Techniques for User Agent Accessibility Guidelines 1.0

W3C Working Draft 27-August-1999

This version:
http://www.w3.org/WAI/UA/WAI-USERAGENT-TECHS-19990827
(plain text, postscript, pdf, gzip tar file of HTML, zip archive of HTML)
Latest version:
http://www.w3.org/WAI/UA/WAI-USERAGENT-TECHS
Previous version:
http://www.w3.org/WAI/UA/WAI-USERAGENT-TECHS-19990809
Latest "User Agent Accessibility Guidelines 1.0":
http://www.w3.org/WAI/UA/WAI-USERAGENT

Editors:
Jon Gunderson <jongund@uiuc.edu>
Ian Jacobs <ij@w3.org>

Abstract

This document provides techniques for implementing the checkpoints defined in "User Agent Accessibility Guidelines 1.0". These techniques address the accessibility of user interfaces, content rendering, program interfaces, and languages such as HTML, CSS and SMIL.

This document is part of a series of accessibility documents published by the Web Accessibility Initiative.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This is a W3C Working Draft for review by W3C Members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by, or the consensus of, either W3C or Members of the WAI User Agent (UA) Working Group.

While User Agent Accessibility Guidelines 1.0 strives to be a stable document (as a W3C Recommendation), the current document is expected to evolve as technologies change and content developers discover more effective techniques for designing accessible Web sites and pages.

This document has been produced as part of the Web Accessibility Initiative and intends to improve user agent accessibility for all users. The goals of the User Agent Guidelines Working Group are discussed in the Working Group charter. A list of the UA Working Group participants is available.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Please send comments about this document to the public mailing list: w3c-wai-ua@w3.org.

Table of Contents

1 Priorities

Each checkpoint in this document is assigned a priority that indicates its importance for users.

[Priority 1]
: This checkpoint **must** be implemented by user agents as a native feature or through compatibility with assistive technology, otherwise one or more groups of users with disabilities will find it impossible to access information. Satisfying this checkpoint is a basic requirement for some individuals to be able to use the Web.

[Priority 2]
: This checkpoint **should** be implemented by user agents as a native feature or through compatibility with assistive technology, otherwise one or more groups of users will find it difficult to access information. Satisfying this checkpoint will remove significant barriers to accessing Web documents.

[Priority 3]
: This checkpoint **may** be implemented by user agents as a native feature or through compatibility with assistive technology, to make it easier for one or more groups of users to access information. Satisfying this checkpoint will improve access to the Web for some individuals.

The checkpoints in this document are numbered to match their numbering in User Agent Accessibility Guidelines 1.0.

2 How the Techniques are Organized

This document is organized as follows:

User Agent Accessibility [p. 5]
: This section introduces some general techniques to promote accessibility in user agent functionality.

Interfaces and Conventions [p. 20]
: This section addresses user agent support for standard programming interfaces, operating system conventions, and W3C specifications.

HTML Techniques [p. 23]
: This section explains how to implement features of HTML (refer to [HTML40] [p. 48] , [HTML32] [p. 48]).

CSS Techniques [p. 29]
: This section explains how to implement features of CSS1 and CSS2 (refer to [CSS1] [p. 48] , [CSS2] [p. 48]).

SMIL Techniques [p. 29]
: This section explains how to implement features of SMIL (refer to [SMIL] [p. 48] .

A checkpoint map [p. 39] has been provided for navigation of the techniques. For each checkpoint, the map includes its definition (as it appears in the "User Agent Accessibility Guidelines 1.0") and links to applicable techniques for the checkpoint. In addition, the beginning of each section of this document lists the checkpoints that are addressed in that section.

2.1 Examples

This document contains a number of examples that illustrate accessible solutions in HTML, CSS, etc.

3 User agent accessibility

A list of assistive technologies and browsers designed for accessibility is available at the WAI Web site (refer to [USERAGENTS] [p. 49]).

3.1 Access to content

Checkpoints [p. 39] in this section: 3.1, 3.2.

It is not sufficient to convert a rendering intended for one medium into a rendering for a different medium (e.g., a graphical rendering to speech) since structural information is lost. Examples: table cells, nested lists (want to know where list item ends).

Also, serial access to content not always convenient, so we need additional mechanisms to select and receive content.

3.1.1 Where does content come from?

Some combination of document source, style sheets (which may hide content or generate content), and user agent additions (which may add contextual information or dynamic information such as whether a link has been visited).

In addition, the user agent may want to provide "intelligent" access to content to simplify the view or to convey models more familiar to users than what is conveyed by the DTD alone.

Also, make available information about abbreviation and acronym expansions. (@@How much rationale to include here?@@).

In Amaya ([AMAYA] [p. 49]) for access to attribute values: Place the cursor at the element in question, open/swap to the structure view. You are shown list of attributes and values. Another technique: select the element (press escape in Linux), then the attributes are all available from the attributes menu. For alt, I can also look at the alternate view, which renders alt text instead of images - a lynx-like view. All the views are synchronized for navigation (and for editing).

3.1.2 What does access mean?

Access means that the user agent renders selected content. Content includes alternative equivalents [p. 24] (e.g., attribute values).

In the simplest case, the user agent renders the document (e.g., two-dimensional graphical layout, audio stream, line-by-line braille stream) and the user has access to the entire rendering.

But this is not sufficient and so the user agent must provide navigation mechanisms that allow the user to set the selection/focus and then request the selected content (or information about the content - refer to another section...).

User interface issues

- How to indicate what piece of content one wants to access?
- How does the user agent present the information?
- How does the user configure the user agent to present associated contextual information?

Refer to the section on navigation [p. 10] for different navigation techniques (serial, direct, tree, etc.).

Structured v. Unstructured selection.

- Users may want to select content based on the rendering structure alone (i.e., that amounts to selecting across element borders).
- Users may want to select content based on structure (e.g., a table cell).

Contextual information

In addition to providing information about content, user agents should provide contextual information. Examples: table cell coordinates or header information. Or list item number within nested lists. Contextual information includes language of content as well.

3.1.3 Changes in content language

Checkpoints [p. 39] in this section: 3.3.

3.1.4 Changes over time

Checkpoints [p. 39] in this section: 3.4.

- Provide time-dependent information in a time-independent manner, such as a static list of links that are time-dependent and occupy the same screen real estate.
- Allow the user to control the timing of changes.
- Allow the user to navigate sequences of related links that vary over time.

3.2 Device independence

Checkpoints [p. 39] in this section: 1.2, 1.1, 1.3, 1.4, 1.5, 1.6

For non-text or replaced content:

- Look for the "alt" attribute (e.g., on IMG in HTML).
- Look for the "title" attribute
- If the replaced content is text (e.g., a "text" element in SMIL), use the text content.

For client-side image maps:

- Look for the "alt" attribute
- Look for the "title" attribute

3.3 User control of style

To ensure accessibility, users must have **final control** over certain renderings.

For text and color:

Checkpoints [p. 39] in this section: 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6.

- For changing text size, allow font size changes or provide a zoom mechanism.

[Ed. These may be rendered in a variety of ways. How do we specify rendering?]

Implement CSS ([CSS1] [p. 48] , [CSS2] [p. 48]) including the CSS2 cascade order and user style sheets. The CSS2 cascade order ensures that user style sheets with "!important" take precedence over author style sheets, giving users final control. Style sheets give authors design flexibility while offering users final control over presentation (refer also to [WAI-WEBCONTENT] [p. 49] , checkpoint 3.3). CSS should be implemented by user agents that implement CSS for text that it renders. CSS includes properties for audio, braille (fixed and refreshable), screen, and print rendering, and all relevant properties for supported output media should be implemented.

Note that in the CSS cascade order, markup is given less weight than style sheet rules. Thus, an author may use both presentation markup and style sheets, and user agents that support style sheets will prefer the latter.

A user style sheet can be implemented through a user interface, which means that the user may not have to understand how to write style sheets; they are generated or the user agent acts as though they were. For an example of this, refer to the style sheets implementation of Amaya ([AMAYA] [p. 49]), which provides a GUI-based interface to create and apply internal style sheets. The same technique could be used to control a user style sheet.

For images, applets, and animations:

Checkpoints [p. 39] in this section: 4.2 and 5.7.

Background images may be controlled by the use of local style sheets, and more effectively if these can be dynamically updated. Animation rate depends on the players used. User agents that provide native rendering of animation (for example a movie player, a driver for animated GIF images, or a java machine) should enable the control of animation rates, or at least allow the user to stop, and to play frame-by-frame, as well as straight rendering. A user agent could provide control of the general timing of a presentation, combined with the ability to select from available tracks manually. An issue to bear in mind is that when animation is synchronized with audio, a user may need the ability to play the animation separately from the associated audio.

For time-based presentations:

Implement user controls to start, atop, rewind and pause presentations, and where multiple tracks are supported, to choose which tracks should be rendered. SMIL ([SMIL] [p. 48]) provides for a number of these features. A SMIL implementation should provide for direct user control, as well as activation of the controls through a published API, for developers of assistive technologies.

For user agents rendering audio:

Checkpoints [p. 39] in this section: 5.11, 5.13, 5.9, 3.9, 3.8, and 5.12.

On selecting from among available description tracks. SMIL ([SMIL] [p. 48]) allows users to specify captions in different languages. By setting language preferences in the SMIL player, users may access captions (or audio) in different languages.

The G2 player from Real Networks currently allows users to specify which language they prefer, which can be evaluated in a SMIL ([SMIL] [p. 48]) document to choose from among text or audio tracks. Currently only one language can be indicated which does not permit choosing, for example, English spoken audio with Spanish captions.

The Quicktime player currently permits turning on and off any number of tracks individually, which can include audio, video, and text.

For user agents rendering video:

Checkpoints [p. 39] in this section: 5.8, 5.10, 5.9, and 3.8.

Implement the CSS positioning and/or SMIL layout languages. Allow the user to freeze a presentation, manually move and resize component video tracks (including captions, subtitles and signed translations) and to apply CSS stylesheets to text-based presentation and SVG.

For user agents rendering speech:

Checkpoints [p. 39] in this section: 5.15, 3.8, and 5.14.

CSS2 ([CSS2] [p. 48] properties for speech can allow users to control speech rate, volume, and pitch. These can be implemented by allowing the user to write and apply a local style sheet, or can be automatically generated by means of (accessible) user controls, which should also be controllable through an API.

[Editor: @@CMN- I vaguely recall seeing an example of this implemented in Java through the Accessibility API. Has anyone else seen this, or did I make it up?@@]

User interface:

Checkpoints [p. 39] in this section: 5.16.

- Allow the user to select large or small buttons and controls (and ensure that these values are applied consistently across the user interface) @@CMN: Opera does this.@@
- Allow the user to control features such as menu font sizes, or speech rates - this may be achieved through use of operating system standards.
- Allow the user to regroup buttons and controls, and reorder menus (@@CMN: MS Word does this. This is also related to reconfiguring commands, etc.@@)

3.3.1 Feature control

Checkpoints [p. 39] in this section: 4.1, 4.3, 4.4, 4.10, 4.9, and 4.5.

[Ed. Add note here that while useful to turn off support, say for all images, it's also useful to be able to view one particular image.]

User agents may:

- Allow users to turn of support for spawned viewports entirely
- Prompt them before spawning a viewport

For example, user agents may recognize the HTML construct `target="_blank"` and spawn the window according to the user's preference.

Checkpoints [p. 39] in this section: 4.14.

Period page refresh can be achieved with the following markup in HTML:

```
<META http-equiv="refresh" content="60">
```

The user agent should allow the user to disable this type of page refresh.

Checkpoints [p. 39] in this section: 4.13.

Although no HTML specification defines this behavior formally, some user agents support the use of the META element to refresh the current page after a specified number of seconds, with the option of replacing it by a different URI. Instead of this markup, authors should use server-side redirects (with HTTP).

User agents can provide a link to another page rather than changing the page automatically.

3.4 Viewports, Selection, and Focus

3.4.1 Highlighting the viewport, selection, and focus

Checkpoints [p. 39] in this section: 9.1.

- If colors are used to highlight the current viewport, selection, or focus, allow the user to set preferred colors and to ensure sufficient contrasts.
- If the current viewport is a window, allow the user to cause the window to pop to the foreground.
- If the current viewport is a frame or the user doesn't want windows to pop to the foreground, use colors, reverse videos, or other visual clues to indicate the current viewport. For speech or braille output, render the title or name of a frame or window and indicate changes in the current viewport.

3.4.2 Tracking selection and focus

Checkpoints [p. 39] in this section: 10.2 and 8.2.

3.5 Navigation and searching

Checkpoints [p. 39] in this section: 8.7

Configuration of navigation, navigating sets of items (but not always linear, e.g., tables).

3.5.1 Navigation of active elements

Checkpoints [p. 39] in this section: 8.4.

Sequential navigation includes all active elements. User agents might provide other navigation mechanisms limited to a particular type of element. For example "Find the next table" or "Find the previous form". The following checkpoints suggest some types of navigation.

- Serial navigation. It is important that application developers maintain a logical keyboard navigation order. The navigation order is defined as the order of navigation among components and component elements via the keyboard.

Generally users navigate by tabbing between components or groups and using the arrow keys within a component group or component's elements. The ability to tab between software components is a key feature in the implementation of keyboard accessibility. (Cross-reference to keyboard access.) Buttons of common functionality, such as a set of radio buttons used to set the location of a panel (top left, bottom left, and so on.), should be grouped together so the first element of the visible group can be tabbed to. Allow the user to use the arrow keys to navigate to each end of the group.
- How to indicate that something is in tabbing order in Java: A component is inclusive in the tabbing order when added to a panel and its isFocusTraversable() method returns true. A component can be removed from the tabbing order by simply extending the component, overloading this method, and returning false.
- For active elements, one way or two way.
- In a table, up/down and left/right.

Direct navigation:

Excessive use of serial navigation can reduce the usability of software for both disabled and non-disabled users. As a developer, you need to determine the point at which tabbing gets in the way and provide a keyboard alternative. This is done through the use of keyboard Mnemonics and Accelerators.

- Need for element identification.
- Access by position in document.
- Access by element content (e.g., first letter).
- In a table, access to cell based on coordinates.

3.5.2 Navigation of document structure

Checkpoints [p. 39] in this section: 8.6

- DOM is minimal (tree nav)
- Best navigation will involve a mix of source tree information and rendered information.
- May use commonly understood document models rather than strict DTD navigation. E.g., nesting headers in HTML.
- Goal of simplifying the structure view as much as possible.
- Allow the user to control level of detail/ view of structure.
- Depth first as well as breadth first possible. Up/down nav as well.
- Navigation of synchronized multimedia: allow users to stop, pause, fast forward, advance to the next clip, etc.

3.5.3 Table navigation

Checkpoints [p. 39] in this section: 8.3

Users of non-graphical rendering technologies and users with learning disabilities, when browsing a page, should be able to quickly determine the nature of a table located on a page. The able-bodied user is able to visually examine the table and extract a sense of the table contents with a quick scan of the cells. Users with blindness or low vision, or users who have difficulty translating printed material may not able to do this. Providing table summary information, when first moving the point-of-regard to a table allows the nature of a table to be easily determined.

An auditory rendering agent, when the point-of-regard moves to a table, might say, "Table: Tax tables for 1998," thus identifying the nature of the table. The user could then use keyboard commands to move the point of regard to the next logical block of information, or use a different command to "burrow" into the table.

The "burrow" command should have an opposite "pop" command, which would move the point of regard from an individual cell to the table as a whole, so that the user can leave a table from any cell within it, rather than navigating to the end.

If the user "pops" up to look over the summary information, it should be possible to "burrow" back to the same cell.

When navigating a table that contains another table, this strategy can avoid confusion. For example, if each row of a table contained five cells, but the second row contained a 4x4 table in the third cell, a user could be disoriented when the row did not end as expected. However, when the point of regard moved to the third cell of the table, a compliant browser would report that this was a table, and describe its contents. The user would have the option of navigating to the forth cell of the parent table, or burrowing into the table within this cell.

When rendering tabular information, the fact that it is tabular information should be apparent. For a graphical user agent, such information is commonly made obvious by the border attribute. However, for a non-graphical agent, such information must also be made appreciable.

As the user agent shifts the point of regard to a table, it should first provide information about the entire table. This information might be the Caption, Title, or Summary information of the table. Access to this information would allow the user to determine whether or not to examine the contents of the table, or to move the point of regard to the next block of content.

In many data tables, the meaning of the contents of a cell are related to the contents of adjacent cells. For example, in a table of sales figures, the sales for the current quarter might be best understood in relation to the sales for the previous quarter, located in the adjacent cell.

In order to provide access to contextual information for individuals using non-graphical browsers, or for individuals with certain types of learning disabilities, it is necessary for the user agent to allow the point of regard to be moved from cell to cell, both right/left and up/down via keyboard commands.

The most direct method of performing such navigation would be via the cursor keys, though other navigation strategies might be used.

Users of graphical browsers can easily locate cells within a table that are at the intersection of a row and column of interest. To provide equivalent access to users of non-graphical browsers, equivalent means of navigation should be provided. The search function of a browser will allow the user to locate key terms within a table, but will not allow the user to find cells that are at the intersection of rows and columns of interest.

Techniques:

- An advanced search mode might provide entries for header information, allowing the user to find information at the intersection of columns and rows using the key terms.
- A search mode might allow the user to search for key terms that are related to key header terms, allowing searches to be restricted to specific rows or headers within a table.

The header information visible in a TH cell may be abbreviated, in which case it should be user preference to see the "abbr" value if any or the full contents.

Axis information may also help the user search into confined portions of the table.

Column groups and row groups are other confining partitions of a table in which a search may be limited.

Software:

- Table navigation script from the Trace Center
-

3.5.4 Searching

Checkpoints [p. 39] in this section: 8.5.

- Element content
- Attribute content
- Patterns
- Search the source view.
- For forms, allow users to find required controls. Allow users to search on labels as well as content of some controls.
- For multimedia presentations:
  - Allow users to search and examine time-dependent media elements and links in a time-independent manner. For example, present a static list of time-dependent links.
  - Allow users to search closest timestamp from a text stream or a media elements or links and find other media elements active at the same time.
  - Allow users to view a list of all media elements or links of the presentations sorted by start or end time or alphabetically.

3.5.5 View navigation

Checkpoints [p. 39] in this section: 8.1, 9.3.

[Ed. @@CMN: For example, Opera does allow the user to navigate between views - in it's case various windows, as does emacs-w3. Other systems rely on the Operating System to do it - in MSWindows and the X window system there are keyboard and mouse driven methods for moving among open windows.]

3.6 Context and orientation

3.6.1 Status information

Checkpoints [p. 39] in this section: 10.4 and [##Tnotify-resource-nav].

3.6.2 Element context

Checkpoints [p. 39] in this section: 9.4.

3.6.3 Links

- Address broken link handling so that it doesn't disorient users. For example, leave viewport as is and notify user.

Checkpoints [p. 39] in this section: 9.5 and 9.6.

3.6.4 Tables

Checkpoints [p. 39] in this section: 9.9 and 9.8.

3.6.5 Form controls

Checkpoints [p. 39] in this section: 9.10, 10.6, 9.7.

- For labels explicitly associated with form controls (e.g., "for" attribute on LABEL in HTML), make available label information when the user is navigating among the form controls.

Statement of form submission problems from Gregory Rosmaita:

Point A: As a user, I do *not* want to be prompted time I submit a form, provided that I submitted the form by activating its submit button. If, however, I simply hit the ENTER or the RETURN key from within a FORM control (i.e., rather than explicitly activating the SUBMIT mechanism), I would like the UA to request confirmation before submitting the form content.

Point B: As a user, I do NOT want the form content automatically submitted if I inadvertently press the ENTER or RETURN key.

PROBLEM STATEMENT FOR POINT B:

Inadvertently pressing the RETURN or ENTER key is quite a prevalent phenomenon amongst users of every level of expertise - especially those who often find it necessary to switch between user agents. Lynx, for example, uses the ENTER key within FORMs as a means of exposing drop-down (or pop-up, depending upon your point of view) SELECT menus. Thus, when one encounters a SELECT menu using Lynx, one: exposes the content of the menu by pressing the ENTER key, and then is able to navigate between OPTIONs using the up and down arrows or via Lynx's text-search feature. When one finds the appropriate OPTION, it is selected by pressing ENTER, which causes the selected item to be displayed in the SELECT menu listbox.

The problems posed by the default "submit on enter" feature of most GUI browsers, is not limited to the SELECT menu problem outlined above. Lynx (as well as several other text-based browsers) uses the ENTER/RETURN key as a means of toggling several FORM controls, such as the selection of checkboxes and radio buttons.

Moreover, I would like to stress that the "Auto-Submit-On- Enter" feature is not only quite problematic for one operating in an eyes-free environment, but for those unaccustomed to using online forms, and for those unfamiliar with a particular user agent's default key- bindings for forms, as well as those (like myself and countless others) who surf the web using a variety of browsers, often switching from browser to browser -- ALT- TAB-ing from Lynx32 to MSIE to Opera, for example -- in order to better comprehend the contents of a page or while attempting to navigate an poorly structured site or a poorly marked-up form

Point C: As a speech user, I am constantly frustrated and misdirected by the use of javascript and event handler controlled pseudo-forms, wherein the user is presented with a menu (in the form of a listbox in GUI browsers), and is redirected to a different viewport upon selection of an OPTION.

PROBLEM STATEMENT FOR POINT C:

The markup behind such pseudo-forms is a mix of javascript (in particular the "function switchpage(select)" command) and HTML FORM controls, which utilize HTML4's event handler script attributes (in particular the "onchange" event handler attribute has been defined. An example (gleaned from the document source for one website follows:

```
<SELECT NAME="condition" onchange="switchpage(this)">
```

When such a menu is encountered by a websurfer who is using speech synthesis in conjunction with a javascript enabled user agent, his or her instinctual reaction will be to use the UA's navigation mechanism (usually the up and down arrows) to review the available OPTIONs. However, each time a new OPTION is displayed, the user is abruptly taken to a new viewport. Conversely, if one is using a user agent that does not support javascript (or has javascript support disabled), then the menu is displayed, but since there is no SUBMIT mechanism associated with it, there is no mechanism by which one can use the menu to quickly switch viewports - the apparent purpose of this type of pseudo-form. And while one can avoid having the viewport abruptly changed when encountering the menu (at least in the Windows environment) by using the ALT-LEFT-ARROW keystroke to display the menu in a drop-down list, (a) very few users know this keystroke, and (b) when one encounters a listbox on a page in an aural environment, one usually assumes that he or she is navigating a valid FORM, in which there are no unexpected side effects to perusing the contents of a SELECT menu using the arrow keys

Note. I have chosen to address the issue of pseudo-forms in this context, for, although they straddle the boundary between "Form Controls" and Checkpoint 5.8, pseudo-forms rely on FORM elements for activation This issue has been raised in the past, particularly by Chris Kreussling.

Techniques:

1. Allow the user to configure the user agent. Choices should include:
   - "Never Allow Automatic Submission of Form Content " or "Never Submit {Do Not Prompt}"
   - "Always Allow Automatic Submission of Form Content" or "Always Submit Without Prompting"
   - "Prompt Before Submitting Form Content"

   The default setting should be: "Prompt before submitting form content", so as to allow the user to decide whether or not HTML4 event handling will occur automatically.
2. Configuration can be determined by prompting the user the first time an event handled or script-driven FORM is encountered. Choices should include:
   - "Submit" {optional verbiage: "This Time Only"}
   - "Do Not Submit" {optional verbiage: "This Time Only"}
   - "Always Allow Automatic Submission of Form Content" or "Always Submit Without Prompting"
   - "Never Allow Automatic Submission of Form Content " or "Never Submit {Do Not Prompt}"
   - "Always Prompt Before Submitting Form Content"

   If the user chooses "Prompt Before Submitting Form Content", this prompt could be recycled in an abbreviated fashion. The prompt should include:
   - "Submit This Time Only"
   - "Do Not Submit"
   - "Always Submit and Do Not Ask/Warn Me Again"
   - "Never Submit and Do Not Ask/Warn Me Again"

Refer also to [WAI-WEBCONTENT] [p. 49] , checkpoint 6.3: Content developers must ensure that pages are accessible with scripts turned off or in browsers that don’t support scripts.

3.6.6 Frames

Checkpoints [p. 39] in this section: None?

3.6.7 Scripts

Checkpoints [p. 39] in this section: 10.1, 10.3.

3.7 Keyboard access

Checkpoints [p. 39] in this section: 2.1.

3.7.1 Tabbing order

- How to specify in HTML

3.7.2 Keyboard shortcuts

Checkpoints [p. 39] in this section: 2.2, 2.3, 2.6.

- How to specify in HTML
- Visibility of.
- Documentation of. At a minimum: list in README file that comes with software.

Some suggestions:

1. Allow the user to use the find command to jump to a link instead of tabbing there. It would save a lot of keystrokes, especially if one programs the keystrokes as macros. But this requires that focus moves to the location that find highlights.
2. Allow the user to use find command to jump to text in buttons.
3. Allow the user to use find command to jump to image by searching on its alternative content (e.g., "alt" attribute).

Some reserved keyboard shortcuts are listed in the appendix on accessibility features of some operating systems.

3.7.3 Software consistency

Checkpoints [p. 39] in this section: 9.11

3.8 Configuration

- Profiles
- Default values
- Device-independent configuration

3.8.1 User profiles

Checkpoints [p. 39] in this section: 11.1

A configuration profile allows users to save their user agent settings and re-apply them easily, which is particularly valuable in an environment where several people may use the same machine.

The user should be able to easily transfer their profiles between installations of the same user agent. One way to facilitate this is to follow operating system conventions for profiles where applicable.

Users should be able to switch rapidly between profiles (or the default settings). This is helpful when:

- Several people use the same machine.
- A user with a disability is being helped by able-bodied user who may not recognize the information being displayed using the user's profile.

User agents may apply a profile when the user logs in. They may also allow users to apply settings interactively, for example by allowing them to choose from a list of named profiles in a menu.

Sample profiles (based on common usage scenarios) can assist users in the initial set up of the user agent. These profiles can serve as models and may be copied and fine-tuned to meet an individual's particular needs.

3.8.2 Keyboard configuration

Checkpoints [p. 39] in this section: 2.4, 2.8, 2.7, and 2.5

[Ed. Discuss keyboard access here. How access is specified in HTML: "accesskey".]

[Ed. New section. Users must be allowed to control keyboard configuration based on specific needs. For poor motor control, keys far apart. For poor mobility, keys close together. General principle: fewest keystrokes, short distance to move.]

3.8.3 User interface

Checkpoints [p. 39] in this section: 11.2.

- Allow multiple icon sizes.

3.9 Documentation

Checkpoints [p. 39] in this section: 12.2.

Universal design means that access to features that help accessibility should be integrated into normal menus. User agents should avoid regrouping access to accessibility features into specialized menus. Documentation includes anything that explains how to install, get help for, use, or configure the product. Users must have access to installation information, either in electronic form (diskette, over the Web), by fax, or by telephone.

3.9.1 Where to document accessibility

Checkpoints [p. 39] in this section: 12.3.

Include references to accessibility features in these parts of the documentation:

1. Indexes. Include terms related to product accessibility in the documentation index (e.g., "accessibility", "disability" or "disabilities").
2. Tables of Contents Include terms related to product accessibility in the documentation table of contents (e.g., features that promote accessibility)
3. Include instructions on how to modify all user configurable defaults and preferences (e.g, images, video, style sheets, and scripts) as specified by the documentation.
4. Include a list of all keyboard shortcuts in the accessibility section of the documentation.

3.9.2 Accessible documentation

Checkpoints [p. 39] in this section: 12.1.

Documentation created in HTML should follow the Web Content Accessibility Guidelines [p. 49] .

Electronic documentation created in open standard formats such as HTML and ASCII can often be accessed in the user's choice of application such as a word processor or browser. Accessing documentation in familiar applications is particularly important to users with disabilities who must learn the functionalities of their tools and be able to configure them for their needs. Commonly used applications are also more likely to be compatible with assistive technology. Electronic documentation should not be provided in proprietary formats.

Users with print impairments may need or desire documentation in alternative formats such as Braille, large print, or audio tape. User agent manufacturers may provide user manuals in alternative formats. Documents in alternative formats can be created by agencies such as Recording for the Blind and Dyslexic and the National Braille Press.

User instructions should be expressed in an input device-independent manner. Provide instructions for using or configuring the user agent in a manner that can be understood by a user of any input device including a mouse or keyboard. For example, "Click on the Home button on the toolbar or select "Home" from the Go menu to return to the Home page. "

4 Interfaces and conventions

4.1 System conventions

Checkpoints [p. 39] in this section: 6.6

Develop the UA User Interface (UI) with standard interface components per the target platform(s). Most major operating system platforms provide a series of design and usability guidelines, these should be followed when possible (see platforms below).

These checklists, style guides, and human interface guidelines provide very valuable information for developing applications (e.g., UAs) for any platform/operating system/GUI. If your custom interface cannot provide information or operation as defined above, then you may need to design your UA using any additional options provided by that platform.

For instance, software should use the standard interface for keyboard events rather than working around it.

Evaluate your standard interface components on the target platform against any built in operating system accessibility functions (see Appendix 8) and be sure your UA operates properly with all these functions.

For example, take caution with the following:

- Microsoft Windows supports an accessibility function called "High Contrast". Standard window classes and controls automatically support this setting. However, applications created with custom classes or controls must understand how to work with the "GetSysColor" API to ensure compatibility with High Contrast.
- Apple Macintosh supports an accessibility function called "Sticky Keys". Sticky Keys operates with keys the operating system understands to be defined as modifier keys, and therefore a custom UA control should not attempt to define a new modifier key.

4.1.1 Some guidelines for specific platforms

- "Macintosh Human Interface Guidelines" [APPLE-HI] [p. 49] Apple Computer Inc.
- "IBM Guidelines for Writing Accessible Applications Using 100% Pure Java" [JAVA-ACCESS] [p. 49] .
- "An ICE Rendezvous Mechanism for X Window System Clients" [ICE-RAP] [p. 49] .
- "Information for Developers About Microsoft Active Accessibility" [MSAA] [p. 50] .
- "The Inter-Client communication conventions manual" [ICCCM] [p. 49] .
- "Lotus Notes accessibility guidelines" [NOTES-ACCESS] [p. 50]
- "Java accessibility guidelines and checklist" [JAVA-CHECKLIST] [p. 49]

- "The Java Tutorial. Trail: Creating a GUI with JFC/Swing" [JAVA-TUT]. [p. 49]
- "The Microsoft Windows Guidelines for Accessible Software Design" [MS-SOFTWARE] [p. 50] .

4.1.2 General guidelines for producing accessible software

- "Accessibility for applications designers" [MS-ENABLE] [p. 50] .
- "Application Software Design Guidelines" [TRACE-REF] [p. 50] .
- "Designing for Accessibility" [SUN-DESIGN] [p. 50]
- "EITAAC Desktop Software standards" [EITAAC] [p. 49] .
- "Requirements for Accessible Software Design" [ED-DEPT] [p. 49]
- "Software Accessibility" [IBM-ACCESS] [p. 49] .
- "Towards Accessible Human-Computer Interaction" [SUN-HCI] [p. 50] .
- "What is Accessible Software" [WHAT-IS] [p. 50] .
- Accessibility guidelines for Unix and X Window applications [XGuidelines] [p. 50]

4.2 Testing UA operation with platform standards

Ensure your UA can be operated using the standard interfaces on the target platform(s). Some example tests include:

- All functional UI components must be keyboard accessible and therefore, must be oper-able by software or devices which emulate a keyboard (Use SerialKeys [see Appendix 8] and/or voice recognition software to test keyboard event emulation.). Individuals with varying physical abilities should be able to access your UA using a SerialKeys device or using voice recognition, provided it is keyboard accessible.
- All functional UI components must track selection and focus. Individuals who have low vision and use screen magnification software should be able to follow highlighted item(s) (e.g., selection), text input location (e.g., sometimes referred to as the "caret"), and any control or component with focus, if your UA exposes these properties correctly.
- All functional UI components must provide readable "text" names or labels, even when not visible. Providing this type of information in your UA along with the prior two examples, means that individuals who are blind and accessing your UA using screen reading software and/or a Braille output device should be able to operate and navigate within it.
- All functional UI components which convey important information using sound, also need to provide alternate, parallel visual representation of the information for individuals who are deaf, hard of hearing, or operating your UA in an environment where the use of sound isn’t practical.

4.3 Accessibility interfaces

Checkpoints [p. 39] in this section: 6.1, 6.2, 6.3, 6.5.

The operating system application programming interfaces (APIs) that support accessibility are designed to provide a bridge between the standard user interface supported by the operating system and alternative user interfaces developed by third-party assistive technology vendors to provide access to persons with disabilities. Applications supporting these APIs are therefore generally more compatible with third-party assistive technology.

The User Agent Accessibility Guidelines Working Group strongly recommends using and supporting APIs that improve accessibility and compatibility with third-party assistive technology. Third-party assistive technology can use the accessibility information provided by the APIs to provide an alternative user interface for various disabilities.

The following is an informative list of currently public APIs that support accessibility:

- Microsoft Active Accessibility ([MSAA] [p. 50]) in Windows 95/NT versions.
- Sun Microsystems Java Accessibility API ([JAVAAPI] [p. 49]) in Java Code.

Many operating systems have built-in accessibility features for improving the usability of the standard operating system by persons with disabilities. When designing software that runs above an underlying operating system, developers should ensure that the application:

1. Makes use of operating system level features. See the appendix of accessibility features [p. 17] for some common operating systems.
2. Inherits operating system settings related to accessibility. Pertinent settings include font and color information as well as other pieces of information discussed in this document.

Write output to and take input from standard system APIs rather than direct from hardware controls where possible. This will enable the I/O to be redirected from or to assistive technology devices - for example, screen readers and braille devices often redirect output (or copy it) to a serial port, while many devices provide character input, or mimic mouse functionality. The use of generic APIs makes this feasible in a way which allows for interoperability of the assistive technology with a range of applications.

4.4 The document object model

Checkpoints [p. 39] in this section: 6.4.

[Ed. Need to emphasize more of what DOM can do.]

Use DOM Level 1 [DOM1] [p. 48] for access to HTML and XML document information. However, as the DOM specification indicates:

> The DOM Level 1 does not include mechanisms to access and modify style specified through CSS 1. Furthermore, it does not define an event model for HTML documents. This functionality is planned to be specified in a future Level of this specification.

It is important for user agents to provide access to style and scripting information in a document, so the following techniques should be used to achieve this.

Use standard rather than custom controls when designing user agents. Third-party assistive technology developers are more likely able to access standard controls than custom controls. If you must use custom controls, review them for accessibility and compatibility with third-party assistive technology.

4.5 Information for assistive technologies

[Ed. To be completed.]

5 Support for HTML accessibility

[Ed. Link to W3C Note on HTML Accessibility Features instead of this section]

Checkpoints [p. 39] in this section: 7.2, 7.1

Following is a list of accessibility features in HTML:

- Support the "longdesc" attribute defined for IMG elements ([HTML 4.0] [p. 48] , section 13.2). This attribute may be used to attach additional descriptive information to images. [Priority 1]
- Support the CAPTION element ([HTML40] [p. 48] , section 11.2.2) for rich table captions. [Priority 2]
- Support the ACRONYM and ABBR elements ([HTML40] [p. 48] , section 9.2.1) for acronyms and abbreviations. [Priority 2]
- Support the "summary" attribute for TABLE ([HTML40] [p. 48] , section 11.2.1) for table summary information. [Priority 2]
- Support the NOSCRIPT element ([HTML40] [p. 48] , sections 18.3.1 and 16.4.1) for accessible alternatives to scripts. [Priority 2]
- Support the NOFRAMES element ([HTML40] [p. 48] , sections 18.3.1 and 16.4.1) for accessible alternatives to frames. [Priority 2]
- Support the "lang" attribute ([HTML40] [p. 48] , section 8.1). [Priority 2]
- Support the "tabindex" attribute ([HTML40] [p. 48] , section 17.11.1) for assigning the order of keyboard navigation within a document. [Priority 3]
- Support the "accesskey" attribute ([HTML40] [p. 48] , section 17.11.2) for assigning keyboard commands to active components such as links, and form controls. [Priority 3]

5.1 Equivalent information

Checkpoints [p. 39] in this section: 3.1.

User agents must be able to recognize sources of alternative representations of content.

- For the IMG element: The "alt", "title", and "longdesc" attributes
- For the OBJECT element: The content of the element and the "title" attribute.
- For the APPLET element: The "alt" attribute and the content of the element.
- For the AREA element: The "alt" attribute.
- For the INPUT element: The "alt" attribute.
- For the ACRONYM and ABBR elements: The "title" attribute may be used for the acronym or abbreviation expansion.
- For the TABLE element, the "summary" attribute
- For frames, the NOFRAMES element and the "longdesc" attribute on FRAME and IFRAME.
- For scripts, the NOSCRIPT element.

5.1.1 Sources of blinking text and animation

Checkpoints [p. 39] in this section: 4.6 and 4.7.

User agents that recognize the following sources of blinking text and animations must enable users to freeze that text.

In HTML
- The BLINK element. **Note.** The BLINK element is not defined by a W3C specification.
- The MARQUEE element. **Note.** The MARQUEE element is not defined by a W3C specification.

In CSS
- The ’blink’ value of the ’text-decoration’ property.

In GIF animated images.
To be completed.

5.1.2 Alternative representations of information for images, video, applets

Checkpoints [p. 39] in this section: 3.5, 3.6.

[Ed. How to handle null/absent alt.]

5.1.3 Textual equivalents for audio and video

Checkpoints [p. 39] in this section: 3.7 and 5.9.

Scenario-video showing professor writing complex equations and graphs on the overhead and discussing them but not describing what he/she actually wrote on the overhead. Without description this would be inaccessible to people with visual impairments. This could be generalize to any video presentation of visually rich or complex information where the visually presented information is critical to the understanding of the presentation.

5.1.4 Auditory equivalents for video

Checkpoints [p. 39] in this section: 3.7.

5.1.5 Frame equivalents

Checkpoints [p. 39] in this section: 4.12 and 4.11.

[Ed. See Scott's suggestions]

1. In HTML, content of NOFRAMES.
2. Otherwise, list of frames. Use "title" as name of frame, otherwise "name".

5.1.6 Alternative representations for tables

In HTML, "summary" attribute. Also, the "abbr" attribute for headers (see the section on table cells and headers).

5.2 Links

Checkpoints [p. 39] in this section: 9.7.

[Ed. Talk about CSS pseudo-classes for :hover]

[Ed. Talk about using :before to clearly indicate that something is a link (e.g., 'A:before { content : "LINK:" }')]

5.3 Tables

Tables were designed to structure relationships among data. In graphical media, tables are often rendered on a two-dimensional grid, but this is just one possible interpretation of the data. On the Web, the HTML TABLE element has been used more often than not to achieve a formatting effect ("layout tables") rather than as a way to structure true tabular data ("data tables")

Layout tables cause problems for some screen readers and when rendered, confuse users. Even data tables can be difficult to understand for users that browse in essentially one dimension, i.e. for whom tables are rendered serially. Large tables pose particular problems since remembering cell position and header information becomes more difficult as the table grows.

User agents facilitate browsing by providing access to specific table cells and their associated header information. How headers are associated with table cells is markup language-dependent.

Tabular navigation is required by people with visual impairments and some types of learning disabilities to determine the content of a particular cell and spatial relationships between cells (which may convey information). If table navigation is not available users with some types of visual impairments and learning disabilities may not be able to understand the purpose of a table or table cell.

5.3.1 Table rendering

Properly constructed data tables generally have distinct TH head cells and TD data cells. The TD cell content gains implicit identification from TH cells in the same column and/or row.

For layout tables, a user agent can assist the reader by indicating that no relationships among cells should be expected. Authors should not use TH cells just for their formatting purpose in layout tables, as those TH cells imply that some TD cells should gain meaning from the TH cell content.

When a table is "read" from the screen, the contents of multiline cells may become intermingled. For example, consider the following table:

```
This is the top left cell    This is the top right cell
of the table.                of the table.

This is the bottom left      This is the bottom right
cell of the table.           cell of the table.
```

If read directly from the screen, this table might be rendered as "This is the top left cell This is the top right cell", which would be confusing to the user.

A user agent should provide a means of determining the contents of cells as discrete from neighboring cells, regardless of the size and formatting of the cells. This information is made available through the DOM ([DOM1 [p. 48]]).

The contents of a cell in a data table are generally only comprehensible in context (i.e., with associated header information, row/column position, neighboring cell information etc.). User agents should provide users with header information and other contextual agent. Techniques include:

- Provide this information through an API.
- Ignore table markup entirely. This may assist some screen readers.
- Render cells as blocks. This may assist some screen readers. Using this strategy, the user agent might render individual cells with the relevant top and side headers attached.
- Allow navigation and querying of cell/header information. When the point of regard is on an individual cell, the user would be able to use a keyboard command to receive the top and left header information for that cell. The user agent should appropriately account for headers that span multiple cells.
- Allow users to read one table column or row at a time, which may help them identify headers.

[Ed. Discuss repair strategies for finding header information?]

Since not all tables are designed with the header information, a conforming user agent should provide, as an option, a "best guess" of the header information for a cell. Possible strategies include:

- Consider the top and left-most cells of a column or row to be header information.
- Consider upper and left-most cells which have formatting markup to be header information.

The user may choose the form and amount of this information, possibly announcing the row heads only once and then the column head or its abbreviation ("abbr") to announce the cell content.

[Ed. Define algorithm for finding "header information" here. Does it come from THEAD, TH, attributes, HTML 4.0 algorithm for finding header information, etc. Allow the user to choose either "abbr" or what is calculated by the header algorithm.]

5.3.2 Cell rendering

Non-graphical rendering of information by a browser or an assistive technology working through a browser will generally not render more than a single cell, or a few adjacent cells at a time. Because of this, the location of a cell of interest within a large table may be difficult to determine for the users of non-graphical rendering.

In order to provide equivalent access to these users, compliant browsers should provide a means of determining the row and column coordinates of the cell having the point of regard via keyboard commands. Additionally, to allow the user of a non-graphical rendering technology to return to a cell, the browser should allow a means of moving the point of regard to a cell based on its row and column coordinates.

At the time the user enters a table, or while the point of regard is located within a table, the user agent should allow an assistive technology to provide information to the user regarding the dimensions (in rows and columns) of the table. This information, in combination with the summary, title, and caption, can allow the user with a disability to quickly decide whether to explore the table of skip over it.

Dimensions is an appropriate term, though dimensions needn't be constants. For example a table description could read: "4 columns for 4 rows with 2 header rows. In those 2 header rows the first two columns have "colspan=2". The last two columns have a common header and two subheads. The first column, after the first two rows, contains the row headers.

Some parts of a table may have 2 dimensions, others three, others four, etc. Dimensionality higher than 2 are projected onto 2 in a table presentation.

5.3.3 Cell header algorithm

User agents should use the algorithm to calculate header information provided in the HTML 4.0 specification ([HTML40] [p. 48] , section 11.4.3).

[Ed. The following issues were raised by Harvey Bingham.]

1. TH cells on both the left and right of the table need to be considered.
2. For TH cells with "rowspan" set: the content of those TH cells must be considered for each of the N-1 rows below the one containing that TH content.
3. An internal TH in a row surrounded on either side by TDs has no means to specify to which (row or column) that TH overrides what existed to its left or above it.
4. Finding column header cells assumes they are all above the TD cell to which they apply.
5. A TH with "colspan" set needs to be included in the list of TH for the M-1 columns to the right of the column in which the TH is found.

5.4 Frames

Checkpoints [p. 39] in this section: 5.16 and 9.2

5.4.1 Frame formatting

Checkpoints [p. 39] in this section: 4.12.

Possible solutions:

- Provide info through an API.
- Frames as blocks.
- Frameset as list of links to individual frames (based on frame name).
- NOFRAMES content (when?)

5.5 Scripts

Checkpoints [p. 39] in this section: 4.8.

Certain elements of the document language may have associated event handlers that are triggered when certain events occur. User agents must be able to identify those elements with event handlers statically associated (i.e., associated in the document source, not in a script).

In HTML
: All of the attributes beginning with the prefix "on": "onblur", "onchange", "onclick", "ondblclick", "onkeydown", "onkeypress", "onkeyup", "onload", "onmousedown", "onmousemove", "onmouseout", "onmouseover", "onmouseup", "onreset", "onselect", "onsubmit", and "onunload".

[Ed. Other sources?]
[Ed. To be completed.]

5.6 Multimedia

[Ed. Talk about EMBED? Charles recommends that it be put inside an OBJECT if used.]

6 Support for CSS accessibility

Checkpoints [p. 39] in this section: 7.2, 7.1

The accessibility features of Cascading Style Sheets (refer to CSS, level 1 [p. 48] and CSS, level 2 [p. 48]) are described in [CSS-ACCESS] [p. 48] .

Cascading Style Sheets may be part of a source document or linked externally. Stand-alone style sheets are useful for implementing *user profiles* in public access computer environments where several people use the same computer. User profiles allow for convenient customization and may be shared by a group.

7 Support for SMIL accessibility

[Ed. Link to W3C Note on SMIL Accessibility Features instead of this section]

Checkpoints [p. 39] in this section: 7.2, 7.1

[Ed. Ensure that in SMIL, users can stop temporal links. See proposal from Marja.]

Accessible presentation means here that the information can be easily accessed by using different media and that the user can easily control whether to show the information or not.

- Support accessible presentation of "title", "abstract" and "author" attributes defined for SMIL synchronization elements.
- Support accessible presentation of "alt", "londesc", "title", "abstract" and "author" attributes defined for SMIL media object elements.
- Support an interface for turning captions on and off. Captions are defined either by using "system-overdub-or-caption" attribute with value "caption" or "system-captions" attribute with value "on".
- Support an interface for turning overdubs on and off. Overdubs are defined by using "system-overdub-or-caption" attribute with value "overdub".
- When captions are shown support a caption region. According to user preferences this could be embedded in the SMIL layout or shown in a separate window.
- When user prefers to, support a stable, time independent view to the links so that time dependent links can be easily searched.
- Offer a way to change system values affecting to "system-bitrate", "system-screen-size" and "system-depth" attributes since these can have an affect to the user interface. In addition, give feedback of the values in a form that

list only those selections that can have an effect to the user interface. For instance, if the SMIL author has defined three different user interfaces for bitrates below 1200, between 1200 and 2400, and over 24000 the user should be given a selection of these three choices not just any bitrate number.

8 Appendix: Accessibility features of some operating systems

Several of the more popular mainstream operating systems now include a common suite of built-in accessibility features that are designed to assist individuals with varying abilities. Despite operating systems differences, the built-in accessibility features use a similar naming convention and offer similar functionalities, within the limits imposed by each operating system (or particular hardware platform).

The following is a list of built-in accessibility features from several platforms:

StickyKeys
: These allow users to perform a multiple simultaneous key sequence by pressing and releasing each key in sequential order. StickyKeys is designed to work with only those keys defined as modifier keys. Modifier keys are pressed in combination with other keys, to change the outcome of the second (or more) pressed keys. For example, the SHIFT key is defined as a modifier key, since it is commonly used to create upper case characters. Each operating system or hardware platform typically defines a set of keys which can act as modifier keys. The most common modifier keys include SHIFT, CONTROL, and ALTERNATE.

MouseKeys
: These allow users to move the mouse cursor and activate the mouse button(s) from the keyboard.

RepeatKeys
: These allow users to set how fast a key repeats (e.g., sometimes referred to as typematic rate) when the key is held pressed (e.g., Repeat Rate), and also allows control over how quickly the key starts to repeat after the key has been pressed (e.g., delay Until Repeat). Key repeating may also be eliminated.

SlowKeys
: These instruct the computer not to accept a key as pressed until it has been pressed and held down for a specific user adjustable length of time.

BounceKeys
: These prevent extra characters from being typed if the user bounces (e.g., tremor) on the same key when pressing or releasing it.

ToggleKeys
: These provide an audible indication for the status of keys that have a toggled state (e.g., keys that maintain status after being released). The most common toggling keys include Caps Lock, Num Lock, and Scroll Lock.

SoundSentry
: These monitor the operating system and applications for sounds, and attempt to provide a graphical indication when a sound is being played. Older versions of

Sound Sentry may have flashed the entire display screen for example, while newer versions of SoundSentry provide the user with a selection of options, such as flashing the active window or flashing the active window caption bar.

The next three built in accessibility features are not as commonly available as the above group of features, but are included here for definition, completeness, and future compatibility.

ShowSounds
: These are user setting or software switches that are available for the operating system and application (including user agents) APIs to read, to notify them that the user wishes audio information to also be presented in a graphical format.

High Contrast
: These automatically change the display fonts and colors to choices which should provide for easier reading.

TimeOut
: These allow the built-in accessibility features to automatically turn off if the computer is unused for a specified length of time, and is intended for use when the computer is in a public setting (e.g., library). TimeOut might also be referred to as reset or automatic reset.

The next accessibility feature listed here is not considered to be a built in accessibility feature (since it only provides an alternate input channel) and is presented here only for definition, completeness, and future compatibility.

SerialKeys
: These allow a user to perform all keyboard and mouse functions from an external assistive device (such as communication aid) communicating with the computer via a serial character stream (e.g., serial port, IR port, etc.) rather than or in conjunction with, the keyboard, mouse, and other standard input devices/methods.

Microsoft Windows 95, Windows 98, and Window NT 4.0

For information about Microsoft keyboard configurations (Internet Explorer, Windows 95, Windows 98, and more), refer to [MS-KEYBOARD] [p. 50] .

The following accessibility features can be adjusted from the Accessibility Options Control Panel:

- StickyKeys: modifier keys include SHIFT, CONTROL, and ALTERNATE. Keyboard Mappings: 5 consecutive clicks of Shift key turns on/off StickyKeys.
- FilterKeys: grouping term for SlowKeys, RepeatKeys, and BounceKeys. Shift key held down for 8 seconds turns on/off SlowKeys and RepeatKeys.
- MouseKeys: Left shift + left alt + numlock, turns on/off MouseKeys
- ToggleKeys: Numlock key held for 5 seconds, turns on/off ToggleKeys
- SoundSentry:
- ShowSounds:

- Automatic reset: term used for TimeOut
- High Contrast: left shift + left alt + print screen, turns on/off High Contrast.
- SerialKeys:

Other keyboard shortcuts:

- 6 consecutive clicks of Control key turns on/off screen reader numeric keypad.
- 6 consecutive clicks of Alt key reserved for future use.

Additional accessibility features available in Windows 98:

Magnifier
: This is a windowed, screen enlargement and enhancement program used by persons with low vision to magnify an area of the graphical display (e.g., by tracking the text cursor, focus, etc.). Magnifier can also invert the colors used by the system within the magnification window.

Accessibility Wizard
: This is a setup tool intended to assist a person with making choices which setting up the accessibility features on a workstation.

Apple Macintosh Operating System

The following accessibility features can be adjusted from the Easy Access Control panel (Note: Apple convention uses a space within the accessibility feature names.)

- Sticky Keys: modifier keys include the SHIFT, OPEN APPLE (COMMAND), OPTION (ALT) and CONTROL keys.
- Slow Keys:
- Mouse Keys:

The following accessibility features can be adjusted from the Keyboard Control Panel.

- Key Repeat Rate (e.g., part of RepeatKeys)
- Delay Unit Repeat (e.g., part of RepeatKeys)

The following accessibility feature can be adjusted from the Sound or Monitors and Sound Control Panel (depends upon which version of the OS).

- Adjusting the volume to off or mute causes the Macintosh to flash the title bar whenever the operating system detects a sound (e.g., SoundSentry)

Additional accessibility features available for the Macintosh OS:

CloseView
: This is a full screen, screen enlargement and enhancement program used by persons with low vision to magnify the information on the graphical display, and it can also change the colors used by the system.

SerialKeys
: This is available as freeware from Apple and several other Web sites.

AccessX, X Keyboard Extension (XKB), and the X Window System

(Note: AccessX became a supported part of the X Window System X Server with the release of the X Keyboard Extension in version X11R6.1)

The following accessibility features can be adjusted from the AccessX graphical user interface X client on some DEC, SUN, and SGI operating systems. Other systems supporting XKB may require the user to manipulate the features via a command line parameter(s).

- StickyKeys: modifier keys are platform dependent, but usually include the SHIFT, CONTROL, and META keys.
- RepeatKeys:
- SlowKeys:
- BounceKeys:
- MouseKeys:
- ToggleKeys:

DOS (Disk Operating System)

The following accessibility features are available from a freeware program called AccessDOS, which is available from several Internet Web sites including IBM, Microsoft, and the Trace Center, for either PC-DOS or MS-DOS versions 3.3 or higher.

- StickyKeys: modifier keys include the SHIFT, CONTROL, and ALTERNATE keys.
- Keyboard Response Group: grouping term for SlowKeys, RepeatKeys, and BounceKeys
- MouseKeys:
- ToggleKeys:
- SoundSentry (incorrectly name ShowSounds):
- SerialKeys:
- TimeOut:

9 Appendix: Loading assistive technologies for DOM access

There are several methods for developers to accomplish this. Most of these methods fall into four categories:

1. Launch the entire AT inside the address space of the (UA) browser
2. Launch some part of the AT, a piece of stub code, a DLL, a Browser Helper Object [special DLL], etc., inside the address space of the (UA) browser
3. Write your own combined UA/AT (e.g., pwWebSpeak)
4. Out-of-process access to the DOM

These methods are ordered as developments within a rapidly changing technology with the most recent advances/methods listed first.

Loading assistive technologies for direct access to User Agent DOMs

Note. This method and the method described in the next section are very similar. What differs is the amount of, or capability of, the AT that actually gets loaded in the same process or address space as the User Agent.)

Access to application specific data across process boundaries might be costly in terms of performance. Therefore, user agents may wish to provide a mechanism to load the entire assistive technology (AT), into the process space of the application as a separate thread with direct access to the DOM.

Determining the Assistive Technologies to load

One technique is to store a reference to an assistive technology in a system registry file or, in the case of Jave, a properties file. Registry files are common among many operating system platforms.

In Windows you have the system registry file. On OS/2 you have the `system.ini` file and on distributed network client networks you often have a system registry server that an application running on the network client computer can query.

In Java 2, the existence of an "accessibility.properties" file causes the system event queue to examine the file for assistive technologies required for loading. If the file contains a property called "assistive_technologies", it will load all registered assistive technologies and start them on their own thread in the Java Virtual Machine that is a single process. An example entry for Java is as follows:

```
assistive_technologies=com.ibm.sns.svk.AccessEngine
```

In Windows, a similar technique could be followed by storing the name of a Dynamic Link Library (DLL) for an assistive technology in a designated assistive technology key name, AT pair. An example entry for Windows could be as follows:

```
HKEY_LOCAL_MACHINE\Software\Accessibility\DOM
       "ScreenReader, VoiceNavigation"
```

Attaching the Assistive Technologies to the DOM.

Once the assistive technology is determined from the registry, any user agent on the given operating system can now determine if an assistive technology needs to be loaded with their application and load it.

On a non-Java platform, a technique to do this would be to create a separate thread with a reference to the User Agent's DOM using a Dynamic Link Library (DLL). This new thread will load the DLL and call a specified DLL entry name with a pointer to the DOM interface. The assistive technology's task will then run until such time as is necessary to end communication with the DOM.

Once loaded, the assistive technology can monitor the DOM as needed. The assistive technology has the option of communicating with a main assistive technology of its own and process the DOM as a caching mechanism for the main AT application or be used as a bridge to the DOM for the main assistive technology.

In the future, it will be necessary to provide a more comprehensive reference to the application that not only provides direct access to it's client area DOM, but also multiple DOM's that it is processing and an event model for monitoring them.

Example Technique: Java's Direct Access

Java is a working example where the direct access to application components is performed in a timely manner. Here, an assistive technology running on a separate thread monitors GUI events such as focus changes. Focus changes give the AT notification of which component object has focus. The AT can communicate directly with all components in the application by walking the parent/child hierarchy and connecting to each component's methods and monitor events directly. In this case an AT has direct access to component specific methods as well as those provided for by the Java Accessibility API. There is no reason that a DOM interface to UA components could not be provided

In Java 1.1.x, Sun's Java access utilities load an assistive by monitoring the Java awt.properties file for the presence of assistive technologies and loads them as shown in the following code example:

```
import java.awt.*;
import java.util.*;

String atNames = Toolkit.getProperty("AWT.assistive_technologies",null);
if (atNames != null) {
    StringTokenizer parser = new StringTokenizer(atNames," ,");
    String atName;
    while (parser.hasMoreTokens()) {
       atName = parser.nextToken();
       try {
          Class.forName(atName).newInstance();
       }
       catch (ClassNotFoundException e) {
          throw new AWTError("Assistive Technology not found: " + atName);
       }
       catch (InstantiationException e) {
```

```
            throw new AWTError("Could not instantiate Assistive" +
                               " Technology: " + atName);
         }
         catch (IllegalAccessException e) {
            throw new AWTError("Could not access Assistive" +
                               " Technology: " + atName);
         } catch (Exception e) {
            throw new AWTError("Error trying to install Assistive" +
                               " Technology: " + atName + " " + e);
         }
      }
   }
```

In the above code example, the function Class.forName(atName).newInstance() creates a new instance of the assistive technology. The constructor for the assistive technology will then be responsible for monitoring application component objects by monitoring system events.

In the following code example, the constructor for the assistive technology "Access Engine," adds a focus change listener using Java accessibility utilities. When the assistive technology is notified of an objects gaining focus it has direct access to that object. If the Object, o, implemented a DOM interface the assistive technology would now have direct access to the DOM in the same process space as the application.

```
import java.awt.*;
import javax.accessibility.*;
import com.sun.java.accessibility.util.*;
import java.awt.event.FocusListener;

class AccessEngine implements FocusListener {
   public AccessEngine() {
      //Add the AccessEngine as a focus change listener
      SwingEventMonitor.addFocusListener((FocusListener)this);
   }

   public void focusGained(FocusEvent theEvent) {
      // get the component object source
      Object o = theEvent.getSource();
      // check to see if this is a DOM component
      if (o instanceof DOM) {
         ...
      }
   }
   public void focusLost(FocusEvent theEvent) {
      // Do Nothing
   }
}
```

In this example, the assistive technology has the option of running standalone or acting as a cache for a bridge that communicates with a main assistive technology running outside the Java virtual machine.

Loading part of the assistive technologies for direct access to User Agent DOMs

Access to application specific data across process boundaries might be costly in terms of performance. Therefore, user agents may wish to provide a mechanism to load part of the assistive technology (AT) into the process space of the application as a separate thread, with direct access to the DOM, to provide the specific functionality they require. This could consist of a piece of stub code, a DLL, a Browser Helper Object, etc. An example of how to do this follows.

Browser Helper Objects

In order to attach to a running instance of Internet Explorer 4.0, you can use a "Browser Helper Object." A "Browser Helper Object" is a DLL that will attach itself to every new instance of Internet Explorer 4.0 (only if you explicitly run iexplore.exe). You can use this feature to gain access to the object model of a particular running instance of Internet Explorer. You can also use this feature to get events from an instance of Internet Explorer 4.0. This can be tremendously helpful when many method calls need to be made to IE, as each call will be performed much more quickly than the out of process case.

There are some requirements when creating a Browser Helper Object

- The application that you create must be an in-proc server (that is, DLL).
- This DLL must implement IObjectWithSite.
- The IObjectWithSite::SetSite() method must be implemented. It is through this method that your application receives a pointer to Internet Explorer's IUnknown. (Internet Explorer actually passes a pointer to IWebBrowser2 but the implementation of SetSite() receives a pointer to IUnknown.) You can use this IUnknown pointer to automate Internet Explorer or to sink events from Internet Explorer.
- It must be registered as a Browser Helper Object as described above.

For more information, please check out:

http://support.microsoft.com/support/kb/articles/Q179/2/30.asp.
http://msdn.microsoft.com/library/techart/bho.htm

Java Access Bridge

In order for native Windows ATs to gain access to Java applications without the creating a Java native solution Sun Microsystems provides the "Java Access Bridge." This bridge is loaded as an AT as described in section 6.1.3. The bridge uses a Java Native Invocation (JNI) to Dynamic Link Library) (DLL) communication and caching mechanism that allows a native assistive technology to gather and monitor accessibility information in the Java environment. In this environment, the AT determines that a Java application or applet is running and communicates with the Java Access Bridge DLL to process accessibility information about the application/applet running in the Java Virtual Machine.

Loading assistive technologies "as" the User Agent with access to the DOMs

Specialized user agents might also include the necessary assistive technology as part of their interface, and thus provide possibly the best of both worlds. An example would be pwWebSpeak, from The Productivity Works (refer to [Prodworks] [p. 50]).

[Ed. Have The Productivity Works provide short description.]

Loading assistive technologies for indirect access to User Agent DOMs

Access to application specific data across process boundaries or address space might be costly in terms of performance. However, there are other reasons to consider when accessing the User Agent DOM that might lead a developer to wish to access the DOM from their own process or memory address space. One obvious protection this method provides, is that if the User Agent application fails, it doesn't disable the user's AT as well. Another consideration would be legacy systems, where the user relies on their AT for access to other applications as well as the User Agent, and thus would have their AT loaded all the time, not just for accessing the User Agent.

There are several ways to gain access to the User Agent's DOM. Most User Agents support some kind of external interface, or act as a mini-server to other applications running on the desktop. Internet Explorer is a good example of this, as IE can behave as a component object model (COM) server to other applications. Mozilla, the open source release of Navigator also supports cross platform COM (XPCOM).

An example of using COM to access the IE Object Model can be seen in the code snippet below. This is an example of how to use COM to get a pointer to the WebBrowser2 module, which in turn allows you to get a interface/pointer to the document object, or IE DOM for the web page in view.

```
/* first, get a pointer to the WebBrowser2 control */
if (m_pIE == NULL) {
   hr = CoCreateInstance(CLSID_InternetExplorer,
        NULL, CLSCTX_LOCAL_SERVER, IID_IWebBrowser2,
        (void**)&m_pIE);

   /* next, get a interface/pointer to the document in view,
      this is an interface to the document object model (DOM)*/

   void CHelpdbDlg::Digest_Document() {
      HRESULT hr;
      if (m_pIE != NULL) {
         IDispatch* pDisp;
         hr = m_pIE->QueryInterface(IID_IDispatch, (void**) &pDisp);
         if (SUCCEEDED(hr)) {

            IDispatch* lDisp;
```

```
                    hr = m_pIE->get_Document(&lDisp);
                    if (SUCCEEDED(hr)) {

                        IHTMLDocument2* pHTMLDocument2;
                        hr = lDisp->QueryInterface(IID_IHTMLDocument2,
                                     (void**) &pHTMLDocument2);
                        if (SUCCEEDED(hr)) {

                        /* with this interface/pointer, IHTMLDocument2*,
                           you can then work on the document */
                           IHTMLElementCollection* pColl;
                           hr = pHTMLDocument2->get_all(&pColl);
                           if (SUCCEEDED(hr)) {

                              LONG c_elem;
                              hr = pColl->get_length(&c_elem);
                              if (SUCCEEDED(hr)) {
                                 FindElements(c_elem, pColl);
                              }
                              pColl->Release();
                           }
                           pHTMLDocument2->Release();
                        }
                        lDisp->Release();
                    }
                    pDisp->Release();
                 }
              }
           }
        }
```

For more information on using COM with IE, please visit the Microsoft web site:

http://www.microsoft.com/com/default.asp

For more information on using XPCOM with Mozilla, please visit the Mozilla web site:

http://www.mozilla.org/

For a working example of the method described in 6.1.4, please visit the following web site and review HelpDB, developed as a testing tool for web table navigation.

http://trace.wisc.edu/world/web/document_access/

10 Appendix: Checkpoint Map

This index lists each checkpoint and the sections in this document where it is discussed. Furthermore, each guideline number links to its definition in the guidelines document. Each checkpoint also links to its definition in the guidelines document.

Guideline 1:

1.1 Ensure that all functionalities offered through the user interface may be operated through standard input device APIs supported by the operating system. [Priority 1] (Checkpoint 1.1 in guidelines)
Techniques are in section 3.2 Device independence [p. 7]
1.2 Ensure that the user can interact with all active elements of a document in a device independent manner. [Priority 1] (Checkpoint 1.2 in guidelines)
Techniques are in section 3.2 Device independence [p. 7]
1.3 Ensure that the user can install the user agent software in a device independent manner. [Priority 1] (Checkpoint 1.3 in guidelines)
Techniques are in section 3.2 Device independence [p. 7]
1.4 Ensure that the user can configure the user agent in a device independent manner. [Priority 1] (Checkpoint 1.4 in guidelines)
Techniques are in section 3.2 Device independence [p. 7]
1.5 Ensure that the user can access user agent documentation in a device independent manner. [Priority 1] (Checkpoint 1.5 in guidelines)
Techniques are in section 3.2 Device independence [p. 7]
1.6 Ensure that all messages to the user (e.g., warnings, errors, etc.) are available through standard output device APIs supported by the operating system. [Priority 1] (Checkpoint 1.6 in guidelines)
Techniques are in section 3.2 Device independence [p. 7]

Guideline 2:

2.1 By default and without additional customization, ensure that all functionalities offered by the user agent are accessible using the keyboard. [Priority 1] (Checkpoint 2.1 in guidelines)
Techniques are in section 3.7 Keyboard access [p. 17]
2.2 Provide documentation on default keyboard commands and include with user agent documentation and/or user help system. [Priority 1] (Checkpoint 2.2 in guidelines)
Techniques are in section 3.7.2 Keyboard shortcuts [p. 17]
2.3 Provide information to the user about the current keyboard configuration. [Priority 1] (Checkpoint 2.3 in guidelines)
Techniques are in section 3.7.2 Keyboard shortcuts [p. 17]
2.4 Allow the user to configure the keystrokes used to activate user agent functionalities. Wherever possible, allow single key activation of functions. [Priority 2] (Checkpoint 2.4 in guidelines)
Techniques are in section 3.8.2 Keyboard configuration [p. 18]
2.5 Allow the user to turn on and off author-specified keyboard configurations. [Priority 2] (Checkpoint 2.5 in guidelines)
Techniques are in section 3.8.2 Keyboard configuration [p. 18]
2.6 Use platform conventions to indicate which keys activate which user agent functionalities. [Priority 2] (Checkpoint 2.6 in guidelines)
Techniques are in section 3.7.2 Keyboard shortcuts [p. 17]

2.7 Avoid default keyboard configurations that interfere with system conventions. [Priority 2] (Checkpoint 2.7 in guidelines)
    Techniques are in section 3.8.2 Keyboard configuration [p. 18]
2.8 Provide a default keyboard configuration for frequently performed operations. [Priority 3] (Checkpoint 2.8 in guidelines)
    Techniques are in section 3.8.2 Keyboard configuration [p. 18]

Guideline 3:

3.1 Ensure that the user has access to document content, including alternative representations of content. [Priority 1] (Checkpoint 3.1 in guidelines)
    Techniques are in section 3.1 Access to content [p. 5] and
    5.1 Equivalent information [p. 24]
3.2 *For dependent user agents only.* Ensure that the user has access to the content of an element selected by the user. [Priority 1] (Checkpoint 3.2 in guidelines)
    Techniques are in section 3.1 Access to content [p. 5]
3.3 *For dependent user agents only.* Render content according to natural language identification. For unsupported natural languages, notify the user of language changes when configured to do so. [Priority 1] (Checkpoint 3.3 in guidelines)
    Techniques are in section 3.1.3 Changes in content language [p. 7]
3.4 Provide time-independent access to time-dependent active elements or allow the user to control the timing of changes. [Priority 1] (Checkpoint 3.4 in guidelines)
    Techniques are in section 3.1.4 Changes over time [p. 7]
3.5 When no alternative text representation has been specified, indicate what type of object is present. [Priority 2] (Checkpoint 3.5 in guidelines)
    Techniques are in section 5.1.2 Alternative representations of information for images, video, applets [p. 24]
3.6 When alternative text has been specified explicitly as empty (i.e., an empty string), render nothing. [Priority 3] (Checkpoint 3.6 in guidelines)
    Techniques are in section 5.1.2 Alternative representations of information for images, video, applets [p. 24]
3.7 Allow the user to specify that description tracks (e.g., caption, auditory description, video of sign language, etc.) be rendered at the same time as audio and video tracks. [Priority 1] (Checkpoint 3.7 in guidelines)
    Techniques are in section 5.1.3 Textual equivalents for audio and video [p. 24] and
    5.1.4 Auditory equivalents for video [p. 25]
3.8 If a technology allows for more than one description track (e.g., caption, auditory description, video of sign language, etc.), allow the user to choose from among the tracks. [Priority 1] (Checkpoint 3.8 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7] and
    3.3 User control of style [p. 7]
3.9 If a technology allows for more than one audio track, allow the user to choose from among tracks. [Priority 1] (Checkpoint 3.9 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]

Guideline 4:

4.1 Allow the user to turn on and off rendering of images. [Priority 1] (Checkpoint 4.1 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]
4.2 Allow the user to turn on and off rendering of background images. [Priority 1] (Checkpoint 4.2 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
4.3 Allow the user to turn on and off rendering of video. [Priority 1] (Checkpoint 4.3 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]
4.4 Allow the user to turn on and off rendering of sound. [Priority 1] (Checkpoint 4.4 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]
4.5 Allow the user to turn on and off rendering of audio captions. [Priority 1] (Checkpoint 4.5 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]
4.6 Allow the user to turn on and off animated or blinking text. [Priority 1] (Checkpoint 4.6 in guidelines)
    Techniques are in section 5.1.1 Sources of blinking text and animation [p. 24]
4.7 Allow the user to turn on and off animations and blinking images. [Priority 1] (Checkpoint 4.7 in guidelines)
    Techniques are in section 5.1.1 Sources of blinking text and animation [p. 24]
4.8 Allow the user to turn on and off support for scripts and applets. [Priority 1] (Checkpoint 4.8 in guidelines)
    Techniques are in section 5.5 Scripts [p. 28]
4.9 Allow the user to turn on and off support for user style sheets. [Priority 1] (Checkpoint 4.9 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]
4.10 Allow the user to turn on and off support for author style sheets. [Priority 1] (Checkpoint 4.10 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]
4.11 Allow the user to turn on and off support for spawned windows. [Priority 1] (Checkpoint 4.11 in guidelines)
    Techniques are in section 5.1.5 Frame equivalents [p. 25]
4.12 Allow the user to turn on and off rendering of frames. [Priority 2] (Checkpoint 4.12 in guidelines)
    Techniques are in section 5.1.5 Frame equivalents [p. 25]
4.13 Allow the user to turn on and off author-specified page forwards that occur after a time delay and without user intervention. [Priority 3] (Checkpoint 4.13 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]
4.14 Allow the user to turn on and off automatic page refresh. [Priority 3] (Checkpoint 4.14 in guidelines)
    Techniques are in section 3.3.1 Feature control [p. 9]

Guideline 5:

5.1 Allow the user to control font family. [Priority 1] (Checkpoint 5.1 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.2 Allow the user to control the size of text. [Priority 1] (Checkpoint 5.2 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.3 Allow the user to control foreground color. [Priority 1] (Checkpoint 5.3 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.4 Allow the user to control background color. [Priority 1] (Checkpoint 5.4 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.5 Allow the user to control selection highlighting (e.g., foreground and background color). [Priority 1] (Checkpoint 5.5 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.6 Allow the user to control focus highlighting (e.g., foreground and background color). [Priority 1] (Checkpoint 5.6 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.7 Allow the user to control animation rate. [Priority 2] (Checkpoint 5.7 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.8 Allow the user to control video frame rates. [Priority 1] (Checkpoint 5.8 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.9 Allow the user to control the position of audio captions. [Priority 1] (Checkpoint 5.9 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7] and
    5.1.3 Textual equivalents for audio and video [p. 24]
5.10 Allow the user to start, stop, pause, and rewind video. [Priority 2] (Checkpoint 5.10 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.11 Allow the user to control audio playback rate. [Priority 1] (Checkpoint 5.11 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.12 When the user agent renders audio natively, allow the user to control the audio volume. [Priority 2] (Checkpoint 5.12 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.13 Allow the user to start, stop, pause, and rewind audio. [Priority 2] (Checkpoint 5.13 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.14 Allow the user to control speech playback rate. [Priority 1] (Checkpoint 5.14 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]
5.15 Allow the user to control speech volume, pitch, gender and other articulation characteristics. [Priority 2] (Checkpoint 5.15 in guidelines)
    Techniques are in section 3.3 User control of style [p. 7]

5.16 When new windows or user interface components are spawned, allow the user to control window size and position. [Priority 2] (Checkpoint 5.16 in guidelines)

Techniques are in section 3.3 User control of style [p. 7] and
5.4 Frames [p. 28]

Guideline 6:

6.1 Use and provide accessible interfaces to other technologies. [Priority 1] (Checkpoint 6.1 in guidelines)

Techniques are in section 4.3 Accessibility interfaces [p. 22]

6.2 Provide programmatic read and write access to user agent functionalities and user interface controls (including selection and focus) by using operating system and development language accessibility resources and conventions. [Priority 1] (Checkpoint 6.2 in guidelines)

Techniques are in section 4.3 Accessibility interfaces [p. 22]

6.3 Notify dependent user agents of changes to the document and user interface controls (including selection and focus) by using operating system and development language accessibility resources and conventions. [Priority 1] (Checkpoint 6.3 in guidelines)

Techniques are in section 4.3 Accessibility interfaces [p. 22]

6.4 *For graphical desktop browsers only.* Comply with W3C Document Object Model specifications and export interfaces defined by those specifications. [Priority 1] (Checkpoint 6.4 in guidelines)

Techniques are in section 4.4 The document object model [p. 22]

6.5 *For graphical desktop browsers only.* Provide programmatic exchange of information in a timely manner. [Priority 2] (Checkpoint 6.5 in guidelines)

Techniques are in section 4.3 Accessibility interfaces [p. 22]

6.6 Follow operating system conventions and accessibility settings. In particular, follow conventions for user interface design, default keyboard configuration, product installation, and documentation. [Priority 2] (Checkpoint 6.6 in guidelines)

Techniques are in section 4.1 System conventions [p. 20]

Guideline 7:

7.1 Implement the accessibility features defined for supported specifications. [Priority 1] (Checkpoint 7.1 in guidelines)

Techniques are in section 5 Support for HTML accessibility [p. 23] and
6 Support for CSS accessibility [p. 29] and
7 Support for SMIL accessibility [p. 29]

7.2 Support appropriate W3C Recommendations. [Priority 2] (Checkpoint 7.2 in guidelines)

Techniques are in section 5 Support for HTML accessibility [p. 23] and
6 Support for CSS accessibility [p. 29] and
7 Support for SMIL accessibility [p. 29]

Guideline 8:

8.1 Allow the user to navigate viewports (including frames). [Priority 1] (Checkpoint 8.1 in guidelines)

Techniques are in section 3.5.5 View navigation [p. 14]

8.2 For user agents that offer a browsing history mechanism, when the user returns to a previous view, restore the point of regard in the viewport. [Priority 1] (Checkpoint 8.2 in guidelines)

Techniques are in section 3.4.2 Tracking selection and focus [p. 10]

8.3 *For dependent user agents only.* Allow the user to navigate just among table cells of a table (notably left and right within a row and up and down within a column). [Priority 1] (Checkpoint 8.3 in guidelines)

Techniques are in section 3.5.3 Table navigation [p. 12]

8.4 Allow the user to navigate just among all active elements in the document. [Priority 2] (Checkpoint 8.4 in guidelines)

Techniques are in section 3.5.1 Navigation of active elements [p. 10]

8.5 Allow the user to search for rendered text content, including alternative text content. [Priority 2] (Checkpoint 8.5 in guidelines)

Techniques are in section 3.5.4 Searching [p. 13]

8.6 Allow the user to navigate the document structure. [Priority 2] (Checkpoint 8.6 in guidelines)

Techniques are in section 3.5.2 Navigation of document structure [p. 11]

8.7 Allow the user to configure structured navigation. [Priority 3] (Checkpoint 8.7 in guidelines)

Techniques are in section 3.5 Navigation and searching [p. 10]

Guideline 9:

9.1 Provide a mechanism for highlighting and identifying (through a standard interface where available) the current viewport, selection, and focus. [Priority 1] (Checkpoint 9.1 in guidelines)

Techniques are in section 3.4.1 Highlighting the viewport, selection, and focus [p. 10]

9.2 *For dependent user agents only.* Provide the user with information about the number of viewports. [Priority 2] (Checkpoint 9.2 in guidelines)

Techniques are in section 5.4 Frames [p. 28]

9.3 *For dependent user agents only.* Allow the user to view a document outline constructed from its structural elements (e.g., from header and list elements). [Priority 2] (Checkpoint 9.3 in guidelines)

Techniques are in section 3.5.5 View navigation [p. 14]

9.4 Describe a selected element's context within a document (e.g., numerical or relative position). [Priority 2] (Checkpoint 9.4 in guidelines)

Techniques are in section 3.6.2 Element context [p. 14]

9.5 For a selected link, indicate whether following the link will involve a fee. [Priority 2] (Checkpoint 9.5 in guidelines)

Techniques are in section 3.6.3 Links [p. 14]

9.6 For a selected link, provide information to help the user decide whether to follow the link. [Priority 3] (Checkpoint 9.6 in guidelines)
Techniques are in section 3.6.3 Links [p. 14]
9.7 Provide a mechanism for highlighting and identifying (through a standard interface where available) active elements of a document. [Priority 3] (Checkpoint 9.7 in guidelines)
Techniques are in section 3.6.5 Form controls [p. 14] and
5.2 Links [p. 25]
9.8 *For dependent user agents only.* Provide access to header information for a selected table cell. [Priority 1] (Checkpoint 9.8 in guidelines)
Techniques are in section 3.6.4 Tables [p. 14]
9.9 *For dependent user agents only.* Indicate the row and column dimensions of a selected table. [Priority 3] (Checkpoint 9.9 in guidelines)
Techniques are in section 3.6.4 Tables [p. 14]
9.10 Provide the user with access to any label explicitly associated with a form control. [Priority 2] (Checkpoint 9.10 in guidelines)
Techniques are in section 3.6.5 Form controls [p. 14]
9.11 Maintain consistent user agent behavior and default configurations between software releases. Consistency is less important than accessibility and adoption of system conventions. [Priority 3] (Checkpoint 9.11 in guidelines)
Techniques are in section 3.7.3 Software consistency [p. 17]

Guideline 10:

10.1 Provide information about document and viewport changes (to users and through programming interfaces). [Priority 1] (Checkpoint 10.1 in guidelines)
Techniques are in section 3.6.7 Scripts [p. 17]
10.2 Ensure that when the selection or focus changes, it is in the viewport after the change. [Priority 2] (Checkpoint 10.2 in guidelines)
Techniques are in section 3.4.2 Tracking selection and focus [p. 10]
10.3 Allow the user to selectively turn on and off notification of common types of document and viewport changes. [Priority 3] (Checkpoint 10.3 in guidelines)
Techniques are in section 3.6.7 Scripts [p. 17]
10.4 When loading a resource (e.g., document, video clip, audio clip, etc.) indicate what portion of the resource has loaded and whether loading has stalled. [Priority 3] (Checkpoint 10.4 in guidelines)
Techniques are in section 3.6.1 Status information [p. 14]
10.5 Indicate the relative position of the viewport in a resource (e.g., the percentage of the document that has been viewed, the percentage of an audio clip that has been played, etc.). [Priority 3] (Checkpoint 10.5 in guidelines)
Techniques are in section
10.6 Prompt the user to confirm any form submission not explicitly initiated by the user. [Priority 2] (Checkpoint 10.6 in guidelines)
Techniques are in section 3.6.5 Form controls [p. 14]

Guideline 11:

11.1 Allow the user to configure the user agent in named profiles that may be shared (by other users or software). [Priority 2] (Checkpoint 11.1 in guidelines)
    Techniques are in section 3.8.1 User profiles [p. 18]
11.2 Allow the user to configure the graphical arrangement of user interface controls. [Priority 3] (Checkpoint 11.2 in guidelines)
    Techniques are in section 3.8.3 User interface [p. 18]

Guideline 12:

12.1 Provide a version of the product documentation that conforms to the Web Content Accessibility Guidelines. [Priority 1] (Checkpoint 12.1 in guidelines)
    Techniques are in section 3.9.2 Accessible documentation [p. 19]
12.2 Ensure that all user agent functionalities that promote accessibility are documented. [Priority 1] (Checkpoint 12.2 in guidelines)
    Techniques are in section 3.9 Documentation [p. 18]
12.3 Describe product features known to promote accessibility in a section of the product documentation. [Priority 2] (Checkpoint 12.3 in guidelines)
    Techniques are in section 3.9.1 Where to document accessibility [p. 19]

Acknowledgments

Many thanks to the following people who have contributed through review and comment: Paul Adelson, James Allan, Denis Anson, Kitch Barnicle, Harvey Bingham, Olivier Borius, Judy Brewer, Bryan Campbell, Kevin Carey, Wendy Chisholm, David Clark, Chetz Colwell, Wilson Craig, Nir Dagan, Daniel Dardailler, B. K. Delong, Neal Ewers, Geoff Freed, John Gardner, Al Gilman, Larry Goldberg, Glen Gordon, John Grotting, Markku Hakkinen, Earle Harrison, Chris Hasser, Kathy Hewitt, Philipp Hoschka, Masayasu Ishikawa, Phill Jenkins, Jan Kärrman (for help with html2ps), Leonard Kasday, George Kerscher, Marja-Riitta Koivunen, Josh Krieger, Catherine Laws, Greg Lowney, Scott Luebking, William Loughborough, Napoleon Maou, Charles McCathieNevile, Masafumi Nakane, Mark Novak, Charles Oppermann, Mike Paciello, David Pawson, Michael Pederson, Helen Petrie, David Poehlman, Michael Pieper, Jan Richards, Hans Riesebos, Joe Roeder, Lakespur L. Roca, Gregory Rosmaita, Lloyd Rutledge, Liam Quinn, T.V. Raman, Robert Savellis, Rich Schwerdtfeger, Constantine Stephanidis, Jim Thatcher, Jutta Treviranus, Claus Thogersen, Steve Tyler, Gregg Vanderheiden, Jaap van Lelieveld, Jon S. von Tetzchner, Willie Walker, Ben Weiss, Evan Wies, Chris Wilson, Henk Wittingen, and Tom Wlodkowski,

References

[CSS1]
"CSS, level 1 Recommendation", B. Bos, H. Wium Lie, eds. The CSS1 Recommendation is: http://www.w3.org/TR/1999/REC-CSS1-19990111.

[CSS2]
"CSS, level 2 Recommendation", B. Bos, H. Wium Lie, C. Lilley, and I. Jacobs, eds. The CSS2 Recommendation is: http://www.w3.org/TR/1998/REC-CSS2-19980512.

[CSS-ACCESS]
"Accessibility Features of CSS", I. Jacobs, J. Brewer, eds. The latest version of this W3C Note is available at: http://www.w3.org/TR/CSS-access.

[DOM1]
"Document Object Model (DOM) Level 1 Specification", V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood, eds. The DOM Level 1 Recommendation is: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.

[HTML40]
"HTML 4.0 Recommendation", D. Raggett, A. Le Hors, and I. Jacobs, eds. The HTML 4.0 Recommendation is: http://www.w3.org/TR/1998/REC-html40-19980424.

[HTML32]
"HTML 3.2 Recommendation", D. Raggett, ed. The HTML 3.2 Recommendation is: http://www.w3.org/TR/REC-html32.

[MATHML]
"Mathematical Markup Language", P. Ion and R. Miner, eds. The MathML 1.0 Recommendation is: http://www.w3.org/TR/1998/REC-MathML-19980407.

[RFC2119]
"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt

[SMIL]
"Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", P. Hoschka, editor. The SMIL 1.0 Recommendation is: http://www.w3.org/TR/1998/REC-smil-19980615

[UA-TECHNIQUES]
"Techniques for User Agent Accessibility Guidelines 1.0", J. Gunderson, I. Jacobs, eds. This document explains how to implement the checkpoints defined in "User Agent Accessibility Guidelines 1.0". The latest draft of the techniques is available at: http://www.w3.org/WAI/UA/WAI-USERAGENT-TECHS/

[WAI-AUTOOLS]
"Authoring Tool Accessibility Guidelines", J. Treviranus, J. Richards, I. Jacobs, C. McCathieNevile, eds. The latest Working Draft of these guidelines for designing accessible authoring tools is available at: http://www.w3.org/TR/WD-WAI-AUTOOLS/

[WAI-WEBCONTENT]
"Web Content Accessibility Guidelines", W. Chisholm, G. Vanderheiden, and I. Jacobs, eds., 5 May 1999. This W3C Recommendation is http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505.

[XML]
"Extensible Markup Language (XML) 1.0.", T. Bray, J. Paoli, C.M. Sperberg-McQueen, eds. The XML 1.0 Recommendation is: http://www.w3.org/TR/1998/REC-xml-19980210

Services

Note. *W3C cannot maintain stability for any of the following references outside of its control. These references are included for convenience.*

[AMAYA]
Amaya is W3C's testbed browser/editor.

[APPLE-HI]
Information on accessibility guidelines for Macintosh applications. Information on Apple's scripting model can be found at tn1095 and tn1164. Refer also to the Inside Macintosh chapter devoted to Interapplication Communication.

[USERAGENTS]
List of Alternative Web Browsers. This list is maintained by WAI.

[ED-DEPT]
"Requirements for Accessible Software Design", US Department of Education, version 1.1 March 6, 1997.

[EITAAC]
"EITAAC Desktop Software standards", Electronic Information Technology Access Advisory (EITAAC) Committee.

[IBM-ACCESS]
"Software Accessibility" IBM Special Needs Systems.

[ICCCM]
"The Inter-Client communication conventions manual". A protocol for communication between clients in the X Window system.

[ICE-RAP]
"An ICE Rendezvous Mechanism for X Window System Clients", W. Walker. A description of how to use the ICE and RAP protocols for X Window clients.

[JAVA-ACCESS]
"IBM Guidelines for Writing Accessible Applications Using 100% Pure Java", R. Schwerdtfeger, IBM Special Needs Systems. Available at: http://www.austin.ibm.com/sns/snsjavag.htm

[JAVAAPI]
Information on Java Accessibility API can be found at Java Accessibility Utilities.

[JAVA-CHECKLIST]
"Java Accessibility Guidelines and Checklist". IBM Special Needs Systems.

[JAVA-TUT]
"The Java Tutorial. Trail: Creating a GUI with JFC/Swing". An online tutorial that

describes how to use the Swing Java Foundation Class to build an accessible User Interface.

[MSAA]
Information on active accessibility can be found at the Microsoft WWW site on Active Accessibility.

[MS-ENABLE]
Information on accessibility guidelines for Windows applications.

[MS-KEYBOARD]
Information on keyboard assistance for Internet Explorer and MS Windows.

[MS-SOFTWARE]
"The Microsoft Windows Guidelines for Accessible Software Design". **Note.** This is a "self-extracting archive", an application that will probably only run on MS-Windows systems.

[NOTES-ACCESS]
"Lotus Notes Accessibility Guidelines" IBM Special Needs Systems.

[Prodworks]
The Productivity Works.

[SUN-DESIGN]
"Designing for Accessibility", Eric Bergman and Earl Johnson. This paper discusses specific disabilities including those related to hearing, vision, and cognitive function.

[SUN-HCI]
"Towards Accessible Human-Computer Interaction", Eric Bergman, Earl Johnson, Sun Microsytems 1995. A substantial paper, with a valuable print bibliography.

[TRACE-REF]
"Application Software Design Guidelines" compiled by G. Vanderheiden. A thorough reference work.

[WHAT-IS]
"What is Accessible Software", James W. Thatcher, Ph.D., IBM, 1997. This paper gives a short example-based introduction to the difference between software that is accessible, and software that can be used by some assistive technologies.

[XGuidelines]
Information on accessibility guidelines for Unix and X Window applications. The Open Group has various guides that explain the Motif and Common Desktop Environment (CDE) with topics like how users interact with Motif/CDE applications and how to customize these environments. **Note.** In X, the terms client and server are used differently from their use when discussing the Web.