segments of Internet users. They typically have money to spend, time on their hands, and may have difficulty moving around. This archetype Web user tends to have the visual readability needs associated with age. Addressing those needs can attract loyal customers who appreciate the effort.

* See <http://www.useit.com/alertbox/high-roi.html>

Keyboard Navigation

Another great way to add accessibility to a Web site without breaking the bank is by proactively controlling how the user navigates using the keyboard. The TAB key is most commonly used for jumping from item to item. Tabbing usually works by default, starting at the top, but may not follow a logical path, depending on how the mark-up hierarchy is laid out.

In the case of WaMu, we assigned a specific TAB order on pages that have the potential to have large numbers of form fields. For example, the Pay Bills and Loans page, which shows a summary of all payees on a person's bank account, allows users to enter a payment amount for each payee. We assigned the TABINDEX attribute dynamically to each payee row and further reinforced the idea by dynamically applying a visual cue (gray background) on each row as the user tabs down the page:

Payee ▾	Amount	Estimated Delivery <u>Start On</u> → <u>Deliver By</u>	Last Payment
Allied Waste Services	\$ <input type="text"/>	 4 business days	\$75.23 07/20/2006
Pacific Gas & Electricity	\$ <input type="text"/>	 06/01/2006 → 06/04/2006	\$12.39 06/19/2006
 Consumer Loan - 6	Make a loan payment		
Touchstone Gym, Inc	\$ <input type="text"/>	 2 business days	\$85.62 07/20/2006

The Pay Bills and Loans table uses JavaScript to highlight the row background when the user either clicks on the input field or uses the TAB key to navigate.

Another example of controlling the TAB key can be found on the account login console, where we used JavaScript to activate the console's drop-down visibility (with focus automatically placed on the first field) when the user presses the TAB key for the first time. This allows the user to log into the banking application without ever having to use the mouse at all. The user simply tabs to initiate the drop-down, enters their credentials by tabbing through each field and finally presses ENTER to submit the form. The JavaScript function used is simple:

```
function handleTab(evt) {
    var e = evt || window.event;

    if (e.keyCode == 9) {
        if (getElementsByClass('form','loginbox')[0]) {
            showLoginBox(getElementsByClass('form','loginbox')[0]);
        }
        if (getElementsByClass('input','usernamefield')[0]) {
            getElementsByClass('input','usernamefield')[0].focus();
        }
        else if (getElementsByClass('input','passwordfield')[0]) {
            getElementsByClass('input','passwordfield')[0].focus();
        }
        return false;
    }
}
```

Progressive Enhancement, AJAX, and Validation

Progressive enhancement and graceful degradation are important concepts that address the availability of Web site content and features based on the user agent's (and users themselves) abilities. The key is to keep the three presentation layers (HTML, CSS, and JavaScript) separate from each other, so that Web browsers that can consume the latest technologies will take advantage of them (progressive enhancement), while those that cannot will simply ignore or will not receive them (graceful degradation). Thus, **content** is available to everyone, no matter which device is used or special need is required.

The Problem with AJAX

Recent research shows that there is no standard methodology for creating accessible AJAX applications. Essentially, the problem lies with many screen reader's inability to recognize the *onreadystatechange* event (AJAX's way of handling asynchronous server requests). Several workarounds have been proposed; however, none work uniformly across the various screen readers and Web browsers on the market.

Several options are available for those who want to try implementing an accessible AJAX application. Since screen readers take a snapshot of the DOM and place it in a virtual buffer, the key is in devising a way to inform the screen reader that a change has occurred on the page.

1. Use JavaScript to set *document.location = #response* after the responseText has been received and the page updated. This option only works for some screen readers.
2. Use JavaScript to set *response.focus()* on the changed content. Set the *tabindex* of the HTML element in question to -1, which allows focus to be set on elements that are normally not allowed to receive focus. IE and Firefox both support this technique (Safari does not).

3. Use JavaScript to place the response in an alert box: `alert(request.responseText)`. This method works for most screen readers, but obviously it is a rather obtrusive approach. One workaround is to provide screen reader browsers with a checkbox option for receiving AJAX responses in an alert window.

Unfortunately, there isn't a good way (yet) to build accessibility into AJAX applications, at least not in a uniform and consistent way. For additional information, the following three articles aptly describe the current state of AJAX and accessibility:

» **[AJAX and Screenreaders: When Can it Work?](#)**

James Edwards

» **[Making Ajax Work with Screen Readers](#)**

Gez Lemon and Steve Faulkner

» **[AJAX and Screen Readers – Content Access Issues](#)**

Steve Faulkner

Validation

Code validation plays an important role in accessibility. The W3C provides an HTML validator that is quite useful for *syntax* validation. However, one of the biggest misconceptions about the W3C validator is that many believe it will also set the bar for accessibility. The truth is that a Web page that validates against the W3C validator is not necessarily more accessible, nor is a Web page inaccessible if it does not pass W3C validation.

It is important to use the correct tools to accomplish the right tasks. A great (and free) online tool for accessibility validation is Watchfire's WebXACT, which is similar to the W3C validator in that it scans a Web page's source code but produces results with criteria that address accessibility and privacy needs in particular.

Conclusion

As this white paper has shown, Web accessibility can be achieved with small yet effective adjustments to development methodologies. Until standardization between screen reader manufacturers becomes more prevalent, as it did with Web browser manufacturers, true accessibility for the Web will remain a moving target. In the meantime, it is still important to define the role of accessibility in the software development cycle in terms of client expectations and end user satisfaction. This requires the participation of not only developers, but also user experience designers, program managers, and quality assurance testers. Like all successful Web product launches, teamwork, communication, and understanding will produce results that will meet traditional accessibility needs while providing content accessibility to the maximum number of Internet-enabled devices and systems.

About the Author



As a presentation layer architect, graphical alchemist, and Web Standards advocate for Avenue A | Razorfish, Frederic Welterlin's experience and areas of focus include designing, architecting, and programming client-side templates, providing interaction and technical recommendations, and developing standards and processes for best of breed Web development.

Frederic has eleven years experience working as a user interface designer and developer -- responsible for conceptual and practical design development for a wide range of business and industry needs.

You may send your suggestions and questions to fred.welterlin@avenuea-razorfish.com

“We should work toward a universal linked information system, in which generality and portability are more important than fancy graphics techniques and complex extra facilities.”

- Tim Berners-Lee, from original WWW proposal

References

- Page Zoom in IE7

(<http://blogs.msdn.com/ie/archive/2006/02/07/526805.aspx>)

- Browser Zoom Comparison

(<http://alastairc.ac/2006/11/browser-zoom-comparison/>)

- Web Accessibility Initiative

(<http://www.w3.org/WAI/intro/aria>)

- AJAX and Screenreaders: When Can it Work?

(<http://www.sitepoint.com/article/ajax-screenreaders-work>)

- Making Ajax Work with Screen Readers

(<http://juicystudio.com/article/making-ajax-work-with-screen-readers.php>)

- AJAX and Screen Readers – Content Access Issues

(<http://www.paciellogroup.com/blog/?p=15>)

About Avenue A | Razorfish

Avenue A | Razorfish is one of the largest interactive marketing and technology services agencies in the world. The company helps industry leaders such as Starwood Hotels, Kraft, Ford Motor Company and Carnival Cruise Lines use digital channels to acquire and service customers. Avenue A | Razorfish's full suite of digital offerings includes online advertising, Web site design and development, email and search engine marketing, emerging media strategies, and enterprise portal development. Its award-winning client teams have a great understanding of customer needs and provide solutions through distinct business disciplines, which include: analytics, strategy, technology, media, creative design and user experience. Avenue A | Razorfish has offices in markets across the United States, and global operations in Australia, China, France, Germany, Japan and the United Kingdom. Visit www.avenuea-razorfish.com for more information.

Avenue A | Razorfish
821 2nd Avenue, Suite 1800
Seattle, WA 98104
Phone: 206.816.8800 Fax: 206.816.8808
For more information please visit: www.avenuea-razorfish.com