

## Scripting for Accessibility

Within Applications Technology, many web pages are designed, not as static information sources, but as dynamic applications deployed through a web-browser. These web-based applications often utilize client-side scripting languages such as JavaScript to verify forms, provide interface enhancements such as roll-overs, and open new browser windows. Due to the necessity of these scripts in many applications, it may be frustrating to read through the W3C's Accessibility guidelines and still not have a clear idea of whether the application you've designed is accessible or not. Developing an accessible web-based application can also be frustrating when it's not particularly clear what the actual technical environment is – for example, what technical constraints can a developer expect a disabled end-user to meet? If an application requires JavaScript and a 4.x or newer browser, is the site still considered accessible? Below, we've outlined the W3C scripting guidelines with some additional comments (in bold) to clarify AT's perspective on scripting, as well as provide some helpful hints and strategies to meet the UW's accessibility standards.

To preface these comments, it should be known that AT has designated two primary environments that should at minimum be supported: Microsoft's HomePage Reader, and JAWS. These tools are slightly different in that, while HomePage Reader is its own browser, JAWS actually sits on top of an existing browser and interprets its activity to the end-user. In these two environments, most JavaScript behaviors will be supported.

The topics listed below are organized by subject. Relevant W3C checkpoints are listed in gray text boxes. Each includes the checkpoint number as well as the guideline itself.

### Graceful transformation of scripts

[6.5](#) Ensure that dynamic content is accessible or provide an alternative presentation or page. [Priority 2]

In our two primary environments, the use of script-generated text, and JavaScript encoded URLs (`<A HREF="JavaScript:"></A>`) does not appear to be a problem. When generating text geared toward a specific browser (i.e., text written after "Browser-sniffing"), it's preferable to compile the text using server-side scripting and browser detection. If a non-JavaScript page is detected, the client can then be sent to a page outlining the minimum requirements for your application. (See 12.4 for more details on automatically forwarding users to new web pages.)

### Scripts that cause flickering

[7.1](#) Until user agents allow users to control flickering, avoid causing the screen to flicker. [Priority 2]

This is just good common design sense. Unless your site is specifically meant to cause seizures you probably shouldn't do this.

### Scripts that cause movement and blinking

[7.2](#) Until user agents allow users to control blinking, avoid causing content to blink (i.e., change presentation at a regular rate, such as turning on and off). [Priority 2]

[7.3](#) Until user agents allow users to freeze moving content, avoid movement in pages. [Priority 1]

Same thing applies as 12.2. **Include comments about status bars**

### Directly accessible scripts

[6.4](#) For scripts and applets, ensure that event handlers are input device-independent. [Priority 2]

This is an important distinction. If your JavaScript relies on an “onMouseDown” event, and your end-user doesn’t use a mouse, how will their application function? In addition to the alternatives provided in the guidelines (specifying two handlers for the same element, or using device-independent attributes), it may also be useful to provide keystroke equivalents to navigate to form elements, or titles:

```
<FORM action="submit" method="post">
  <P>
    <LABEL for="user" accesskey="U">name</LABEL>
    <INPUT type="text" id="user">
  </P>
</FORM>
```

End example.

Please note however, there are few totally device-independent event scripts. Obviously a script that requires a key press, is dependent on a keyboard. But in most instances, assistive technology will emulate a keyboard at some level or another.

[6.3](#) Ensure that pages are usable when scripts, applets, or other programmatic objects are turned off or not supported. If this is not possible, provide equivalent information on an alternative accessible page. [Priority 1]

Again, we’re making the assumption that the end-user is operating in 1 of our 2 environments, in which case, JavaScript will be supported. Applets, however, are not supported by JAWS at this time and you will need to provide an alternative method to access the application’s data.

[8.1](#) Make programmatic elements such as scripts and applets directly accessible or compatible with assistive technologies [Priority 1 if functionality is important and not presented elsewhere, otherwise Priority 2.]

This is kind of a conglomeration of the two points we just addressed. Again, assistive technologies will generally have a keyboard equivalent; so as long as you’re creating “device-independent scripts,” as we defined them above, you should be covered by this. Applets are still not supported.

[9.3](#) For scripts, specify logical event handlers rather than device-dependent event handlers. [Priority 2]

The W3C certainly loves to repeat themselves, don’t they?

## Alternative presentation of scripts

[1.1](#) Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). *This includes:* images, graphical representations of text (including symbols), image map regions, animations (e.g., 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. [Priority 1]

[6.2](#) Ensure that equivalents for dynamic content are updated when the dynamic content changes. [Priority 1]

Once again, we're assuming the client's browser will support JavaScript, so this will not likely apply.

## Page updates and new windows

[7.4](#) Until user agents provide the ability to stop the refresh, do not create periodically auto-refreshing pages. [Priority 2]

This also applies to the METATAG reload attribute. Refreshing a page without warning can be very disorientating to some users.

[7.5](#) Until user agents provide the ability to stop auto-redirect, do not use markup to redirect pages automatically. Instead, configure the server to perform redirects. [Priority 2]

At one point or another, all of us have been stuck on some page that when we hit the back button, it continually throws us forward again. This occurs when a JavaScript or METATAG attribute automatically redirects the user to a new page. This is especially disorientating and confusing for some disabled users. By redirecting a user from the server-side, this issue is easily resolved. A quick ASP example for redirecting a user:

```
<%  
response.redirect "MyNewPage.html"  
%>
```

[10.1](#) Until user agents allow users to turn off spawned windows, do not cause pop-ups or other windows to appear and do not change the current window without informing the user. [Priority 2]

The real issue here is not so much never using pop-ups, but to never open a new window automatically. Many websites have ads that appear in this manner, where upon visiting their home page, you're immediately hit with a new window proclaiming the powers of soap. Many disabled users are unable to immediately see the window behind the pop-up window, and are left a bit disorientated. (Example: "Well, this ad is very interesting and all, but happened to my freakin' New York Times?") However, clicking a button which opens a new window is acceptable, as long as you're making it clear to the user that they're opening a new window. One problem with accessible browsers, however, is that a spawned window may not close via scripting. It's common practice to open up a detail window which can then be closed by hitting a "Close

## DoIT Applications Technology

Window” button, or by using the `self.close()` method. HomePage Reader unfortunately does not respond to this method. It may be necessary to alert the user that the window can be closed by hitting Ctrl-W on their keyboard.

Not changing the current window without informing the user is standard courtesy as well. It's kind of like going into a friend's house and rearranging their furniture without telling them. It's bad.