Techniques for User Agent Accessibility Guidelines 1.0

W3C Working Draft 28 July 2000

This version:
http://www.w3.org/WAI/UA/WD-UAAG10-TECHS-20000728
(plain text, gzip PostScript, gzip PDF, gzip tar file of HTML, zip archive of HTML)
Latest version:
http://www.w3.org/WAI/UA/UAAG10-TECHS
Previous version:
http://www.w3.org/WAI/UA/WD-UAAG10-TECHS-20000707

Editors:
Jon Gunderson, University of Illinois at Urbana-Champaign
Ian Jacobs, W3C

Abstract

This document provides techniques for satisfying the checkpoints defined in "Techniques for User Agent Accessibility Guidelines 1.0" [UAAG10] . These techniques cover the accessibility of user interfaces, content rendering, application programming interfaces (APIs), and languages such as the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and the Synchronized Multimedia Integration Language (SMIL).

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 the 28 July 2000 Working Draft of Techniques for User Agent Accessibility Guidelines 1.0, 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 participants in the WAI User Agent (UA) Working Group.

While Techniques for 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.

Please send comments about this document, including suggestions for additional techniques, to the public mailing list w3c-wai-ua@w3.org; public archives are available.

This document is part of a series of accessibility documents published by the Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C). WAI Accessibility Guidelines are produced as part of the WAI Technical Activity.

The goals of the User Agent Working Group are described in the charter. A list of the Working Group participants is available.

A list of current W3C Recommendations and other technical documents can be found at the W3C Web site.

Table of contents

Note: This document supports direct keyboard navigation to the table of contents via the "c" character. Users may have to use additional keyboard strokes depending on their operating environment. Not all user agents support direct keyboard access.

1 Introduction

This document provides some suggestions for satisfying the requirements of the "Techniques for User Agent Accessibility Guidelines 1.0" [UAAG10] . The techniques listed in this document are not required for conformance to the Guidelines. These techniques are not necessarily the only way of satisfying the checkpoint, nor are they necessarily a definitive set of requirements for satisfying a checkpoint.

1.1 How the techniques are organized

Section 2 of this document reproduces the guidelines and checkpoints of the Techniques for User Agent Accessibility Guidelines 1.0 [UAAG10] . Each checkpoint definition includes a link to the checkpoint definition in Techniques for User Agent Accessibility Guidelines 1.0 [UAAG10] . Each checkpoint definition is followed by a list of techniques, information about related resources, and references to the accessibility topics in section 3. These accessibility topics may apply to more than one checkpoint and so have been split off into a stand-alone section.

Note: Most of the techniques in this document are designed for graphical desktop browsers and multimedia players. However, some of them also make sense for assistive technologies and other user agents. In particular, techniques about communication between user agents will benefit assistive technologies. Refer, for example, to the appendix on loading assistive technologies for access to the document object model .

1.2 Related resources

"Techniques for User Agent Accessibility Guidelines 1.0" and the Guidelines [UAAG10] are part of a series of accessibility guidelines published by the Web Accessibility Initiative (WAI). The series also includes "Web Content Accessibility Guidelines 1.0" [WCAG10] (techniques [WCAG10-TECHS] , and evaluation and repair techniques [AERT]) and "Authoring Tool Accessibility Guidelines 1.0" [ATAG10] (and techniques [ATAG10-TECHS]).

1.3 Document conventions

The following editorial conventions are used throughout this document:

- HTML element names are in uppercase letters (e.g., H1, BLOCKQUOTE, TABLE, etc.)
- HTML attribute names are double-quoted in lowercase letters (e.g., "alt", "title", "class", etc.)

1.4 Priorities

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

[Priority 1]
: This checkpoint **must** be satisfied by user agents, otherwise one or more groups of users with disabilities will find it impossible to access the Web. Satisfying this checkpoint is a basic requirement for enabling some people to access the Web.

[Priority 2]
: This checkpoint **should** be satisfied by user agents, otherwise one or more groups of users with disabilities will find it difficult to access the Web. Satisfying this checkpoint will remove significant barriers to Web access for some people.

[Priority 3]
: This checkpoint **may** be satisfied by user agents to make it easier for one or more groups of users with disabilities to access information. Satisfying this checkpoint will improve access to the Web for some people.

2 User agent accessibility guidelines

This section lists each checkpoint of Techniques for User Agent Accessibility Guidelines 1.0 [UAAG10] along with some possible techniques for satisfying it. Each checkpoint also links to more general accessibility topics where appropriate.

Guideline 1. Support input and output device-independence.

Checkpoints for communication with other software:

1.1 Ensure that every functionality available through the user interface is also available through every input API implemented by the user agent. This checkpoint does not require developers to reimplement the input methods associated with the keyboard, pointing device, voice, and other input APIs . The device-independence required by this checkpoint applies to the functionalities described by the other checkpoints in this document (e.g., installation, documentation, user agent user interface configuration, etc.). [Priority 1] (Checkpoint 1.1)

Note: This checkpoint does not require developers to implement all operating system input APIs , only to make the software accessible through those they do implement. Developers are not required to reimplement input methods of APIs , e.g., text input through a mouse API or pointer motion through a keyboard API.

Techniques:

Ensure that the user can do the following with all supported input devices:

- Select content and operate on it. For example, if the user can select text with the mouse and make that text the content of a new link by pushing a button, they must also be able to do so through the keyboard and other supported devices. Other operations include cut, copy, and paste.
- Set the focus . Ensure that software may be installed, uninstalled, and updated in a device-independent manner.
- Navigate content.
- Navigate links (refer to link techniques).
- Use the graphical user interface menus.
- Fill out forms.
- Access documentation.
- Configure the software.
- Install, uninstall, and update the user agent software.

Ensure that people with disabilities are involved in the design and testing of the software.

1.2 Use the standard input and output device APIs of the operating system. Do not bypass the standard output APIs when rendering information. [Priority 1] (Checkpoint 1.2)

Note: For example, do not bypass (for reasons of speed, efficiency, etc.) standard APIs to manipulate the memory associated with rendered content , since assistive technologies monitor rendering through the APIs . When available, developers should use APIs at a higher level of abstraction than the standard device APIs for the operating system. If these higher level APIs do not use the standard device APIs properly, developers should also use the standard device APIs.

Techniques:

- Operating system and application frameworks provide standard mechanisms for communication with input devices. In the case of Windows, OS/2, the X Windows System, and Mac OS, the window manager provides Graphical User Interface (GUI) applications with this information through the messaging queue. In the case of non-GUI applications, the compiler run-time libraries provide standard mechanisms for receiving keyboard input in the case of desktop operating systems. Should you use an application framework such as the Microsoft Foundation Classes, the framework used must support the same standard input mechanisms.
- Do not communicate directly with an input device; this may circumvent system messaging. For instance, in Windows, do not open the keyboard device driver directly. It is often the case that the windowing system needs to change the form and method for processing standard input mechanisms for proper application coexistence within the user interface framework.
- Do not implement your own input device event queue mechanism; this may circumvent system messaging. Some assistive technologies use standard system facilities for simulating keyboard and mouse events. From the application's perspective, these events are no different than those generated by the user's actions. The Journal Playback Hooks (in both OS/2 and Windows) is one example of an application that feeds the standard event queues.
- Operating system and application frameworks provide standard mechanisms for using standard output devices. In the case of common desktop operating systems such as Windows, OS/2, and Mac OS, standard APIs are provided for writing to the display and the multimedia subsystems.
- Do not render text in the form of a bitmap before transferring to the screen, since some screen readers rely on the user agent's offscreen model. An offscreen model is rendered content created by an assistive technology that is based on the rendered content of another user agent. Assistive technologies that rely on an offscreen model generally construct it by intercepting standard system drawing calls. For example, in the case of display drivers, some screen readers are designed to monitor what is drawn on the screen by hooking drawing calls at different points in the drawing process. While knowing about the user agent's formatting may provide some useful information to assistive technologies, this document emphasizes access to the document object model rather than a particular rendering. For instance, instead of relying on system calls to draw text, assistive technologies should access the text through the document object

model.

- Common operating system 2D graphics engines and drawing libraries provide functions for drawing text to the screen. Examples of this are the Graphics Device Interface (GDI) for Windows, Graphics Programming Interface (GPI) for OS/2, and the X library (XLIB) for the X Windows System or Motif.
- Do not communicate directly with an output device.
- Do not draw directly to the video frame buffer.
- Do not provide your own mechanism for generating pre-defined system sounds.
- When writing textual information in a GUI operating system, use standard operating system APIs for drawing text.
- Use operating system resources for rendering audio information. When doing so, do not take exclusive control of system audio resources. This could prevent an assistive technology such as a screen reader from speaking if they use software text-to-speech conversion. Also, in operating systems like Windows, a set of standard audio sound resources are provided to support standard sounds such as alerts. These preset sounds are used to activate SoundSentry graphical cues when a problem occurs; this benefits users with hearing disabilities. These cues may be manifested by flashing the desktop, active caption bar, or active window. It is important to use the standard mechanisms to generate audio feedback so that operating system or special assistive technologies can add additional functionality for users with hearing disabilities.
- Enhance the functionality of standard system controls to improve accessibility where none is provided by responding to standard keyboard input mechanisms. For example provide keyboard navigation to menus and dialog box controls in the Apple Macintosh operating system. Another example is the Java Foundation Classes, where internal frames do not provide a keyboard mechanism to give them focus. In this case, you will need to add keyboard activation through the standard keyboard activation facility for Abstract Window Toolkit components.

1.3 Implement the standard keyboard API of the operating system and ensure that every functionality available through the user interface is available through this API. This checkpoint always applies on systems with a standard keyboard API . [Priority 1] (Checkpoint 1.3)

Note: This checkpoint is an important special case of checkpoint 1.1. Refer also to checkpoint 10.8.

Techniques:

- Apply the techniques for checkpoint 1.1 to the keyboard.
- Account for author-specified keyboard bindings, such as those specified by "accesskey" attribute in HTML 4.01 ([HTML4] , section 17.11.2).
- Allow the user to trigger event handlers (e.g., mouseover, mouseout, click,

etc.) from the keyboard.

- Test that all user interface components may be operable by software or devices that emulate a keyboard. Use SerialKeys and/or voice recognition software to test keyboard event emulation.

Checkpoints for user interface accessibility:

1.4 Ensure that the user can interact with all active elements in a device-independent manner. [Priority 1] (Checkpoint 1.4)

Note: For example, users who are blind or have physical disabilities must be able to activate text links, the links in a client-side image map, and form controls without a pointing device. This checkpoint is an important special case of checkpoint 1.1.

Techniques:

- Refer to checkpoint 1.1 and checkpoint 1.5.
- Refer to image map techniques .
- In the "Document Object Model (DOM) Level 2 Specification" [DOM2] , all elements may have associated behaviors. Assistive technologies should be able to activate these elements through the DOM. For example, a DOM 'focusin' event may cause a JavaScript function to construct a pull-down menu. Allowing programmatic activation of this function will allow users to operate the menu through speech input (which benefits users of voice browsers in addition to assistive technology users). Note that, for a given element, the same event may trigger more than one event handler, and assistive technologies must be able to activate each of them. Descriptive information about handlers can allow assistive technologies to select the most important functions for activation. This is possible in the Java Accessibility API [JAVAAPI] , which provides an an AccessibleAction Java interface. This interface provides a list of actions and descriptions that enable selective activation. Refer also to checkpoint 5.3.

1.5 Ensure every non-text message (e.g., prompt, alert, notification, etc.) that is part of the user agent's user interface also has a text equivalent in the user interface. This text equivalent must be available to assistive technologies through an API . [Priority 1] (Checkpoint 1.5)

Note: For example, if the user is notified of an event by an auditory cue, a text equivalent in the status bar would satisfy this checkpoint. Using accessible standard interface controls (per checkpoint 5.8) should make text equivalents available to assistive technologies for rendering as synthesized speech or Braille. Refer also to checkpoint 5.5.

Techniques:

- Render text messages graphically on the status bar of the user interface.
- For graphical user interface elements such as proportional scroll bars, provide a text equivalent that conveys the proportion of the content viewed (e.g., as a percentage) and that may be rendered graphically, as synthesized speech, and as Braille.
- For beeps or flashes provide a text equivalent that can be rendered as Braille, synthesized speech, or graphically-rendered text.
- For user interface components that convey important information using sound, also provide alternate, parallel graphical representation of the information for individuals who are deaf, hard of hearing, or operating the user agent in a noisy or silent environment where the use of sound is not practical. Provide Braille renderings of text equivalents for deaf-blind users who cannot use audio or graphical cues and who rely on Braille.

Guideline 2. Ensure user access to all content.

Checkpoints for content accessibility:

2.1 Make all content available through the user interface. [Priority 1] (Checkpoint 2.1)

Note: Users must have access to the entire document object through the user interface, including equivalent alternatives for content , attributes, style sheets, etc. This checkpoint does not require that all content be available in every viewport . A document source view is part of a solution for providing access to content, but is not a sufficient solution on its own. Refer to guideline 5 for more information about programmatic access to content.

Techniques:

- Some users benefit from concurrent access to primary and alternative content. For instance, users with low vision may want to view images (even imperfectly) but require a text equivalent for the image; the text may be rendered with a large font or as speech. If a multimedia presentation has several captions (or subtitles) available, allow the user to choose from among them. Captions might differ in level of detail, reading levels, natural language , etc.
- When content changes dynamically (e.g., due to scripts or content refresh), users must have access to the content before and after the change.
- Provide structured (not all at once) access to attribute values. For instance, allow the user to select and element and read values for all attributes set for that element. For many attributes, this type of inspection should be significantly more usable than a document source view.
- A document source view may be the most usable readily-achievable view for some content such as embedded fragments of style and script languages.
- Refer to the section on access to content .
- Refer to the section on link techniques .

- Refer to the section on table techniques .
- Refer to the section on frame techniques .
- Refer to the section on form techniques .
- Sections 10.4 ("Client Error 4xx") and 10.5 ("Server Error 5xx") of the HTTP 1.1 specification state that user agents should have the following behavior in case of these error conditions:

  > Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents SHOULD display any included entity to the user.

- Make available information about abbreviation and acronym expansions. For instance, in HTML, look for abbreviations specified by the ABBR and ACRONYM elements. The expansion may be given with the "title" attribute (refer to the Web Content Accessibility Guidelines 1.0 [WCAG10] , checkpoint 4.2). To provide expansion information, user agents may:
  - Allow the user to configure that the expansions be used in place of the abbreviations,
  - Provide a list of all abbreviations in the document, with their expansions (a generated glossary of sorts)
  - Generate a link from an abbreviation to its expansion.
  - Allow the user to query the expansion of a selected or input abbreviation.
  - If an acronym has no explicit expansion in one location, look for another occurrence in content with an explicit expansion. User agents may also look for possible expansions (e.g., in parentheses) in surrounding context, though that is a less reliable repair.

2.2 For a presentation that requires user input within a specified time interval, allow the user to configure the user agent to pause the presentation automatically and await user input before proceeding. [Priority 1] (Checkpoint 2.2)

Techniques:

- Render time-dependent links as a static list that occupies the same screen real estate; authors may create such documents in SMIL 1.0 [SMIL] . Include temporal context in the list of links. For example, provide the time at which the link appeared along with a way to easily jump to that portion of the presentation.
- Provide easy-to-use controls (including both mouse and keyboard commands) to allow users to pause a presentation and advance and rewind by small or large time increments. **Note:** When a user must respond to a link by pausing the program and activating the link, the time dependent nature of the link does not change since the user must respond somehow in the predetermined time. The pause feature is only effective in conjunction

with the ability to rewind to the link, or when the pause can be configured to stop the presentation automatically and require the user to respond before continuing, either by responding to the user input or by continuing with the flow of the document.

- Highlight the fact that there are active elements in a presentation and allow users to navigate to and activate them. For example, indicate the presence of active elements on the status bar and allow the user to navigate among them with the keyboard or mouse.
- For additional control, user agents may allow users to slow the presentation.

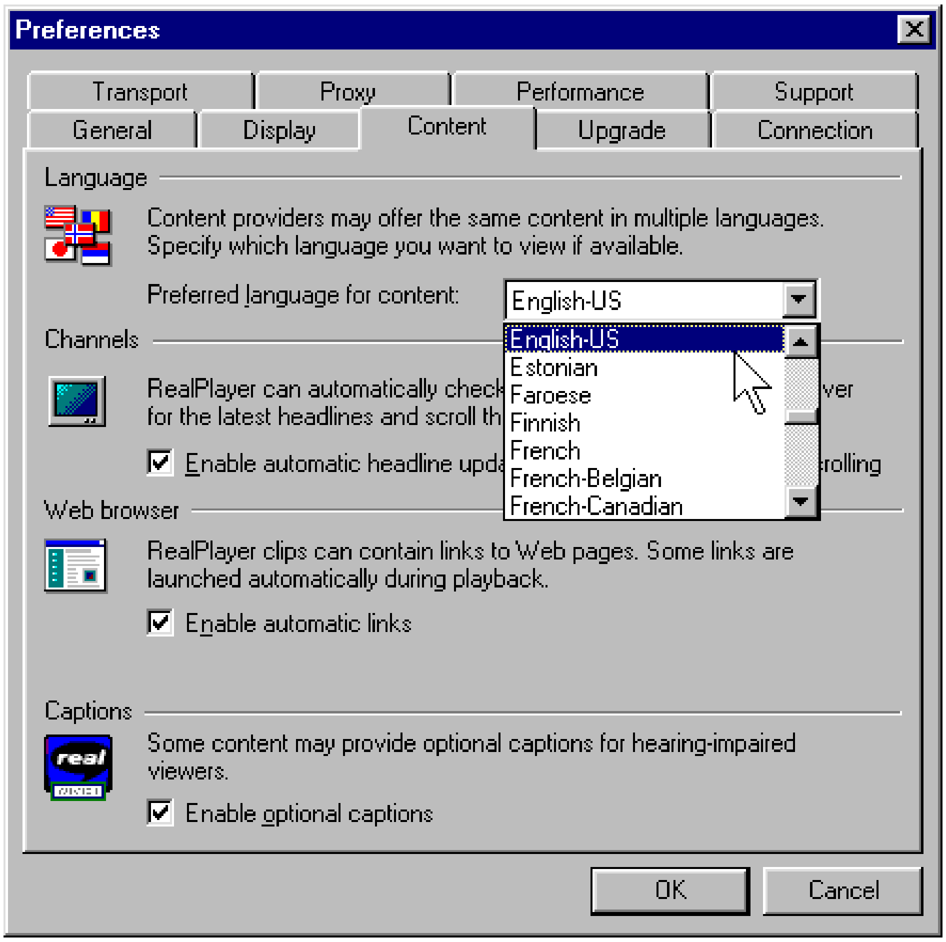
2.3 If content available in a viewport has equivalent alternatives , provide easy access in context to the alternatives. [Priority 1] (Checkpoint 2.3)

Note: For example, if an image in an HTML document has text equivalents , provide access to them by rendering them nearby, allowing the user to configure the user agent to render them in place of the image, or allowing the user to follow a readily available link to them.

Techniques:

- Refer to the section on access to content .
- Allow users to choose more than one equivalent at a given time. For instance, multilingual audiences may wish to have captions in different natural languages on the screen at the same time. Users may wish to use both captions and auditory descriptions concurrently as well.
- Make apparent through the user agent user interface which audio tracks are meant to be played mutually exclusively.
- In the user interface, construct a list of all available tracks from short descriptions provided by the author (e.g., through the "title" attribute).
- Allow the user to configure different natural language preferences for different types of equivalents (e.g., captions and auditory descriptions). Users with disabilities may need to choose the language they are most familiar with in order to understand a presentation for which equivalent tracks are not all available in all desired languages. In addition, some users may prefer to hear the program audio in its original language while reading captions in another, fulfilling the function of subtitles or to improve foreign language comprehension. In classrooms, teachers may wish to configure the language of various multimedia elements to achieve specific educational goals.
- Consider system level natural language preferences as the user’s default language preference. However, do not send HTTP Accept-Language request headers ([RFC2616] , section 14.4) based on the operating system preferences. First, there may be a privacy problem as indicated in RFC 2616, section 15.1.4 "Privacy Issues Connected to Accept Headers". Also, the operating system defines one language, while the Accept-Language request header may include many languages in different priorities. Setting

Accept-Language to be the operating system language may prevent a user from receiving content from a server that does not have a match for this particular language but does for other languages acceptable to the user.



This image shows how users select a natural language preference in the Real Player. This setting, in conjunction with language markup in the presentation, determines what content is rendered.

2.4 Allow the user to specify that text transcripts , collated text transcripts , captions , and auditory descriptions be rendered at the same time as the associated audio and visual tracks. Respect author-specified synchronization cues during rendering. [Priority 1] (Checkpoint 2.4)

Techniques:

- Captions and auditory descriptions may not make sense unless rendered synchronously with the primary content. For instance, if someone with a hearing disability is watching a video presentation and reading associated captions, the captions must be synchronized with the audio so that the individual can use any residual hearing. For auditory descriptions, it is crucial that the primary audio track and the auditory description track be synchronized to avoid having them both play at once, which would reduce the clarity of the presentation.

- The idea of "sensible time-coordination" of components in the definition of synchronize centers on the idea of simultaneity of presentation, but also encompasses strategies for handling deviations from simultaneity resulting from a variety of causes. Consider how deviations might be handled for captions for a multimedia presentation such as a movie clip. Captions consist of a text equivalent of the audio track that is synchronized with the visual track. Captions are essential for individuals who require an alternative way of accessing the meaning of audio, such as individuals who are deaf. Typically, a segment of the caption text appears visually near the video for several second while the person reads the text. As the visual track continues, a new segment of the caption text is presented. However, a problem arises if the caption text is longer than can fit in the display space. This can be particularly difficult if due to a visual disability, the font size has been enlarged, thus reducing the amount of caption text that can be presented. The user agent must respond sensibly to such problems, for example by ensuring that the user has the opportunity to navigate (e.g., scroll down or page down) through the caption segment before proceeding with the visual presentation and presenting the next segment.

Developers of user agents must determine how they will handle synchronization challenges, such as:

1. Under what circumstances will the "primary" presentation automatically pause? Some circumstances where this might occur include:
    - the segment of caption text is more than can fit on the visual display
    - the user wishes more time to read captions or the collated text transcript
    - the auditory description is of longer duration than the natural pause in the audio.
2. Once the "primary" presentation has paused, then under what circumstances will it resume (e.g., only when the user signals it to resume, or based on a predefined pause length)?
3. If the user agent allows the user to jump to a location in a presentation by clicking on a text equivalent (or some outline of it), then do all rendered equivalents jump at the same time? Will one be able to return to one’s previous location (or undo the action)?

Developers of user agents must anticipate many of the challenges that may arise in synchronization of diverse equivalents.

The term "synchronization cues" in checkpoint 2.4 refers to pieces of information that may affect synchronization, such as the size and expected duration of equivalents and their segments, the type of element and how much those elements can be sped up or slowed down (both from technological and intelligibility standpoints), user preferences, etc.

- User agents that implement SMIL 1.0 ([SMIL]) should implement the "Accessibility Features of SMIL" [SMIL-ACCESS] . In particular, SMIL user agents should allow users to configure whether they want to view captions, and this user interface switch should be bound to the 'system-captions' test attribute. Users should be able to indicate a preference for receiving available auditory descriptions, but SMIL 1.0 does not include a mechanism equivalent to 'system-captions' for auditory descriptions. The next version of SMIL is expected to include a test attribute for auditory descriptions.

  Another SMIL 1.0 test attribute, 'system-overdub-or-captions', allows users to select between subtitles and overdubs in multilingual presentations. User agents should *not* interpret a value of 'caption' for this test attribute as meaning that the user prefers accessibility captions; that is the purpose of the 'system-captions' test attribute. When subtitles and accessibility captions are both available, users who are deaf may prefer to view captions, as they generally contain information not in subtitles: information on music, sound effects, who is speaking, etc.

- User agents that play QuickTime movies should allow the user to turn on and off the different tracks embedded in the movie. Authors may use these alternate tracks to provide equivalent alternatives. The Apple QuickTime player currently provides this feature through the menu item "Enable Tracks."
- User agents that play Microsoft Windows Media Object presentations should provide support for Synchronized Accessible Media Interchange (SAMI [SAMI] , a protocol for creating and displaying captions) and should allow users to configure how captions are viewed. In addition, user agents which play Microsoft Windows Media Object presentations should enable people to turn on and off other equivalent alternatives, including auditory description and alternate visual tracks.
- For other formats, at a minimum, users must be able to turn on and off auditory descriptions and captions.

2.5 For non-text content that has no recognized text equivalent , generate a text equivalent from other author-supplied content. If the non-text content is included by URI reference, base the text equivalent on the URI reference and the content type of the Web resource . [Priority 2] (Checkpoint 2.5)

Note: Some markup languages (such as HTML 4 [HTML4] and SMIL 1.0 [SMIL] require the author to provide text equivalents for some content. When they don't, the user agent is required to repair the invalid content by generating a text equivalent. Refer also to checkpoint 2.6.

Techniques:

- When HTTP is used, HTTP headers provide information about the URI of the Web resource ("Content-Location") and its type ("Content-Type"). Refer to the HTTP 1.1 specification [RFC2616] , sections 14.14 and 14.17, respectively. Refer to the HTTP 1.1 specification [RFC2616] , section 3.2.1 for information about URI references.

- Text equivalents may come from markup, inside images (e.g., refer to "Describing and retrieving photos using RDF and HTTP" [PHOTO-RDF]), etc. User agents are expected to recognize equivalents by specification Refer to techniques for missing equivalent alternatives of content .

2.6 When the author has specified an empty text equivalent for non-text content , do not generate one. [Priority 3] (Checkpoint 2.6)

Note: Authors may provide an empty text equivalent (e.g., `alt=""`) when one is required by specification, but the non-text content has no other function than pure decoration. Please refer to the Web Content Accessibility Guidelines 1.0 [WCAG10] for more details. Refer also to checkpoint 2.5.

Techniques:

- User agents should render nothing in this case because the author may specify a null text equivalent for content that has no function in the page other than as decoration. In this case, the user agent should not render generic labels such as "[INLINE]" or "[GRAPHIC]".
- Allow the user to toggle the rendering of null text equivalents: between nothing and an indicator of a null equivalent (e.g., an icon with the text equivalent "EMPTY TEXT EQUIVALENT").

2.7 For author-identified but unsupported natural languages , allow the user to configure the user agent to identify those language changes in content . [Priority 3] (Checkpoint 2.7)

Techniques:

- A user agent should treat the natural language of content as part of context information. When the language changes, the user agent should either render the content in the supported natural language or notify the user of the language change, if configured for notification. For instance, a user agent that doesn’t support Korean might allow configuration to announce the language change with the message "Korean text -- unable to read". The user should also be able to choose no notification of language changes. Rendering could involve speaking in the designated natural language in the case of a voice browser or screen reader. If the natural language is not supported, the language change notification could be spoken in the default language by a screen reader or voice browser.
- Switching natural languages for blocks of content may be more helpful than switching for short phrases. In some language combinations (e.g., Japanese being the primary and English being the secondary or quoted language), short foreign language phrases are often well-integrated in the primary language. Dynamic switching for these short phrases may make the content sound unnatural and possibly harder to understand. User

agents might allow users to choose elements for which they want notification.
- Refer to techniques for generated content , which may be used to insert text to indicate a language change.
- Refer to techniques for synthesized speech .
- Refer to checkpoint 2.7 and checkpoint 5.4.
- For information on language codes, refer to "Codes for the representation of names of languages" [ISO639] .
- If users do not want to see or hear blocks of content in another natural language, allow the user to suggest hiding that content (e.g., with style sheets).
- Refer to "Character Model for the World Wide Web" [CHARMOD] . It contains basic definitions and models, specifications to be used by other specifications or directly by implementations, and explanatory material. In particular, this document addresses early uniform normalization, string identity matching, string indexing, and conventions for URIs.
- Implement content negotiation so that users may specify language preferences. Or allow the user to choose a Web resource when several are available in different languages.
- Use an appropriate glyph set when rendering visually and an appropriate voice set when rendering as speech.
- Render characters with the appropriate directionality. Refer to the "dir" attribute and the BDO element in HTML 4.01 ([HTML4] , sections 8.2 and 8.2.4 respectively). Refer also to the Unicode standard [UNICODE] .
- A user agent may not be able to render all characters in a document meaningfully, for instance, because the user agent lacks a suitable font, a character has a value that may not be expressed in the user agent's internal character encoding, etc. In this case, section 5.4 of HTML 4.01 [HTML4] recommends the following for undisplayable characters:
    1. Adopt a clearly visible (or audible), but unobtrusive mechanism to alert the user of missing resources.
    2. If missing characters are presented using their numeric representation, use the hexadecimal (not decimal) form since this is the form used in character set standards.

Guideline 3. Allow the user to configure the user agent not to render some content that may reduce accessibility.

In addition to the techniques below, refer also to the section on user control of style .

Checkpoints for content accessibility:

3.1 Allow the user to configure the user agent not to render background images. [Priority 1] (Checkpoint 3.1)

Note: Background images may make it difficult or impossible to read

superimposed text. When background images are not rendered, user agents should render a solid background color. User agent may choose not to retrieve the background image resource at all. Refer also to checkpoint 4.3 and checkpoint 4.4.
Note: This checkpoint only requires control of background for "two-layered renderings", i.e., one rendered background (other than a solid color) with all other content rendered "above it".

Techniques:

- Allow the user to turn off embedded or background images through the user agent user interface . Note that any equivalent alternatives for those images must still be available.
- In CSS, background images may be turned on/off with the 'background' and 'background-image' properties ([CSS2] , section 14.2.1).
- This checkpoint does not address issues of multi-layered renderings and does not require the user agent to change background rendering for multi-layer renderings (refer, for example, to the 'z-index' property in Cascading Style Sheets, level 2 ([CSS2] , section 9.9.1).

3.2 Allow the user to configure the user agent not to render audio. [Priority 1] (Checkpoint 3.2)
Note: Audio may interfere with other sources of sound such as the output of a speech synthesizer. User agents may play the audio "silently". They may choose not to retrieve the resource at all. Refer also checkpoint 4.6, checkpoint 4.8 and checkpoint 4.9.

Techniques:

- None yet.

3.3 Allow the user to configure the user agent not to render video. [Priority 1] (Checkpoint 3.3)
Note: The user agent should render a still image or an icon to indicate that video has not been rendered. User agent may choose not to retrieve the resource at all.

Techniques:

- Render a still image in its place.
- Implement the 'visibility' property of CSS 2 ([CSS2] , section 11.2). A value of 'hidden' will cause a blank place-holder box to be rendered in place of the video on the screen while retaining the layout of the page. This differs from a value of 'none' for the 'display' property ([CSS2] , section 9.2.5), which suppresses rendering of the video completely and may cause the layout of the page to be changed.
- Allow the user to hide a video presentation from view, even though it continues to play in the background.

3.4 Allow the user to configure the user agent to render animated or blinking text as motionless text. [Priority 1] (Checkpoint 3.4)

Techniques:

- Allow the user to turn off animated or blinking text through the user agent user interface (e.g., by pressing the **`Escape`** key to stop animations). Render static text in place of blinking text.
- Some sources of blinking an moving text are:
  - The BLINK element in HTML. **Note:** The BLINK element is not defined by a W3C specification.
  - The MARQUEE element in HTML. **Note:** The MARQUEE element is not defined by a W3C specification.
  - The 'blink' value of the 'text-decoration' property in CSS ([CSS2] , section 16.3.1).

3.5 Allow the user to configure the user agent to render animated or blinking images as motionless images. [Priority 1] (Checkpoint 3.5)
Note: Refer also to checkpoint 4.5 and checkpoint 4.6.

Techniques:

- Allow the user to turn off animated or blinking text through the user agent user interface (e.g., by pressing the **`Escape`** key to stop animations). Render a still image in its place.

3.6 Allow the user to configure the user agent not to execute scripts and applets. [Priority 1] (Checkpoint 3.6)
Note: This is particularly important for scripts that cause the screen to flicker, since people with photosensitive epilepsy can have seizures triggered by flickering or flashing, particularly in the 4 to 59 flashes per second (Hertz) range.

Techniques:

- Peak sensitivity to flickering or flashing occurs at 20 Hertz.
- Refer to the section on script techniques

3.7 Allow configuration so that author-specified "client-side redirects" (i.e., those initiated by the user agent, not the server) do not change content automatically. Allow the user to access the new content manually (e.g., by following a link). [Priority 2] (Checkpoint 3.7)

Techniques:

- Refer to the HTTP 1.1 specification [RFC2616] for information about redirection mechanisms.

3.8 Allow configuration so that author-specified content refreshes do not change content automatically. Allow the user to access the new content manually (e.g., by activating a button or following a link). Advise the user to refresh content according to the same schedule as the automatic refresh, and indicate when the user has not yet refreshed content. [Priority 2] (Checkpoint 3.8)

Techniques:

- Alert the users to pages that refresh automatically and allow them to specify a refresh rate through the user agent user interface .
- Allow the user to slow content refresh to once per 10 minutes.
- Allow the user to stop automatic refresh, but indicate that content needs refreshing and allow the user to refresh the content by activating a button or link. Or, prompt the user and ask whether to continue with forwards.
- Some HTML authors create a refresh effect by using a META element with http-equiv="refresh" and the refresh rate specified in seconds by the "content" attribute.

3.9 Allow the user to configure the user agent not to render images. [Priority 2] (Checkpoint 3.9)

Techniques:

- Provide a simple command that allows users through the user agent user interface to turn on/off the rendering of images on a page. When images are turned off, render any associated equivalents.
- Refer to techniques for checkpoint 3.1.

Guideline 4. Ensure user control of styles.

In addition to the techniques below, refer also to the section on user control of style .

Checkpoints for fonts and colors:

4.1 Allow the user to configure and control the size of text. Make available the range of system font sizes. [Priority 1] (Checkpoint 4.1)

For example, allow the user to specify a font size directly through the user agent user interface or in a user style sheet . Or, allow the user to zoom or magnify content.

Techniques:

- The choice of optimal techniques depends in part on which markup language is being used. For instance, HTML user agents may allow the user to change the font size of a particular piece of text (e.g., by using CSS user style sheets) independent of other content (e.g., images). Since the user agent can reflow the text after resizing the font, the text will become more legible without, for example, distorting bitmap images. On the other hand, some languages, such as SVG, do not allow text reflow, which means that changes to font size may cause text to overlap with other content, reducing accessibility. SVG is designed to scale, making a zoom functionality the more natural technique for SVG user agents satisfying this checkpoint.

- Inherit text size information from user preferences specified for the operating system.
- Use operating system magnification features.
- Implement the 'font-size' property in CSS ([CSS2] , section 15.2.4).
- Allow the user to override author-specified font sizes.
- When scaling text, maintain size relationships among text of different sizes.
- Allow users to configure link text to be rendered so that users with physical disabilities using a mouse may easily activate links. This may be done through style sheets, for example.

4.2 Allow the user to configure font family. Allow the user to select from among the range of system font families. [Priority 1] (Checkpoint 4.2)

Techniques:

- Inherit font family information from user preferences specified for the operating system.
- Implement the 'font-family' property in CSS ([CSS2] , section 15.2.2).
- Allow the user to override author-specified font families.

4.3 Allow the user to configure foreground color. Allow the user to select from among the range of system colors. [Priority 1] (Checkpoint 4.3)

Techniques:

- Inherit foreground color information from user preferences specified for the operating system.
- Implement the 'color' and 'border-color' properties in CSS 2 ([CSS2] , sections 14.1 and 8.5.2, respectively).
- Allow the user to specify minimal contrast between foreground and background colors, adjusting colors dynamically to meet those requirements.
- Allow the user to override author-specified foreground colors.

4.4 Allow the user to configure background color. Allow the user to select from among the range of system colors. [Priority 1] (Checkpoint 4.4)

Techniques:

- Inherit background color information from user preferences specified for the operating system.
- Implement the 'background-color' property (and other background properties) in CSS 2 ([CSS2] , section 14.2.1).

- Allow the user to override author-specified background colors.

Checkpoints for multimedia presentations , audio-only presentations , and visual-only presentations :

4.5 Allow the user to slow the presentation rate of audio, video, and animations. For a visual track, provide at least one setting between 40% and 60% of the original speed. For a pre-recorded audio track including audio-only presentations , provide at least one setting between 75% - 80% of the original speed. For a synchronized multimedia presentation where the visual track may be slowed from 100% to to 80% of its original speed, synchronize the visual and audio tracks. Below 80%, the user agent is not required to render the audio track . [Priority 1] (Checkpoint 4.5)

Refer also to checkpoint 2.4.

Techniques:

- Allowing the user to slow the presentation of video, animations, and audio will benefit individuals with specific learning disabilities, cognitive disabilities, or individuals with newly acquired sensory limitations (such as a person who is newly blind and learning to use a screen reader). The same feature will benefit individuals who have beginning familiarity with a natural language .
- When changing the rate of audio, avoid pitch distortion.
- Some formats do not allow changes in playback rate.

4.6 Allow the user to start, stop, pause, resume, advance, and rewind to the beginning audio, video, and animations. [Priority 1] (Checkpoint 4.6)

Techniques:

- Allow the user to advance or rewind the presentation in increments. This is particularly valuable to users with physical disabilities who may not have fine control over advance and rewind functionalities. Allow users to configure the size of the increments.
- If buttons are used to control advance and rewind, make the advance/rewind distances proportional to the time the user activates the button. After a certain delay, accelerate the advance/rewind.
- Apply techniques for changing audio speed without introducing distortion.
- Note that Home Page Reader [HPR] lets users insert bookmarks in presentations.

4.7 Allow the user to position text transcripts , collated text transcripts , and captions on graphical displays. The range of available positions must be the same range available to the author according to specification. [Priority 1] (Checkpoint 4.7)

Techniques:

- Some users need to be able to position captions, etc. so that they do not obscure other content or are not obscured by other content. Other users (e.g., users with screen magnifiers or who have other visual disabilities) require pieces of content to be in a particular relation to one another, even if this means that some content will obscure other content.
- User agents should implement the positioning features of the employed markup or style sheet language. Even when a markup language does not explicitly allow positioning, when a user agent can recognize distinct text transcripts , collated text transcripts , or captions , the user agent should allow the user to reposition them. User agents are not required to allow repositioning when the captions, etc. cannot be separated from other media (e.g., the captions are part of the video track).
- Implement the CSS 2 'position' property ([CSS2] , section 9.3.1).
- Allow the user to choose whether captions appear at the bottom or top of the video area or in other positions. Currently authors may place captions overlying the video or in a separate box. Captions prevent users from being able to view other information in the video or on other parts of the screen, making it necessary to move the captions in order to view all content at once. In addition, some users will find captions easier to read if they can place them in a location best suited to their reading style.
- Allow users to configure a general preference for caption position and to be able to fine tune specific cases. For example, the user may want the captions to be in front of and below the rest of the presentation.
- Allow the user to drag and drop the captions to a place on the screen. To ensure device-independence, allow the user to enter the screen coordinates of one corner of the caption.
- Allow the user to position all parts of a presentation rather than trying to identify captions specifically (i.e., solving the problem generally may be easier than for captions alone).
- Do not require users to edit the source code of the presentation to achieve the desired effect.

Checkpoints for audio volume control:

4.8 Allow the user to configure and control the global audio volume. The user must be able to choose zero volume (i.e., silent). [Priority 1] (Checkpoint 4.8)

Note: User agents should allow global control of volume through available system-level controls.

Techniques:

- Use audio control mechanisms provided by the operating system. Control of volume mix is particularly important, and the user agent should provide easy access to those mechanisms provided by the operating system.
- Implement the CSS 2 'volume' property ([CSS2] , section 19.2).

- Allow the user to configure a volume level at the operating system level.
- Implement the 'display', 'play-during', and 'speak' properties in CSS 2 ([CSS2] , sections 9.2.5, 19.6, and 19.5, respectively).
- Authors sometimes specify background sounds with the "bgsound" attribute. **Note:** This attribute is **not** part of HTML 4.01 [HTML4] .

4.9 Allow the user to control independently the volumes of distinct audio sources synchronized to play simultaneously. [Priority 1] (Checkpoint 4.9)

Note: Refer also to checkpoint 4.11.

Techniques:

None.

Checkpoints for synthesized speech:

Refer also to techniques for synthesized speech .

4.10 Allow the user to configure and control synthesized speech playback rate according to the full range offered by the speech synthesizer. The lower bound for this range must be at most 120 words per minute. The upper bound for this range must be at least 400 words per minute. The user must be able to increase or decrease the playback rate in increments of 5% of the current playback rate. [Priority 1] (Checkpoint 4.10)

Techniques:

- Use synthesized speech mechanisms provided by the operating system.
- Implement the CSS 2 'speech-rate' property ([CSS2] , section 19.8).

4.11 Allow the user to control the synthesized speech volume independently of other sources of audio. [Priority 1] (Checkpoint 4.11)

Note: Refer also to checkpoint 4.9.

Techniques:

- The user agent should allow the user to make synthesized speech louder and softer than other audio sources.
- Use synthesized speech mechanisms provided by the operating system.
- Implement the CSS 2 'volume' property ([CSS2] , section 19.2).

4.12 Allow the user to configure synthesized voice gender, pitch, pitch range, stress, and richness according to the full range of values offered by the speech synthesizer. [Priority 2] (Checkpoint 4.12)

Note: This list of voice characteristic properties is based on the list in section 19.8 of Cascading Style Sheets Level 2 [CSS2] . Ranges of values for these properties may vary among speech synthesizers.

Techniques:

- Use synthesized speech mechanisms provided by the operating system.
- Implement the voice characteristic properties of CSS 2: 'voice-family', 'pitch', 'pitch-range', 'stress', 'richness', ([CSS2] , sections 19.8 and 19.9).
- One example of a speech API is Microsoft's Speech Application Programming Interface [SAPI] .

Checkpoints for user interface accessibility:

4.13 Allow the user to select from available author and user style sheets or to ignore them. [Priority 1] (Checkpoint 4.13)

Note: By definition, the user agent's default style sheet is always present, but may be overridden by author or user styles.

Techniques:

- For HTML [HTML4] , make available "class" and "id" information so that users can override styles.
- Implement user style sheets .
- Implement the "!important" semantics of CSS 2 ([CSS2] , section 6.4.2).

4.14 Allow the user to configure how the selection is highlighted (e.g., foreground and background color). Offer at least three rendering options, including colors and fonts. Allow the user to select from among the range of system colors and fonts. [Priority 1] (Checkpoint 4.14)

Techniques:

- As an sample implementation, note that Netscape Navigator [NAVIGATOR] for X Windows uses resources to control the selection colors (`*selectForeground` and `*selectBackground`).
- Implement the CSS 2 "HighLightText and "Highlight" predefined color values ([CSS2] , section 18.2).
- Inherit selection information from user's settings for the operating system.

4.15 Allow the user to configure how the content focus is highlighted (e.g., foreground and background color). Offer at least three rendering options, including colors and fonts. Allow the user to select from among the range of system colors and fonts. The default focus highlight mechanism must be different from the default selection highlight mechanism. [Priority 1] (Checkpoint 4.15)

Techniques:

- Implement the CSS 2 ':focus' pseudo-class and dynamic outlines and focus of CSS 2 ([CSS2] , sections 5.11.3 and 18.4.1, respectively).

For example, the following rule will cause links with focus to appear with a blue background and yellow text.

```
A:focus { background: blue; color: yellow }
```

The following rule will cause TEXTAREA elements with focus to appear with a particular focus outline:

```
TEXTAREA:focus { outline: thick black solid }
```

- Inherit focus information from user’s settings for the operating system.
- Test the user agent to ensure that individuals who have low vision and use screen magnification software are able to follow highlighted item(s).

4.16 Allow the user to configure whether the current focus moves automatically to a viewport that opens without an explicit request from the user. [Priority 2] (Checkpoint 4.16)

Techniques:

- Allow the user to configure how current focus changes when a new viewport opens. For instance, the user might choose between these two options:
  1. Do not change the focus when a viewport opens, but notify the user (e.g., with a beep, flash, and text message on the status bar). Allow the user to navigate directly to the new window upon demand.
  2. Change the focus when a window opens and use a subtle alert (e.g., a beep, flash, and text message on the status bar) to indicate that the focus has changed.
- If a new viewport or prompt appears but focus does not move to it, notify assistive technologies (per checkpoint 5.5) so that they may discreetly inform the user.
- When a viewport is duplicated, the focus in the new viewport should initially be the same as the focus in the original viewport. Duplicate viewports allow users to navigate content (e.g., in search of some information) in one viewport while allowing the user to quickly return to the point of regard in the duplicate viewport. There are other techniques for accomplishing this (e.g., "registers" in emacs).
- For user agents that implement CSS 2 [CSS2] , the following rule will generate a message to the user at the beginning of link text for links that are meant to open new windows when followed:

```
A[target=_blank]:before{content:"Open new window"}
```

4.17 Allow the user to configure the user agent so that after one viewport is open, no other viewports open except as the result of explicit user request. [Priority 2] (Checkpoint 4.17)

Note: Some users may become disoriented when there are too many open viewports. In addition to configuration, user agents should allow the user to control the number of open viewports by selecting and closing them. Following an author-specified link that opens a new viewport does not constitute an explicit request from the user. Refer also to checkpoint 4.16, checkpoint 5.5, and checkpoint 9.3.

Techniques:

- For HTML [HTML4] , allow the user to control the process of opening a document in a new "target" frame or a viewport created by a script. For example, for `target="_blank"`, open the window according to the user's preference.
- For SMIL [SMIL] , allow the user to control viewports created with the "new" value of the "show" attribute.

Guideline 5. Observe system conventions and standard interfaces.

Checkpoints for communication with other software:

5.1 Provide programmatic read access to HTML and XML content by conforming to the W3C Document Object Model (DOM) Level 2 Core and HTML modules and exporting the interfaces they define. [Priority 1] (Checkpoint 5.1)

Note: These modules are defined in DOM Level 2 [DOM2] , chapters 1 and 2. Please refer to that specification for information about which versions of HTML and XML are supported and for the definition of a "read-only" DOM. This checkpoint is an important special case of checkpoint 2.1. For content other than HTML and XML, refer to checkpoint 5.3.

Techniques:

- Note that the W3C DOM is designed to be used on a server as well as a client and does not address some user interface-specific information.
- Refer to the appendix on loading assistive technologies for DOM access .
- For information about rapid access to Internet Explorer's [IE-WIN] DOM through COM, refer to [BHO] .
- Refer to the Java Weblets implementation of the DOM [JAVAWEBLET] .

5.2 If the user can modify HTML and XML content through the user interface , provide the same functionality programmatically by conforming to the W3C Document Object Model (DOM) Level 2 Core and HTML modules and exporting the interfaces they define. [Priority 1] (Checkpoint 5.2)

Note: For example, if the user interface allows users to complete HTML forms, this must also be possible through the DOM APIs . These modules are defined in DOM Level 2 [DOM2] , chapters 1 and 2. Please refer to DOM Level 2 [DOM2] for information about which versions of HTML and XML are supported. This checkpoint is an important special case of checkpoint 2.1. For markup languages other than HTML and XML, refer to checkpoint 5.3.

Techniques:

- Refer to techniques for checkpoint 5.1.

5.3 For markup languages other than HTML and XML, provide programmatic access to content using standard APIs (e.g., platform-independent APIs and standard APIs for the operating system). [Priority 1] (Checkpoint 5.3)

Note: This checkpoint addresses content not covered by checkpoints checkpoint 5.1 and checkpoint 5.2. This checkpoint is an important special case of checkpoint 2.1.

Techniques:

- Refer to techniques for checkpoint 5.4.

5.4 Provide programmatic read and write access to user agent user interface controls using standard APIs (e.g., platform-independent APIs such as the W3C DOM, standard APIs for the operating system, and conventions for programming languages, plug-ins, virtual machine environments, etc.) [Priority 1] (Checkpoint 5.4)

Note: For example, provide access to information about the user agent's current input configuration so that assistive technologies can trigger functionalities through keyboard events, mouse events, etc.

Techniques:

- Use standard operating system and programming language APIs that support accessibility by providing a bridge between the standard user interface supported by the operating system and alternative user interfaces developed by assistive technologies . User agents that implement these APIs are generally more compatible with assistive technologies and provide accessibility at no extra cost. Some public APIs that promote accessibility include:
  - Microsoft Active Accessibility ([MSAA]) in Windows 95/98/NT versions.
  - Sun Microsystems Java Accessibility API ([JAVAAPI]) in Java JDK. If the user agent supports Java applets and provides a Java Virtual Machine to run them, the user agent should support the proper loading and operation of a Java native assistive technology. This assistive technology can provide access to the applet as defined by Java accessibility standards.
- Use standard user interface controls. 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. Ensure that they provide accessibility information through an API as is done for the standard controls.
- Make use of operating system level features. See the appendix of

accessibility features for some common operating systems.
- Provide information about the selection and focus .
- Inherit operating system settings related to accessibility (e.g., for fonts, colors, natural language preferences, input configurations, etc.).
- Write output to and take input from standard system APIs rather than directly from hardware controls. This will enable the I/O to be redirected from or to assistive technology devices – for example, screen readers and Braille displays 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 that allows for interoperability of the assistive technology with a range of applications.
- For information about rapid access to Internet Explorer's [IE-WIN] DOM through COM, refer to [BHO] .

5.5 Using standard APIs , provide programmatic notification of changes to content and user interface controls (including selection , content focus , and user interface focus). [Priority 1] (Checkpoint 5.5)

Note: Use the standard APIs required by guideline 5.

Techniques:

- Refer to "mutation events" in DOM Level 2 [DOM2] . This module of DOM 2 allows assistive technologies to be informed of changes to the document tree.
- Refer also to information about monitoring HTML events through the document object model in Internet Explorer [IE-WIN] .

5.6 Ensure that programmatic exchanges proceed in a timely manner. [Priority 2] (Checkpoint 5.6)

Note: For example, the programmatic exchange of information required by other checkpoints in this document must be efficient enough to prevent information loss, a risk when changes to content or user interface occur more quickly than the communication of those changes. The techniques for this checkpoint explain how developers can reduce communication delays, e.g., to ensure that assistive technologies have timely access to the document object model and other information needed for accessibility.

Techniques:

- Please refer to the appendix that explains how to load assistive technologies for DOM access .
- Alert the user when information is being lost due to communication delays.

5.7 Provide programmatic access to Cascading Style Sheets (CSS) by conforming to the W3C Document Object Model (DOM) Level 2 CSS module and exporting the interfaces it defines. [Priority 3] (Checkpoint 5.7)

Note: This module is defined in DOM Level 2 [DOM2] , chapter 5. Please refer to that specification for information about which versions of CSS are supported. This checkpoint is an important special case of checkpoint 2.1.

Techniques:

- Refer to techniques for checkpoint 5.1.

Checkpoints for user interface accessibility:

5.8 Follow operating system conventions that benefit accessibility. In particular, follow conventions for user interface design, keyboard configuration, product installation, and documentation . [Priority 2] (Checkpoint 5.8)

Note: Operating system conventions that benefit accessibility are those described in this document and in platform-specific accessibility guidelines. Some of these conventions (e.g., sticky keys, mouse keys, show sounds, etc.) are discussed in the Techniques document [UAAG10-TECHS] . Refer also to checkpoint 10.2.

Techniques:

- Refer to techniques for checkpoint 1.2.
- Refer to techniques for checkpoint 5.4.
- Refer to techniques for checkpoint 10.2.
- Follow operating system and application environment (e.g., Java) conventions for loading assistive technologies. Refer to the appendix on loading assistive technologies for DOM access for information about how an assistive technology developer can load its software into a Java Virtual Machine.
- Ensure that any online services (e.g., automated update facilities, download-and-install functionalities, sniff-and-fill forms, etc.) observe relevant operating system conventions concerning device independence and accessibility (as well as the Web Content Accessibility Guidelines 1.0 [WCAG10]).
- Evaluate the standard interface controls on the target platform against any built-in operating system accessibility functions (refer to the appendix on accessibility features of some operating systems). Ensure that the user agent operates properly with all these functions. Here is a sample of features to consider:
  - Microsoft Windows offers an accessibility function called "High Contrast". Standard window classes and controls automatically support this setting. However, applications created with custom classes or controls work with the "GetSysColor" API to ensure compatibility with High Contrast.
  - Apple Macintosh offers an accessibility function called "Sticky Keys". Sticky Keys operate with keys the operating system recognizes as modifier keys, and therefore a custom control should not attempt to define a new modifier key.

  - Maintain consistency in the user interface between versions of the software. Consistency is less important than improved general accessibility and usability, but developers should make changes conservatively to the layout of user interface controls , the behavior of existing functionalities, and the default keyboard configuration.
- Follow accessibility guidelines for specific platforms:
  - "Macintosh Human Interface Guidelines" [APPLE-HI]
  - "IBM Guidelines for Writing Accessible Applications Using 100% Pure Java" [JAVA-ACCESS] .
  - "An Inter-client Exchange (ICE) Rendezvous Mechanism for X Window System Clients" [ICE-RAP] .
  - "Information for Developers About Microsoft Active Accessibility" [MSAA] .
  - "The Inter-Client communication conventions manual" [ICCCM] .
  - "Lotus Notes accessibility guidelines" [NOTES-ACCESS] .
  - "Java accessibility guidelines and checklist" [JAVA-CHECKLIST] .
  - "The Java Tutorial. Trail: Creating a GUI with JFC/Swing" [JAVA-TUT] .
  - "The Microsoft Windows Guidelines for Accessible Software Design" [MS-SOFTWARE] .
- Follow general guidelines for producing accessible software:
  - "Accessibility for applications designers" [MS-ENABLE] .
  - "Application Software Design Guidelines" [TRACE-REF] .
  - "Designing for Accessibility" [SUN-DESIGN] .
  - "EITAAC Desktop Software standards" [EITAAC] .
  - "Requirements for Accessible Software Design" [ED-DEPT] .
  - "Software Accessibility" [IBM-ACCESS] .
  - Towards Accessible Human-Computer Interaction" [SUN-HCI] .
  - "What is Accessible Software" [WHAT-IS] .
  - Accessibility guidelines for Unix and X Window applications [XGUIDELINES] .

Guideline 6. Implement accessible specifications.

Checkpoints for content accessibility:

6.1 Implement the accessibility features of all supported specifications (markup languages, style sheet languages, metadata languages, graphics formats, etc.). Accessibility features are those identified in the specification and those features of the specification that support requirements of the "Web Content Accessibility Guidelines 1.0" [WCAG10] , the "Authoring Tool Accessibility Guidelines 1.0" [ATAG10] , and the current document. [Priority 1] (Checkpoint 6.1)

Techniques:

- Make obvious to users features that are known to promote accessibility. Make them easy to find in the user interface and in documentation.
- Refer to the "Accessibility Features of CSS" [CSS-ACCESS] . Note that CSS 2 includes properties for configuring synthesized speech styles.
- Refer to the "Accessibility Features of SMIL" [SMIL-ACCESS] .
- Refer to the following list of accessibility features of HTML 4.01 [HTML4] (in addition to those described in techniques for checkpoint 2.1):
  - The CAPTION element (section 11.2.2) for rich table captions.
  - Table elements THEAD, TBODY, and TFOOT (section 11.2.3), COLGROUP and COL (section 11.2.4) that group table rows and columns into meaningful sections.
  - Attributes "scope", "headers", and "axis" (section 11.2.6) which are semantically significant labels that non-graphical user agents may use to render a table in a linear fashion.
  - The "tabindex" attribute (section 17.11.1) for assigning the order of keyboard navigation within a document.
  - The "accesskey" attribute (section 17.11.2) for assigning keyboard commands to active components such as links and form controls.
- For information about the Sun Microsystems Java Accessibility API in Java JDK, refer to [JAVAAPI] .
- For information about captioning for the Synchronized Accessible Multimedia Interchange (SAMI), refer to [SAMI] .

6.2 Use and conform to W3C Recommendations when they are available and appropriate for a task. [Priority 2] (Checkpoint 6.2)

Note: For instance, for markup, implement HTML 4.01 [HTML4] , XHTML 1.0 [XHTML10] , or XML 1.0 [XML] . For style sheets, implement CSS ([CSS1] , [CSS2]). For mathematics, implement MathML [MATHML] . For synchronized multimedia, implement SMIL 1.0 [SMIL] . For information about programmatic access to HTML and XML content, refer to guideline 5.

Note: For reasons of backward compatibility, user agents should continue to implement deprecated features of specifications. The current guidelines refer to some deprecated language features that do not necessarily promote accessibility but are widely deployed. Information about deprecated language features is generally part of the language’s specification.

Techniques:

- The requirement of this checkpoint is to implement *at least* one W3C Recommendation that is available and appropriate for a particular task. For example, user agents would satisfy this checkpoint by implementing the Portable Network Graphics 1.0 specification [PNG] for raster images. In addition, user agents may implement other image formats such as JPEG, GIF, etc.
- If more than one version or level of a W3C Recommendation is appropriate for a particular task, user agents are encouraged to implement the latest

version.

- Specifications that become W3C Recommendations after a user agent's development cycles permit input are not considered "available" in time for that version of the user agent.
- Each specification defines what conformance means for that specification.
- W3C make available validation services to promote the proper usage and implementation of specifications. Refer to the:
  - HTML and XML validator service [VALIDATOR] .
  - CSS validator service [CSSVALIDATOR] .

Guideline 7. Provide navigation mechanisms.

Checkpoints for user interface accessibility:

7.1 Allow the user to navigate among all viewports (including frames). [Priority 1] (Checkpoint 7.1)

Note: For example, when all frames of a frameset are displayed side-by-side, allow the user to navigate among them with the keyboard. Or, when frames are accessed or viewed one at a time (e.g., by a text browser or speech synthesizer), provide a list of links to other frames. Navigation among all viewports implies at least allowing the user to cycle through all viewports. Navigating into a viewport makes it the current viewport .

Techniques:

- Refer to the frame techniques . Some operating systems provide a means to navigate among all open windows using multiple input devices (e.g., keyboard and mouse). This technique would suffice for switching among user agent viewports that are separate windows. However, user agents may also provide a mechanism to shift the user interface focus among user agent windows, independent of the standard operating system mechanism.

7.2 Associate a point of regard with each state in a viewport's browsing history and when the user returns to a state in the history, restore the associated point of regard. [Priority 1] (Checkpoint 7.2)

Note: For example, when the user navigates from one viewport to another (per checkpoint 7.1) and back, the point of regard should be restored.

Techniques:

- When the user returns to a page after following a link, restore content focus to that link.
- If the user agent allows the user to browse multimedia or audio-only presentations , when the user leaves one presentation for another, pause the presentation. When the user returns to a previous presentation, allow the user to resume the presentation where it was paused (i.e., return the point of regard to the same place in space and time). **Note:** This may be

done for a presentation that is available "completely" but not for a "live" stream or any part of a presentation that continues to run in the background. Allow the user to configure whether leaving a viewport pauses a multimedia presentation.
- Refer to the HTTP 1.1 specification for information about history mechanisms ([RFC2616] , section 13.13).

7.3 Allow the user to navigate all active elements . If the author has not specified a navigation order, allow at least forward sequential navigation of elements, in document order. [Priority 1] (Checkpoint 7.3)

Note: Navigation may include non-active elements in addition to active elements. This checkpoint is an important special case of checkpoint 7.6.

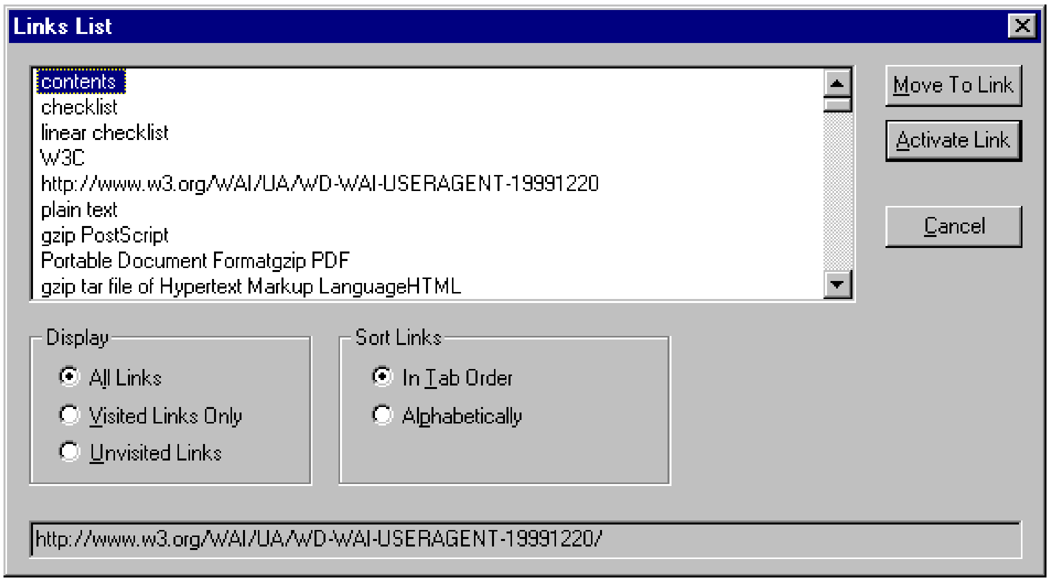
Techniques:

Sequential navigation techniques

- Allow the user to navigate sequentially all active elements by repeatedly pressing a single key. Many user agents today allow users to navigate sequentially by repeating a key combination -- for example, using the **`Tab`** key for forward navigation and **`Shift-Tab`** for reverse navigation. Because the **`Tab`** key is typically on one side of the keyboard while arrow keys are located on the other, users should be allowed to configure the user agent so that sequential navigation is possible with keys that are physically closer to the arrow keys. Refer also to checkpoint 10.4.
- Provide other sequential navigation mechanisms for particular element types or semantic units, e.g., "Find the next table" or "Find the previous form." For more information about sequential navigation of form controls and form submission, refer to techniques for checkpoint 9.2.
- Maintain a logical element navigation order. For instance, users may use the keyboard to navigate among elements or element groups using the arrow keys within a group of elements. One example of a group of elements is a set of radio buttons. Users should be able to navigate to the group of buttons, then be able to select each button in the group. Similarly, allow users to navigate from table to table, but also among the cells within a given table (up, down, left, right, etc.).
- The default sequential navigation order should respect the conventions of the natural language of the document. Thus, for most left-to-right languages, the usual navigation order is top-to-bottom and left-to-right. For right-to-left languages, the order would be top-to-bottom and right-to-left.
- Respect author-specified information about navigation order (e.g., the "tabindex" attribute in HTML 4 [HTML4] , section 17.11.1). Allow users to override the author-specified navigation order (e.g., by offering an alphabetized view of links or other orderings).
- Give the users the option of navigating to *and activating* a link, or just moving the content focus to the link. First-time users of a page may want

access to link text before deciding whether to follow the link (activate). More experienced users of a page might prefer to follow the link directly, without the intervening content focus step.

- In Java, a component is part of the sequential navigation order when added to a panel and its `isFocusTraversable` method returns true. A component can be removed from the navigation order by extending the component, overloading this method, and returning false.



This image shows how JAWS for Windows [JFW] allows users to navigate to links in a document and activate them independently. Users may also configure the user agent to navigate visited links, unvisited links, or both. Users may also change the sequential navigation order, sorting links alphabetically or leaving them in the logical tabbing order. The focus in the links view follows the focus in the primary view.

Direct navigation techniques

- Excessive use of sequential navigation can reduce the usability of software for both disabled and non-disabled users.
- Some useful types of direct navigation include: navigation based on position (e.g., all links are numbered by the user agent), navigation based on element content (e.g., the first letter of text content), direct navigation to a table cell by its row/column position, and searching (e.g., based on form control text, associated labels, or form control names).
- Document available direct navigation mechanisms.

7.4 Allow the user to choose to navigate only active elements . If the author has not specified a navigation order, allow at least forward and reverse sequential navigation of active elements, in document order. [Priority 2] (Checkpoint 7.4)

Techniques:

- Apply the techniques of checkpoint 7.3 to active elements only.

7.5 Allow the user to search for rendered text content , including rendered text equivalents . Allow the user to start a forward search from a location in content selected or focused by the user. After a match, allow searching from location of the match. Provide a case-insensitive search option when applicable to the natural language of text. [Priority 2] (Checkpoint 7.5)

Note: The default search starting point should be the beginning of content. Use operating system conventions for marking the result of a search (e.g., selection or content focus).

Techniques:

- Allow users to search for text content (element content and attribute values).
- Allow users to search forward and backward from the point of regard, the beginning of document, or the end of the document.
- Allow users to search a document source view.
- For forms, allow users to find controls that must be changed by the user before submitting the form. Allow users to search the element content of controls (where applicable) and any label text.
- Allow the user to search among just text equivalents of other content.
- 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 find all media elements active at a particular time in the presentation.
  - Allow users to view a list of all media elements or links of the presentations sorted by start or end time or alphabetically.
  - For frames, allow users to search for content in all frames, without having to be in a particular frame.
  - It may be confusing to allow users to search for text content that is *not* rendered (and thus that they have not viewed). If this type of search is possible, notify the user of this particular search mode.

7.6 Allow the user to navigate efficiently to and among important structural elements identified by the author. For markup languages with known semantics, allow forward sequential navigation to important structural elements. For other markup languages, allow at least forward sequential navigation of the document object, in document order. [Priority 2] (Checkpoint 7.6)

Note: Structured navigation of headings, tables, forms, lists, etc., is most effective when available in conjunction with a configurable view (checkpoint 8.4 and checkpoint 8.5). User agents should follow operating system conventions for indicating navigation progress (e.g., selection or content focus).

Techniques:

User agents should construct the navigation view with the goal of breaking content into sensible pieces according to the author's design. In most cases, user agents should not break down content into individual elements for navigation; element-by-element navigation of the document object does not meet the goal of facilitating navigation to important pieces of content.

Instead, user agents are expected to construct the navigation view from author-supplied markup. For those languages with known conventions for identifying important components, user agents should construct the navigation tree from those components, allowing users to navigate to them, skip them, or navigate into them. In HTML, important elements including headings, tables, forms, DIV elements (notably with a "title" attribute set), navigation mechanisms (marked up with MAP), and lists. HTML also allows authors to specify keyboard configurations ("accesskey", "tabindex"), which can serve as hints about what the author considers important. Tables and forms illustrate the utility of a recursive navigation mechanism. The user should be able to navigate to tables, then change "scope" and navigate within the cells of that table.

Nested tables (a table within the cell of another table) fit nicely within this scheme. However, the headers of an nested table may provide important context for the cells of the same row(s) or column(s) containing the nested table.

The same ideas apply to forms: users should be able to navigate to a form, then among the controls within that form.

In SVG, the "g" element signifies a grouping and should be considered when constructing the navigation view. In SMIL, "par", "seq", and "switch" provide information that may be useful for identifying significant components of content.

User agents should allow users to:

1. Navigate to a piece of content that the author has identified as important according to the markup language specification and conventional usage. In HTML, for example, this includes headings, forms, tables, navigation mechanisms, and lists.
2. Navigate past that piece of content (i.e., avoid the details of that component).
3. Navigate into that piece of content (i.e., chose to view the details of that component).
4. Change the navigation view as they go, expanding and contracting portions of content that they wish to examine or ignore. This will speed up navigation and promote orientation at the same time.

- Use the DOM [DOM2] as the basis of structured navigation (e.g., a postorder traversal). However, for well-known markup languages such as HTML, structured navigation should take advantage of the structure of the source tree and what is rendered.
- Allow navigation based on commonly understood document models, even if

they do not adhere strictly to a Document Type Definition (DTD). For instance, in HTML, although headings (H1-H6) are not containers, they may be treated as such for the purpose of navigation. Note that they should be properly nested.

- Allow the user to limit navigation to the cells of a table (notably left and right within a row and up and down within a column). Navigation techniques include keyboard navigation from cell to cell (e.g., using the arrow keys) and page up/down scrolling. Refer to the section on table navigation .
- Allow depth-first as well as breadth-first navigation.
- Alert the user when navigation has led to the beginning or end of a structure (e.g., end of a list, end of a form, table row or column end, etc.). Refer also to checkpoint 1.5.
- Provide context-sensitive navigation. For instance, when the user navigates to a list or table, provide locally useful navigation mechanisms (e.g., within a table, cell-by-cell navigation) using similar input commands.
- From a given element, allow navigation to the next or previous sibling, up to the parent, and to the end of an element.
- Allow users to navigate synchronized multimedia presentations. Refer also to checkpoint 4.6.
- Allow the user to navigate characters, words, sentences, paragraphs, screenfuls, and other pieces of text content that depend on natural language . This benefits users of speech-based user agents and has been implemented by several screen readers, including Winvision [WINVISION] , Window-Eyes [WINDOWEYES] , and JAWS for Windows [JFW] .
- Allow users to skip author-specified navigation mechanisms such as navigation bars. For instance, navigation bars at the top of each page at a Web site may force users with screen readers or some physical disabilities to wade through many links before reaching the important information on the page. User agents may facilitate browsing for these users by allowing them to skip recognized navigation bars (e.g., through a configuration option). Some techniques for this include:
    1. Providing a functionality to jump to the first non-link content.
    2. In HTML, the MAP element may be used to mark up a navigation bar (even when there is no associated image). Thus, users might ask that MAP elements not be rendered in order to hide links inside the MAP element. User agents might allow users to hide MAP elements selectively. For example, hide any MAP element with a `"title"` attribute specified. **Note:** Starting in HTML 4.01, the MAP element allows block content, not just AREA elements.
- The following is a summary of ideas provided by the National Information Standards Organization [NISO] :

    A talking book's "Navigation Control Center" (NCC) resembles a traditional table of contents, but it is more. It contains links to all headings at all levels in the book, links to all pages, and links to any items that the reader has chosen not to have read. For example, the reader may have turned off the automatic reading of footnotes. To allow the user to retrieve that information quickly, the reference to the footnote is placed in the NCC and the reader can go to the reference, understand the context for the footnote, and then read the footnote.

Once the reader is at a desired location and wishes to begin reading, the navigation process changes. Of course, the reader may elect to read sequentially, but often some navigation is required (e.g., frequently people navigate forward or backward one word or character at a time). Moving from one sentence or paragraph at a time is also needed. This type of local navigation is different from the global navigation used to get to the location of what you want to read. It is frequently desirable to move from one block element to the next. For example, moving from a paragraph to the next block element which may be a list, blockquote, or sidebar is the normally expected mechanism for local navigation.

7.7 Allow the user to configure and control the set of elements navigable according to checkpoint 7.6 by allowing inclusion and exclusion of element types in the navigation sequence. [Priority 3] (Checkpoint 7.7)

Note: For example, allow the user to navigate only paragraphs, or only headings and paragraphs, etc.

Techniques:

- Allow the user to navigate by element type.
- Allow the user to navigate HTML elements that share the same "class" attribute.
- Allow the user to expand or shrink portions of the document structure (configure detail level) for faster access to important parts of content.

Guideline 8. Orient the user.

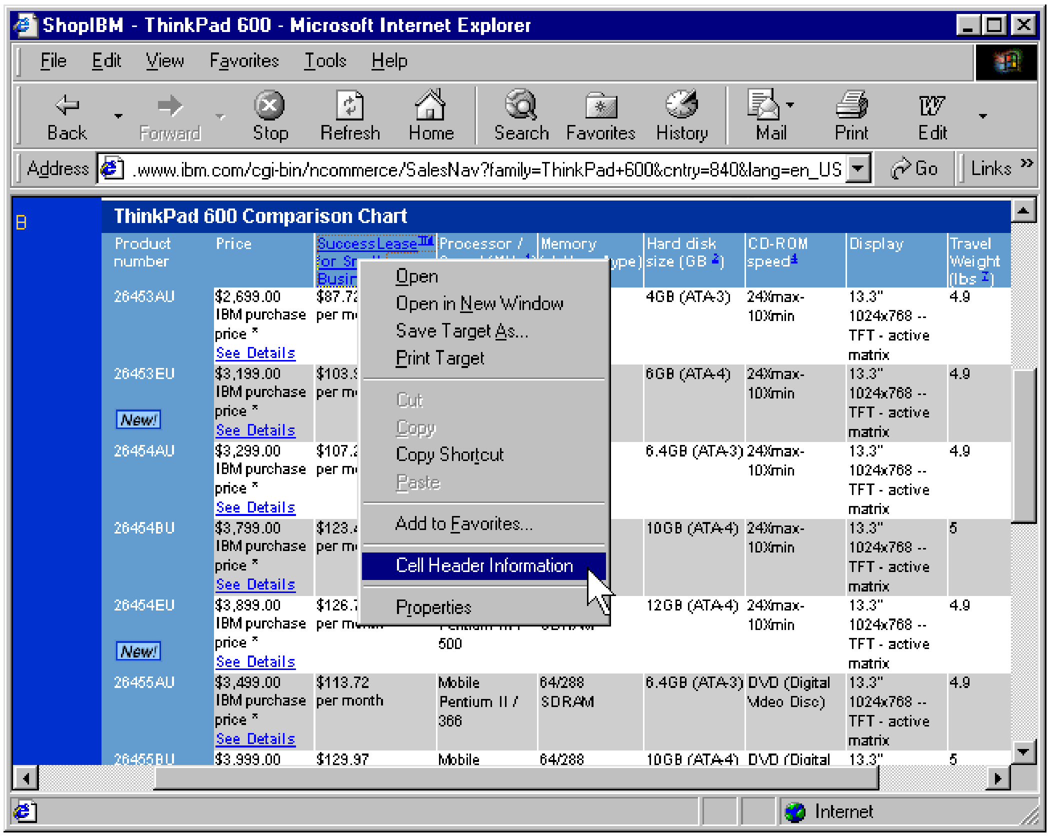
Checkpoints for content accessibility:

8.1 Make available to the user the author-specified purpose of each table and the author-specified relationships among the table cells and headers. [Priority 1] (Checkpoint 8.1)

Note: Depending on the table, some techniques may be more efficient than others for conveying data relationships. For many tables, graphical user agents may satisfy this checkpoint by rendering a table as a two dimensional grid and by ensuring that users can find headers associated with cells. However, for large tables or small viewports, allowing the user to query cells for information about related headers may improve access. Refer also to checkpoint 5.3. **Note:** This checkpoint is an important special case of checkpoint 2.1.

Techniques:

- Provide information about table headers, how headers relate to cells, table summary information, cell position information, table dimensions, etc.
- In HTML, beyond the TR, TH, and TD elements, the table attributes "summary", "abbr", "headers", "scope", and "axis" provide information about relationships among cells and headers. For more information, refer to the section on table techniques .
- Refer to the THEAD, TBODY, and TFOOT elements of HTML 4 ([HTML4] , section 11.2.3). These elements may be "fixed" to the screen (or repeated on paper) with the 'fixed' value of the CSS2 'position' property ([CSS2] , section 9.3.1). When these elements are used by authors, users can scroll through data while retaining headers and footers "in view".
- When rendering a table serially, allow the user to specify how cell header information should be rendered before cell data information. Some possibilities are illustrated by the CSS2 'speak-header' property ([CSS2] , section 17.7.1).



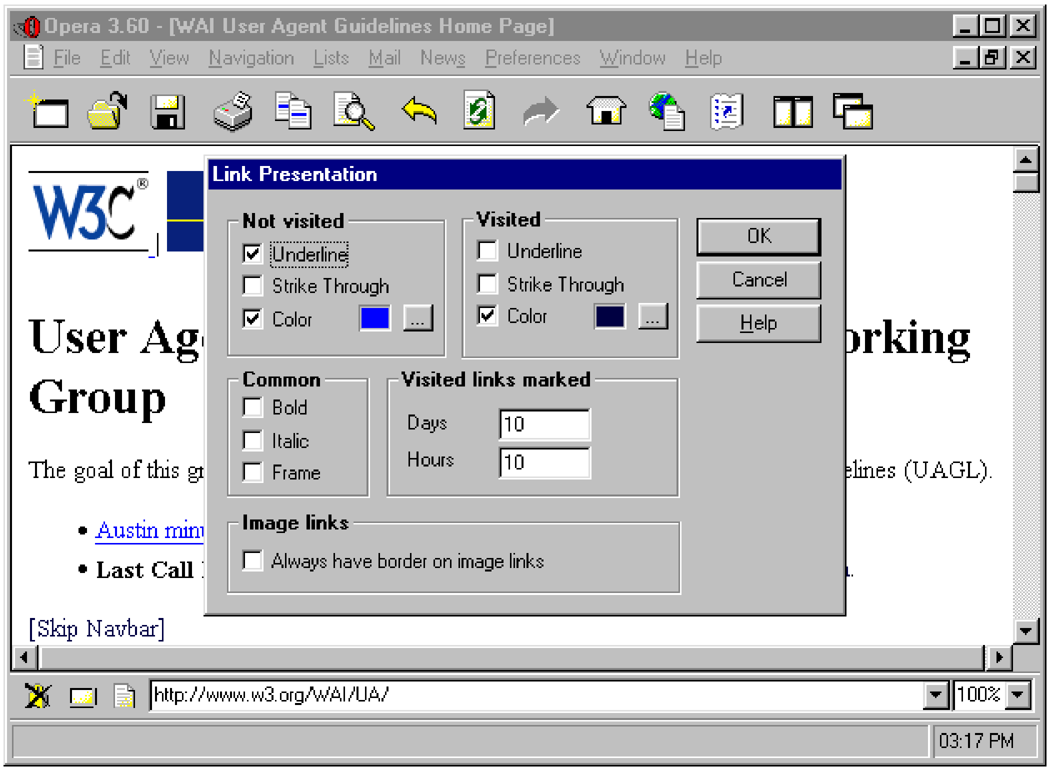
This image shows how Internet Explorer [IE-WIN] provides cell header information through the context menu.

8.2 Render recently visited links in a distinct style. Allow the user to configure this style and offer at least three rendering options, including colors and fonts. Allow the user to select from among the range of system colors and fonts. [Priority 2] (Checkpoint 8.2)

Note: Do not use color as the only distinguishing factor between visited and unvisited links as some users may not perceive colors and some devices may not render them. This checkpoint is an important special case of checkpoint 8.6.

Techniques:

- Do not rely on color alone. Refer to the visited links example in the section on generated content techniques .
- Refer to techniques for checkpoint 7.3.
- Refer to the section on link techniques .



This image shows how Opera [OPERA] allows the user to configure link rendering, including the identification of visited links.

8.3 Render in a distinct style those links that have been marked up to indicate that following them will involve a fee. Allow the user to configure this style and offer at least three rendering options, including colors and fonts. Allow the user to select from among the range of system colors and fonts. [Priority 2] (Checkpoint 8.3)

Note: This checkpoint is an important special case of checkpoint 8.6. The W3C specification "Common Markup for micropayment per-fee-links" [MICROPAYMENT] describes how authors may mark up micropayment information in an interoperable manner.

Techniques:

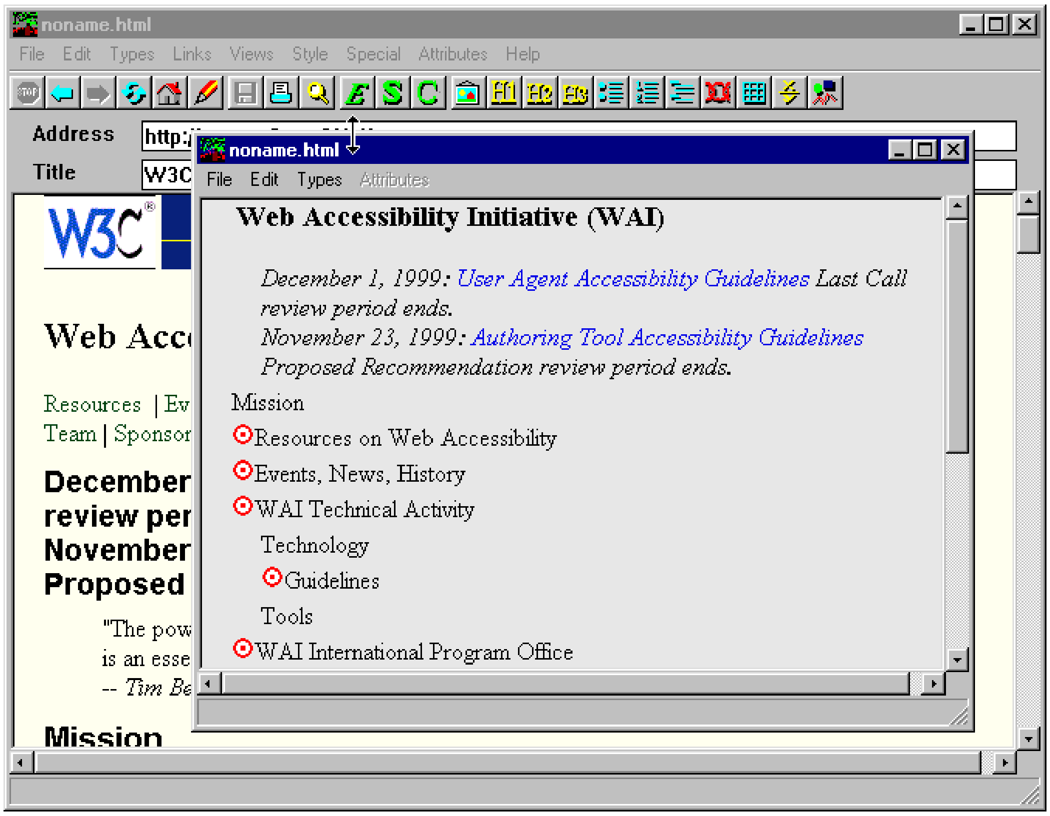
- Use standard, accessible interface controls to present information about fees and to prompt the user to confirm payment.
- For a link that has content focus , allow the user to query the link for fee information (e.g., by activating a menu or key stroke).
- Refer to the section on link techniques .

8.4 Make available to the user an "outline" view of content , composed of text labels for important structural elements (e.g., heading text, table titles, form titles, etc.). The set of important structural elements is the same required by checkpoint 7.6. [Priority 2] (Checkpoint 8.4)

Note: This checkpoint is meant to allow the user to simplify the view of content by hiding some content selectively. For example, for each frame in a frameset, provide a table of contents composed of headings (e.g., the H1 - H6 elements in HTML) where each entry in the table of contents links to the heading in the document. This checkpoint does not require that the outline view be navigable, but this is recommended; refer to checkpoint 7.6. For those elements that do not have associated text titles or labels, the user agent should use generate a brief text label (e.g., from content, the element type, etc.).

Techniques:

- Allow the user to expand or shrink portions of the outline view (configure detail level) for faster access to important parts of content.
- Hide portions of content by using the CSS 'display' and 'visibility' properties ([CSS2] , sections 9.2.5 and 11.2, respectively).
- Provide a structured view of form controls (e.g., those grouped by LEGEND or OPTGROUP in HTML) along with their labels.
- Refer to structured navigation techniques for checkpoint 7.6.
- For documents that do not use structure properly, user agents may attempt to create an outline based on the rendering of elements and heuristics about what elements may indicate about document structure.



This image shows the table of contents view provided by Amaya [AMAYA] . This view is synchronized with the "primary" view so that users may navigate in one viewport and the focus follows in the other. An entry in the table of contents with a target icon means that the heading in the document has an associated anchor.

8.5 Allow the user to configure and control the outline view of checkpoint 8.4 to include and exclude element types. [Priority 3] (Checkpoint 8.5)

Note: For example, allow the user to configure the level of detail of the outline. Refer also to checkpoint 8.4 and checkpoint 5.4.

Techniques:

- The CSS 'display' and 'visibility' properties ([CSS2] , sections 9.2.5 and 11.2, respectively), allow the user to override the default settings in user style sheets .

  Example.

  The following CSS 2 style sheet will turn the display off of all HTML elements inside the BODY element except heading elements:

```
<STYLE type="text/css">
   BODY * { display: none }
   H1, H2, H3, H4, H5, H6 { display: block }
</STYLE>
```

Another approach would be to use class selectors to identify those elements to hide or display.

End example.

8.6 To help the user decide whether to follow a link, make available to the user the following information: link content, link title, whether the link is internal, whether the link has been followed, whether following it may involve a fee, and information about the type, size, and natural language of the linked Web resource . [Priority 3] (Checkpoint 8.6)

Note: User agents are not required to retrieve the Web resource designated by a link as part of computing information about the link.

Techniques:

- For a link that has content focus , allow the user to query the link for information (e.g., by activating a menu or key stroke).
- Refer to the section on link techniques .

8.7 Allow the user to configure and control which link information required by checkpoint 8.6 to present. [Priority 3] (Checkpoint 8.7)

Note: Refer also to checkpoint 8.6.

Techniques:

- Allow configuration through style sheets. Refer to the section on generated content techniques .

Checkpoints for user interface accessibility:

8.8 Implement selection , content focus , and user interface focus mechanisms. Implement them according to system conventions per checkpoint 5.8. [Priority 1] (Checkpoint 8.8)

Note: This checkpoints refers to the *semantics* of the selection and focus; requirements for rendering are addressed by checkpoint 8.9 checkpoint 4.15, and checkpoint 4.14. Refer also to checkpoint 7.1.

Techniques:

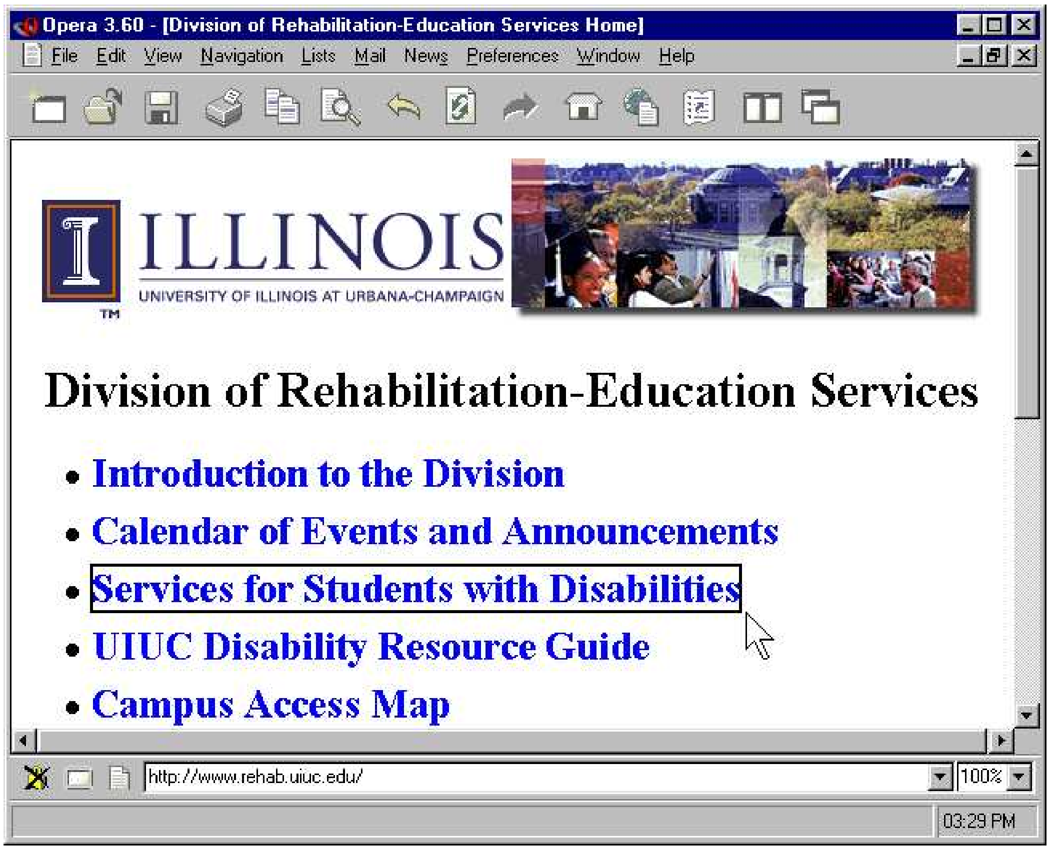
- Refer to Selection and Partial Selection of DOM Level 2 ([DOM2] , section 8.2.2).
- For information about focus in the Motif environment (under X Windows), refer to the OSF/Motif Style Guide [MOTIF] .

8.9 Provide a mechanism for highlighting and identifying (through a standard interface where available) the current viewport , selection , and content focus . [Priority 1] (Checkpoint 8.9)

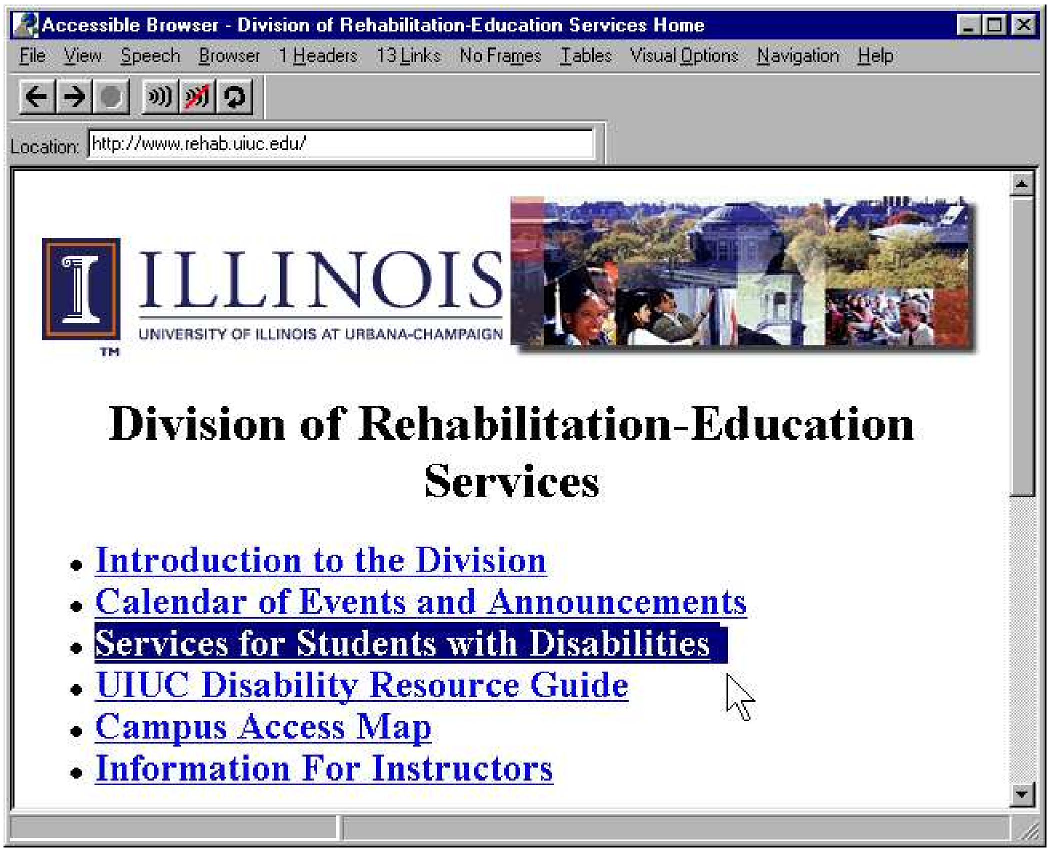
Note: This includes highlighting and identifying frames. This checkpoint is an important special case of checkpoint 1.1. Refer also to checkpoint 8.6.

Techniques:

- If colors are used to highlight the current viewport, selection, or content focus, allow the user to configure these colors. Do not rely on color alone.
- Provide a setting that causes a window that is the current viewport to pop to the foreground.
- Provide a setting that causes a window that is the current viewport to be maximized automatically. For example, maximize the parent window of the browser when launched, and maximize each child window automatically when it receives focus . Maximizing does not necessarily mean occupying the whole screen or parent window; it means expanding the current window so that users have to scroll horizontally or vertically as little as possible.
- If the current viewport is a frame or the user does not want windows to pop to the foreground, use colors, reverse videos, or other graphical clues to indicate the current viewport.
- For speech or Braille output, use the frame or window title to identify the current viewport. Announce changes in the current viewport.
- Use operating system conventions, for specifying selection and content focus (e.g., schemes in Windows).
- Implement the ':hover', ':active', and ':focus' pseudo-classes of CSS 2 ([CSS2] , section 5.11.3). This allows users to modify content focus rendering with user style sheets .
- Refer to the section on frame techniques .



This image shows how Opera [OPERA] uses a solid line border to indicate content focus.



This image shows how the Accessible Web Browser [AWB] uses the system highlight colors to indicate content focus.

8.10 Provide a mechanism for highlighting and identifying active elements . [Priority 2] (Checkpoint 8.10)

Note: On most systems, the focus is used to identify and highlight active elements.

Techniques:

- Allow users to configure highlighting preferences.
- Do not rely on color alone to identify active elements.
- Implement the ':hover', ':active', and ':focus' pseudo-classes of CSS 2 ([CSS2] , section 5.11.3).
- Implement CSS attribute selectors to match elements with associated scripts ([CSS2] , section 5.8).

Guideline 9. Notify the user of content and viewport changes.

Checkpoints for user interface accessibility:

9.1 Ensure that when the selection or content focus changes, it is in a viewport after the change. [Priority 2] (Checkpoint 9.1)

Note: For example, if users navigating links move to a portion of the document outside the viewport, the viewport should scroll to include the new location of the focus.

Techniques:

- There are times when the content focus changes (e.g., link navigation) and the viewport must be moved to track it. There are other times when the viewport changes position (e.g., scrolling) and the content focus is moved to follow it. In both cases, the focus (or selection) is in the viewport after the change.
- If a search causes the selection or focus to change, ensure that the found content is not hidden by the search prompt.
- When the content focus changes, register the newly focused element in the navigation sequence; sequential navigation should start from there.
- Unless viewports have been synchronized explicitly, changes to selection or focus in one viewport should not affect the selection or focus in another viewport.

9.2 Allow configuration so the user is prompted to confirm any form submission not caused by explicit activation of a form submit control. [Priority 2] (Checkpoint 9.2)

Note: For example, do not submit a form automatically when a menu option is selected, when all fields of a form have been filled out, or when a mouseover event occurs.

Techniques:

- Allow the user to configure script-based submission (e.g., triggered by an "onChange" event). For instance, allow these settings:
  1. Do not allow script-based submission.
  2. Allow script-based submission after confirmation from the user.
  3. Allow script-based submission without prompting the user (but not by default).
- A configuration to prompt the user before any form submission would satisfy this checkpoint.
- Users who navigate a document serially may think that the submit button in a form is the "last" control they need to complete before submitting the form. Therefore, for forms in which additional controls follow a submit button, if those controls have not been completed, inform the user and ask for confirmation (or completion) before submission.
- pwWebSpeak [PWWEBSPEAK] generates an explicit form submit button when the author has not provided one.

9.3 Allow the user to configure notification preferences for common types of content and viewport changes. [Priority 3] (Checkpoint 9.3)

Note: For example, allow the user to choose to be notified (or not) that a script has been executed, that a new viewport has been opened, that a pulldown menu has been opened, that a new frame has received focus , etc.

Techniques:

- Refer to the section on frame techniques .
- Allow the user to specify an element type for which notification should be disabled (e.g., TABLE, BODY, and IMG in HTML).
- Allow the user to disable notification of changes to CSS properties.
- Allow the user to disable notification of images that are changed (e.g., animations composed of a sequence of images).

9.4 When transferring content (e.g., a document, image, audio, video, etc.) indicate what percentage of the content has been transferred and whether the transfer has stalled. [Priority 3] (Checkpoint 9.4)

Techniques:

- Indicate progress of loading and when loading has finished, for example with a percentage indication. For images that render gradually (coarsely to finely), it is not necessary to show percentages for each rendering pass.
- Provide status information in a device-independent manner. For instance, proportional scroll bars on graphical viewports do not benefit users who cannot make use of the graphical information. Complement this mechanism, for example, with text in the status bar (or in the scroll mechanism itself) indicating the percentage viewed.
- Provide this information automatically and allow users to query the viewport for it (e.g., through a menu or keyboard binding).
- Allow users to configure when to render status information so that assistive technologies may announce changes in status at appropriate times. For instance, allow the user to hide the status bar in order to hide a text rendering.
- User agents may provide additional information about the proportion of the content that has been transferred (e.g., number of bytes).
- Allow users to configure what status information they want rendered. Useful status information includes:
  - Document proportions (numbers of lines, pages, width, etc.);
  - Number of elements of a particular type (e.g., tables, forms, and headings);
  - Whether the viewport is at the beginning or end of the document;
  - Size of document in bytes;
  - The number of controls in a form and controls in a form control group (e.g., FIELDSET in HTML).

9.5 Indicate as a percentage the relative position of the viewport in rendered content (e.g., the percentage of an audio or video clip that has been played, the percentage of a Web page that has been viewed, etc.). [Priority 3] (Checkpoint 9.5)

Note: The user agent may calculate the percentage according to content focus position, selection position, or viewport position, depending on how the user has been browsing.

Techniques:

- Provide a scrollbar for the viewport. Some specifications address scrolling requirements or suggestions explicitly, such as for the THEAD and TBODY elements of HTML 4.01 ([HTML4] , section 11.2.3) and the 'overflow' property of CSS 2 ([CSS2] , section 11.1.1).
- Indicate the size of the document, so that users may decide whether to download for offline viewing. For example, the playing time of an audio file could be stated in terms of hours, minutes, and seconds. The size of a primarily text-based Web page might be stated in both kilobytes and screens, where a screen of information is calculated based on the current dimensions of the viewport.
- Indicate the number of screens of information, based on the current dimensions of the viewport (e.g., "screen 4 of 10").
- Use a variable pitch audio signal to indicate the viewport's different positions.
- Provide standard markers for specific percentages through the document.
- Provide markers for positions relative to some position – a user selected point, the bottom, the H1, etc.
- Put a marker on the scrollbar, or a highlight at the bottom of the page while scrolling (so you can see what was the bottom before you started scrolling.

Guideline 10. Allow configuration and customization.

Checkpoints for user interface accessibility:

10.1 Provide information to the user about current user preferences for input configurations (e.g., keyboard or voice bindings). [Priority 1] (Checkpoint 10.1)

Techniques:

- Refer to input configuration techniques .

10.2 Avoid default input configurations that interfere with operating system accessibility conventions. [Priority 1] (Checkpoint 10.2)

Note: In particular, default configurations should not interfere with operating conventions for keyboard accessibility. Information about operating system accessibility conventions is available in the Techniques document [UAAG10-TECHS] . Refer also to checkpoint 5.8.

Techniques:

- The default configuration should not include "**`Alt`**-**`F4`**", "**`Control`**-**`Alt`**-**`Delete`**", or other combinations that have reserved meanings on a given operating system.
- Clearly document any default configurations that depart from system conventions.
- Some reserved keyboard bindings are listed in the appendix on accessibility features of some operating systems .

10.3 Provide information to the user about current author-specified input configurations (e.g., keyboard bindings specified in HTML documents with the "accesskey" attribute). [Priority 2] (Checkpoint 10.3)

Techniques:

- Refer to input configuration techniques .
- Provide information about which keys activate form controls.

10.4 Allow the user to change the default input configuration as follows: Allow the user to override any binding that is part of the user agent default input configuration. Allow the user to override the default keyboard bindings with a binding that consists of modifier keys and a single key, or a single key. The user agent is not required to allow the user to override standard bindings for the operating system (e.g., for access to help). [Priority 2] (Checkpoint 10.4)

Note: This checkpoint applies to all input methods: keyboard, voice, graphical user interface, etc. The override requirement only applies to bindings for the same input method (i.e., the user must be able to override a keyboard binding with another keyboard binding). Refer also to checkpoint 10.5, checkpoint 10.9, checkpoint 10.8, and checkpoint 11.3.

Techniques:

- Refer to input configuration techniques .
- Allow users to restore easily the default input configuration.
- When using a physical keyboard, some users require single-key access (refer to checkpoint 10.5), others require that keys activated in combination be physically close together, while others require that they be spaced physically far apart.
- Allow users to select from among pre-packaged configurations, to override some of the chosen configuration, and to save it as a profile . Not only will the user save time configuring the user agent, but this will reduce questions to technical support personnel.
- Do not allow the user to override important user agent or operating system configurations (e.g., to quit the user agent or reconfigure it). Refer to input configuration techniques .
- Allow users to create macros and bind them to key strokes or other input

methods.

10.5 Allow the user to override the default keyboard configuration as follows: Allow the user to override any binding that is part of the user agent default keyboard configuration. Allow the user to assign a single key binding to a majority of the functionalities that are available in the default keyboard configuration. The user agent is not required to allow the user to override standard keyboard bindings for the operating system (e.g., for access to help). [Priority 2] (Checkpoint 10.5)

Note: This checkpoint applies to the keyboard only and always applies on systems with a standard keyboard API . In some modes of interaction (e.g., when the user is entering text), the number of available single keys will be significantly reduced. The number of available single keys will also be determined by the keyboard device capabilities. This checkpoint is an important special case of checkpoint 10.4. Refer also to checkpoint 1.3, checkpoint 10.9, checkpoint 10.8, and checkpoint 11.3.

Techniques:

- Refer to input configuration techniques .
- Opera [OPERA] includes a mode in which users can access important user agent functionalities with single strokes from the numeric keypad.
- Mouse Keys (available on some operating systems) allow users to simulate the mouse through the keyboard. They provide a usable command structure without interfering with the user interface for users who do not require keyboard-only and single-key access.

10.6 Follow operating system conventions to indicate the input configuration . [Priority 2] (Checkpoint 10.6)

Note: For example, on some operating systems, developers may specify which command sequence will activate a functionality so that the standard user interface components display that binding. For example, if a functionality is available from a menu, the letter of the activating key will be underlined in the menu. This checkpoint is an important special case of checkpoint 5.8.

Techniques:

- Refer to input configuration techniques .
- Use system conventions to indicate the current configuration (e.g., in menus, indicate what key strokes will activate the functionality, underline single keys that will work in conjunction with a trigger key such as **`Alt`**, etc.) These are conventions used by the Sun Java Foundations Classes [JAVA-TUT] and Microsoft Foundations Classes for Windows.
- Ensure that information about changes to the input configuration is available in a device-independent manner (e.g., through visual and audio cues, and through text).
- If the currently active configuration changes locally (e.g., a search prompt opens, changing the keyboard mapping for the duration of the prompt), alert

the user.

- Named configurations are easier to remember. This is especially important for people with certain types of cognitive disabilities. For example, if the invocation of a search prompt changes the input configuration, the user may remember more easily which key strokes are active in search mode if alerted that there is a "Search Mode". Context-sensitive help (if available) should reflect the change in mode, and a list of keybindings for the current mode should be readily available to the user.

10.7 For the configuration requirements of this document, allow the user to save user preferences in at least one user profile . Allow users to select from among available profiles or no profile (i.e., the user agent default settings). [Priority 2] (Checkpoint 10.7)

Note: The configuration requirements of the checkpoints in this document involve user preferences for styles, presentation rates, input configurations , navigation, viewport behavior, and user agent notification.

Techniques:

- Follow applicable operating system conventions for input configuration profiles .
- Allow users to choose a different profile, to switch rapidly between profiles, and to return to the default input configuration.

10.8 Ensure that the default input configuration allows easy activation of frequently used functionalities. [Priority 3] (Checkpoint 10.8)

Note: checkpoint 10.4 Make the most frequent operations easy to access and operable through a single command.

Techniques:

- Refer to input configuration techniques .
- Provide different input configuration profiles (e.g., one keyboard profile with key combinations close together and another with key combinations far apart).
- Test the default keyboard configuration for usability. Ask users with different disabilities and combinations of disabilities to test configurations.
- Provide convenient bindings to functionalities that promote accessibility such as navigation of links.
- Provide convenient bindings for controlling the user interface, such as showing, hiding, moving, and resizing graphical viewports .
- For people using one hand, a few fingers, or a headwand pointer, access to important functionalities must be available through one or at most two key strokes.
- Consider distance between keys and key alignment (e.g., "9/I/K", which align almost vertically on many keyboards) in the default configuration. For instance, if **`Enter`** is used to active links, put other link navigation

commands near it (e.g., page up/down, arrow keys, etc. on many keyboards). In configurations for users with reduced mobility, pair related functionalities on the keyboard (e.g., left and right arrows for forward and back navigation).
- Allow users to accomplish tasks through repeated key strokes (e.g., sequential navigation) since this means less physical repositioning for all users. However, repeated key strokes may not be efficient for some tasks. For instance, do not require the user to position the pointing device by pressing the "down arrow" key repeatedly.
- So that users do not mistakenly activate certain functionalities, make certain combinations "more difficult" to invoke (e.g., users are not likely to press **`Control`**-**`Alt`**-**`Delete`** accidentally).
- Avoid deeply nested graphical menus.
- Input configurations should allow quick and direct navigation that does not rely on graphical output. Do not require the user to navigate through a graphical user interface as the only way to activate a functionality.
- Offer a mode that makes the input configuration compatible with other versions of the software (or with other software).
- Refer also to checkpoint 10.6.

10.9 For graphical user interfaces, allow the user to configure the position of controls on tool bars of the user agent user interface , to select or remove controls for the user interface from a predefined set, and to restore the default user interface. [Priority 3] (Checkpoint 10.9)

Note: This checkpoint is an important special case of checkpoint 10.4.

Techniques:

- Allow multiple icon sizes (big, small, other sizes).
- Allow the user to choose icons and/or text.
- Allow the user to change the grouping of icons.
- Allow the user to show and hide controls. This benefits users with cognitive disabilities and users who navigate user interface controls sequentially.
- Allow the user to change the position of control bars, icons, etc. Do not rely solely on drag-and-drop for reordering tool bar. Allow the user to configure the user agent user interface in a device-independent manner (e.g., through a text-based profile).

Guideline 11. Provide accessible product documentation and help.

Checkpoints for accessible documentation:

11.1 Provide a version of the product documentation that conforms to the Web Content Accessibility Guidelines 1.0 [WCAG10] . [Priority 1] (Checkpoint 11.1)

Note: User agents may provide documentation in many formats, but at least one must conform to the Web Content Accessibility Guidelines 1.0 [WCAG10] .

Techniques:

- Distribute accessible documentation over the Web, on CD-ROM, or by telephone. Alternative hardcopy formats may also benefit some users.
- Documentation includes information bundled with a product when it is released as well as information made available subsequently (e.g., bug fixes, etc.).
- Web-based support and/or documentation that is produced or maintained by the manufacturer of a user agent or by a sub-contractor of the user agent's developer must conform to the Web Content Accessibility Guidelines 1.0 [WCAG10] . In particular:
  1. Provide text equivalents of all non-text content (e.g., graphics, audio-only presentations , etc.);
  2. Provide extended descriptions of screen-shots, flow charts, etc.;
  3. Provide a text equivalent for audio user agent tutorials. Tutorials that use speech to guide a user through the operation of the user agent should also be available at the same time as graphical representations.
  4. Use clear and consistent navigation and search mechanisms;
  5. Use the NOFRAMES element when the support/documentation is presented in a FRAMESET;
  6. Refer also to checkpoint 11.3.
- Describe the user interface with device-independent terms. For example, use "select" instead of "click on".
- Provide documentation in small chunks (for rapid downloads) and also as a single source (for easy download and/or printing). A single source might be a single HTML file or a compressed archive of several HTML documents and included images.
- Ensure that run-time help and any Web-based help or support information is accessible and may be operated with a single, well-documented, input command (e.g., key stroke). Use operating system conventions for input configurations related to run-time help.
- Provide documentation in alternative formats such as Braille (refer to "Braille Formats: Principles of Print to Braille Transcription 1997" [BRAILLEFORMATS]), large print, or audio tape. Agencies such as Recording for the Blind and Dyslexic [RFBD] and the National Braille Press [NBP] can create alternative formats.
- Provide accessible documentation for all audiences: end users, developers, etc. For instance, developers with disabilities may wish to add accessibility features to the user agent, and so require information on available APIs and other implementation details.
- Ensure that product identification codes are accessible to users so they

may install their software. Codes printed on product cases will not be accessible to people with visual disabilities.

11.2 Document all user agent features that promote accessibility. [Priority 1] (Checkpoint 11.2)

Note: For example, review the documentation or help system to ensure that it includes information about the accessibility requirements of WAI Guidelines.

Techniques:

- Refer also to techniques for checkpoint 11.4.
- Provide a sensible index to accessibility features. For instance, users should be able to find "How to turn off blinking text" in the documentation (and the user interface). The user agent may implement this feature by turning off scripts, but users should not have to guess (or know) that turning off scripts will turn off blinking text.
- Document configurable features in addition to defaults for those features.
- Document the features implemented to conform with these guidelines.
- Include references to accessibility features in both the table of contents and index of the documentation.

11.3 Document the default input configuration (e.g., default keyboard bindings). [Priority 1] (Checkpoint 11.3)

Techniques:

The following table shows how one might document keyboard bindings. It shows the default keyboard configuration for versions of Netscape Navigator [NAVIGATOR] running on the Macintosh, Unix, and Windows operating systems. If a function exists in the browser but does not have a binding, its corresponding cell is marked with an asterisk. If the function does not exist, it is left blank. **Note:** This table lists some, but not all, functionalities and keyboard bindings of Navigator. It is meant to illustrate, not serve as definitive documentation for Netscape Navigator.

Some entries contain links to special notes. The number in parentheses following the link is the number of the relevant note.

Note: To make this table accessible, a linear version of Navigator Keyboard Bindings is available.

Navigator Keyboard Bindings

Function	**Macintosh (v 4.61)**	**Unix (v 4.51)**	**Windows (v 4.7)**
Move within a document			

Scroll to next page	**Page Down**	**Page Down**	**Page Down**
Scroll to previous page	**Page Up**	**Page Up**	**Page Up**
Scroll to top	*	*	**Control-Home**
Scroll to bottom	*	*	**Control-End**
Move between documents			
Open a new document	**Command+L**	**Alt+O**	**Control+O**
Stop loading a document	**Command+.**	**Esc**	**Esc**
Refresh a document	**Command+R**	**Alt+R**	**Control+R**
Load previous document	**Command+[** or **Command+Left Arrow**	**Alt+Left Arrow**	**Alt+Left Arrow**
Load next document	**Command+]** or **Command+Right Arrow**	**Alt+Right Arrow**	**Alt+Right Arrow**
Navigate elements within a document			
Move focus to next frame	*	*	*
Move focus to previous frame	*	*	*
Move focus to next active element (1)	**Tab**	**Tab**	**Tab**
Move focus to previous active element (1)	**Shift+Tab**	**Shift+Tab**	**Shift+Tab**
Find word in page	**Command+F**	**Alt+F**	**Control+F**

Act on HTML elements			
Select a link	*	*	**`Enter`**
Toggle a check box	*	*	**`Shift`** or **`Enter`**
Activate radio button	*	*	**`Shift`**
Move focus to next item in an option box	*	*	**`Down Arrow`** or **`Right Arrow`**
Move focus to previous item in an option box	*	*	**`Up Arrow`** or **`Left Arrow`**
Select item in an option box	*	*	**`Enter`**
Press a button (2)	**`Return, Space`**	**`Enter, Space`**	**`Enter, Space`**
Navigate menus			
Activate menu	*	*	**`Alt+`** the **`underlined letter in the menu title`**
Deactivate menu	*	**`Esc`**	**`Esc`**
Move focus to next menu item	*	* (3)	**`Down Arrow`**
Move focus to previous menu item	*	* (3)	**`Up Arrow`**
Select menu item	*	**`underlined letter in the menu item`**	**`Enter`**
Move focus to submenu	*	* (3)	**`Right Arrow`**

Move focus to main menu	*	* (3)	**`Left Arrow`**
Navigate bookmarks			
View bookmarks menu	* (4)	*	**`Alt+C+B`**
Move focus to next item in bookmarks menu	Down Arrow (4)	*	**`Down Arrow`**
Move focus to previous item in bookmarks menu	Up Arrow (4)	*	**`Up Arrow`**
Select item in bookmarks menu	Return (4)	*	**`Enter`**
Add bookmark	**`Command+D`**	**`Alt+K`**	**`Control+D`**
Edit bookmarks	**`Command+B`**	**`Alt+B`**	**`Control+B`**
Delete current bookmark (5)	**`Delete`**	**`Alt+D`**	**`Delete`**
Navigate history list			
View history list	**`Command+H`**	**`Alt+H`**	**`Control+H`**
Move focus to next item in history list	*	*	**`Down Arrow`**
Move focus to previous item in history list	*	*	**`Up Arrow`**
Move focus to first item in history list	*	*	**`Left Arrow`**
Select item in history list	*	*	**`Enter`** (6)

Close history list	**`Command+W`**	**`Alt+W`**	**`Control+W`**
Define view			
Increase font size (7)	**`Shift+Command+]`**	**`Alt+]`**	**`Control+]`**
Decrease font size (7)	**`Shift+Command+[`**	**`Alt+[`**	**`Control+[`**
Change font color	*	*	*
Change background color	*	*	*
Turn off author-defined style sheets	*	*	*
Turn on user-defined style sheets (8)	?	?	?
Apply next user-defined style sheet	?	?	?
Apply previous user-defined style sheet	?	?	?
Other functionalities			
Access to documentation	*	*	*

Notes.

1. In Windows, active elements of the user interface include links, text entry boxes, buttons, checkboxes, radio buttons, etc. In Unix and Macintosh, **`Tab`** cycles through text entry boxes only.
2. In Windows, this works for any button, since any button can gain the user interface focus using keyboard commands. In Unix and Macintosh, this only applies to the "Submit" button following a text entry.
3. In Unix, the menus cannot be opened with binding keys. However, once a menu is opened it stays opened until it is explicitly closed, which means

that the menus can still be used with shortcut keys to some extent. Sometimes left and right arrows move between menus and up and down arrows move within menus, but this does not seem to work consistently, even within a single session.
4. In Macintosh, you cannot explicitly view the bookmarks menu. However, if you choose "Edit Bookmarks", which does have a keyboard binding, you can then navigate through the bookmarks and open bookmarked documents in the current window.
5. To delete a bookmark you must first choose "Edit Bookmarks" and then move the focus to the bookmark you want to delete.
6. In Windows, when you open a link from the history menu using **`Enter`**, the document opens in a new window.
7. All three systems have menu items (and corresponding shortcut keys) meant to allow the user to change the font size. However, the menu items are consistently inactive in both Macintosh and Unix. The user seems to be able to actually change the font sizes only in Windows.
8. It is important to allow users to set their own Cascading Style Sheets. Although Navigator does currently allow the user to override the author's choice of foreground color, background color, font, and font size, it does not allow some of the advanced capabilities that make CSS so powerful. For example, a blind user may want to save a series of style sheets which show only headings, only links, etc., and then view the same page using some or all of these style sheets in order to orient himself to the organization of the page before reading the page.

11.4 In a dedicated section of the documentation , describe all features of the user agent that promote accessibility. [Priority 2] (Checkpoint 11.4)

Note: This is a more specific requirement than checkpoint 11.2.

Techniques:

- Integrate information about accessibility features throughout the documentation. The dedicated section on accessibility should provide access to the documentation as a whole rather than standing alone as an independent section. For instance, in a hypertext-based help system, the section on accessibility may link to pertinent topics elsewhere in the documentation.
- Ensure that the section on accessibility features is easy to find.

11.5 In each software release, document all changes that affect accessibility. [Priority 2] (Checkpoint 11.5)

Note: Features that affect accessibility are listed in this document and in platform-specific accessibility guidelines.

Techniques:

- At a minimum provide a text description of changes (e.g., in a README file).
- In particular, document changes to the user interface.

---

3 Accessibility topics

This section presents general accessibility techniques that may apply to more than one checkpoint.

3.1 Access to content

User agents must ensure that users have access to content , either rendered through the user interface or made available to assistive technologies through an API . While providing serial access to a stream of content would satisfy this requirement, this would be analogous to offering recorded music on a cassette: other technologies exist (e.g., CD-ROMs) that allow direct access to music. It is just as important for user agents to allow users to access Web content efficiently, whether the content is being rendered as a two-dimensional graphical layout, an audio stream, or a line-by-line Braille stream. Providing efficient access to content involves:

- Preserving structure when rendering
- Allowing the user to select specific content and query its structure or context
- Providing access to equivalent alternatives of content.
- Using and generating metadata to provide context

These topics are addressed below.

3.1.1 Preserve and provide structure

When used properly, markup languages structure content in ways that allow user agents to communicate that structure across different renderings. A table describes relationships among cells and headers. Graphically, user agents generally render tables as a two-dimensional grid. However, serial renderings (e.g., speech and Braille) must also make those relationships apparent, otherwise users will not understand the purpose of the table and the relationships among its cells (refer to the section on table techniques). User agents must render content in ways that allow users to understand the underlying document structure, which may consist of headings, lists, tables, synchronized multimedia, link relationships, etc. Providing alternative renderings (e.g., an outline view) will also help users understand document structure.

Note: Even though the structure of a language like HTML is defined by a Document Type Definition (DTD), user agents may convey structure according to a "more intelligent" document model when that model is well-known. For instance, in the HTML DTD, heading elements (H1 - H6) do not nest, but presenting the document as nested headings may convey the document's structure more effectively than as a flat list of headers.

3.1.2 Allow access to selected content

The guidelines emphasize the importance of navigation as a way to provide efficient access to content. Navigation allows users to access content more quickly and when used in conjunction with selection and focus mechanisms, allows users to query content for metadata. For instance, blind users often navigate a document by skipping from link to link, deciding whether to follow each link based on metadata about the link. User agents can help them decide whether to follow a link by allowing them to query each focused link for the link text, title information, information about whether the link has been visited, whether the link involves a fee, etc. While much of this information may be rendered, the information must also be available to assistive technologies.

For example, the Amaya browser/editor [AMAYA] makes available all attributes and their values to the user through a context menu. The user selects an element (e.g., with the mouse) and opens an attribute menu that shows which attributes are available for the element and which are set. The user may read or write values to attributes (since Amaya is an editor). Information about attributes is also available through Amaya's structured view, which renders the document tree as structured text.

The selection may be widened (moved to the nearest node one level up the document tree) by pressing the **`Escape`** key; this is a form of structured navigation based on the underlying document object model .

Users may want to select content based on structure alone (as offered by Amaya) but also based on how the content has been rendered. For instance, most user agents allow users to select ranges of text content that may cross "element boundaries".

3.1.3 Access to equivalent alternatives of content

Authors provide equivalent alternatives to content so that users may understand the function of a page or part of a page even though they may not be able to make use of a particular content type. For example, authors must provide text equivalents for non-text content (e.g., images, video, audio-only presentations , etc.) because text may be rendered as speech or Braille and may be used by users with visual or hearing or both disabilities. User agents must ensure that these alternatives are available to users, either through the user interface or through an API .

How authors specify equivalent alternatives depends on the markup language used. For information about equivalent alternatives for SMIL [SMIL] content, refer to "Accessibility Features of SMIL" [SMIL-ACCESS] . In HTML 4.01 [HTML4] , authors specify equivalent alternatives for content as follows:

- For the IMG element (section 13.2): the "alt" (section 13.8), "title" (section 7.4.3), and "longdesc" (section 13.2) attributes. Refer to the section on long descriptions .
- For the OBJECT element (section 13.3): the content of the element and the "title" attribute.
- For the deprecated APPLET element (section 13.4): the "alt" attribute and the content of the element.
- For the AREA element (section 13.6.1): the "alt" attribute.
- For the INPUT element (section 17.4): the "alt" attribute.
- For the ACRONYM and ABBR elements (section 9.2.1): the "title" attribute (for acronym or abbreviation expansion).
- For the TABLE element (section 11.2.1): the "summary" attribute.
- For frames: the NOFRAMES element (section 16.4.1) and the "longdesc" attribute (section 16.2.2) on FRAME and IFRAME (section 16.5).
- For scripts: the NOSCRIPT element (section 18.3.1).

Techniques for providing access to equivalent alternatives include the following:

- Make information available with different levels of detail. For example, for a voice browser, offer two options for equivalent alternatives to HTML images:
  1. Speak only "alt" text by default, but allow the user to hear "longdesc" text on an image by image basis.
  2. Speak "alt" text and "longdesc" for all images.
- Allow the user to configure how the user agent renders a long description (e.g., "longdesc" in HTML 4.01 [HTML4]). Some possibilities include:
  1. Render the long description in a separate view.
  2. Render the long description in place of the associated element.
  3. Do not render the long description, but allow the user to query whether an element has an associated long description (e.g., with a context-sensitive menu) and provide access to it.
  4. Use an icon (with a text equivalent) to indicate the presence of a long description.
  5. Use an audio cue to indicate the presence of a long description when the user navigates to the element.
- For an object (e.g., an image) with an author-specified geometry that the user agent does not render, allow the user to configure how the equivalent alternative should be rendered. For example, within the specified geometry, by ignoring the specified geometry altogether, etc.
- For multimedia presentations with several alternative tracks, ensure access to all tracks and allow the user to select individual tracks. The Quicktime player [QUICKTIME] allows users to turn on and off any number of tracks separately.
- For multimedia presentations with several alternative tracks, allow users to select tracks based on natural language preferences. SMIL 1.0 [SMIL] allows users to specify captions in different natural languages. By setting language preferences in the SMIL player (e.g., the G2 player [G2]), users may access

captions (or audio) in different languages. Allow users to specify different languages for different content types (e.g., English audio and Spanish captions).
- For missing equivalent alternatives of content:
  - The "Altifier Tool" [ALTIFIER] illustrates smart techniques for generating text equivalents (for images, etc.) when the author has not specified any.
  - If no captioning information is available and captioning is turned on, render "no captioning information available" in the captioning region of the viewport.

3.1.4 Context

Authors and user agents provide context to users through content, structure, navigation mechanisms, and query mechanisms. Titles, dimensions, dates, relationships, the number of elements, and other metadata all help orient the user, particularly when available as text. For instance, user agents can help orient users by allowing them to request that document headings and lists be numbered. Refer also to the section on table techniques , which explains how users agents can offer table navigation and the ability to query a table cell for information about the cell's row and column position, associated header information, etc.

- User agents can use style sheet languages such as CSS 2 [CSS2] and XSLT [XSLT] to generate context information (refer to techniques for generated content).
- For information about elements and attributes that convey metadata in HTML, refer to the index of elements and attributes in "Techniques for Web Content Accessibility Guidelines 1.0" [WCAG10-TECHS] .
- For information about elements and attributes that convey metadata in SMIL, refer to the index of attributes in the W3C Note "Accessibility Features of SMIL" [SMIL-ACCESS] .
- Describe a selected element's position within larger structures (e.g., numerical or relative position in a document, table, list, etc.). For example: tenth link of fifty links; document heading 3.4; list one of two, item 4.5; third table, three rows and four columns; current cell in third row, fourth column; etc. Allow users to get this information on demand (e.g., through a keyboard shortcut). Provide this information on the status line on demand from the user.

3.2 User control of style

To ensure accessibility, users must be able to configure the style of rendered content and the user interface. Author-specified styles, while important, may make content inaccessible to some users. User agents must allow users to increase the size of text (e.g., with a zoom mechanism or font size control), to change colors and color combinations, to slow down multimedia presentations, etc.

To give authors design flexibility and allow users to control important aspects of content style, user agents should implement CSS ([CSS1] , [CSS2]) and allow users to create and apply user style sheets . CSS includes mechanisms for tailoring rendering for a particular output medium, including audio, Braille, screen, and print.

- User agents should implement the cascade order of CSS 2 ([CSS2] , section 6.4.1) not CSS 1. In CSS 2, user style sheets with "!important" declarations (section 6.4.2) take precedence over author styles . Refer also to Web Content Accessibility Guidelines 1.0 checkpoint 3.3 [WCAG10] .
- CSS-enabled user agents should consider as part of the cascade the markup used for style, giving it a lower weight than actual style sheets. This allows authors to specify style through markup for older user agents and to use more powerful style sheets for CSS-enabled user agents. Refer to the section on the precedence of non-CSS presentational hints in CSS 2 ([CSS2] , section 6.4.4).
- To hide the CSS syntax from the user, user agents may implement user style sheets through the user agent user interface . User agents can generate a user style sheet from user preferences or behave as though it did. Amaya [AMAYA] provides a GUI-based interface to create and apply internal style sheets. The same technique may be used to control a user style sheet.
- For animations rendered natively, allow users to control the rate of animation, to pause and play animations, to step through the animation, and to play it at the specified rate.
- Allow the user to pause a video presentation, to move, resize, and position tracks that appear on the screen (including captions, subtitles and signed translations) and to apply CSS stylesheets to text-based presentation.
- In the user interface:
    - Allow the user to select large or small buttons and controls. Ensure that these values are applied consistently across the user interface.
    - Allow the user to regroup buttons and controls, and reorder menus.
    - Use standard operating system controls for allowing configuration of font sizes, speech rates, and other style parameters.

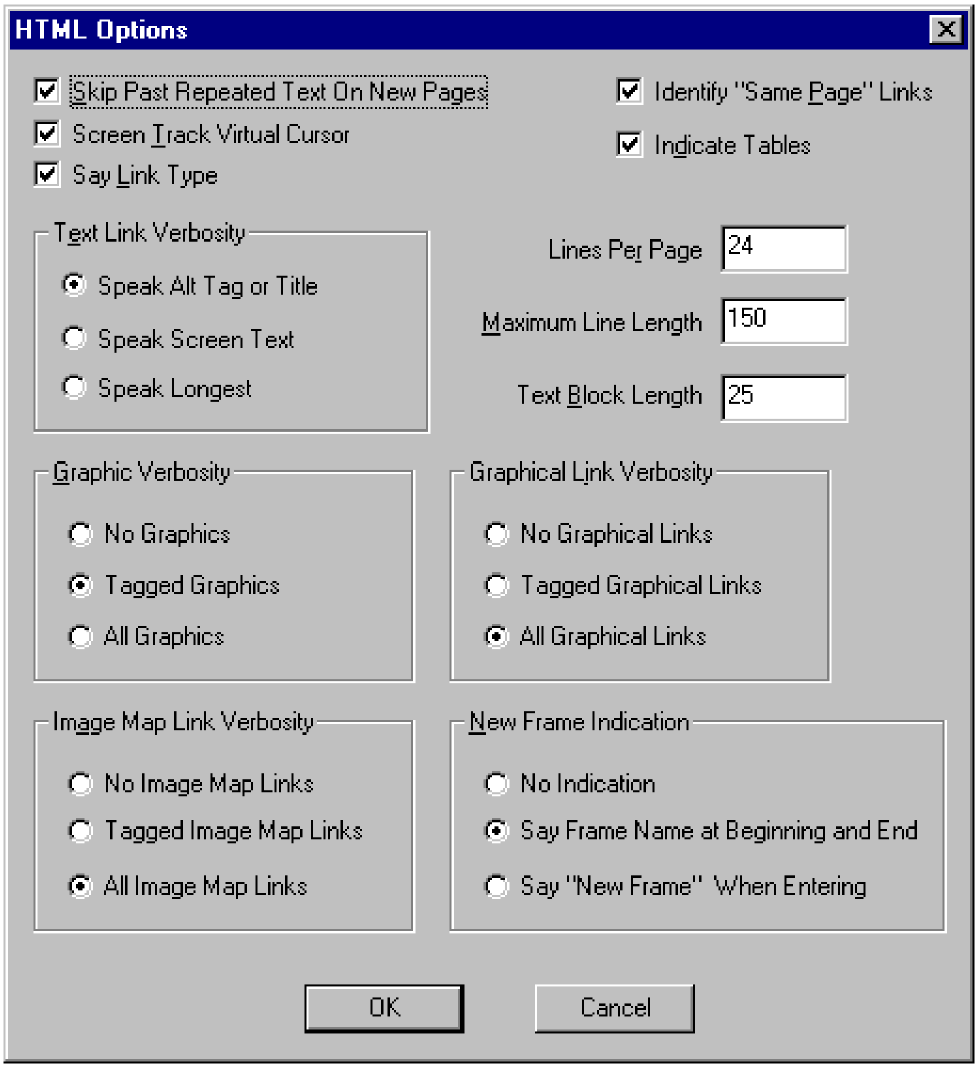
3.3 Link techniques

User agents make links accessible by providing navigation to links, helping users decide whether to follow them, and allowing interaction in a device-independent manner. Link techniques include the following:

- Refer to sequential navigation techniques for information about navigating to links.
- Provide a link view that lists all links in the document. Allow the user to configure how the links are sorted (e.g., by document order, sequential navigation order, alphabetical order, visited or unvisited or both, internal or external or both, etc.).
- Help the user remember links by including metadata in the link view. For example, identify a selected link as "Link X of Y", where "Y" is the total number of links. Lynx [LYNX] numbers each link and provides information about the relative position in the document. Position is relative to the current page and the number of the current page out of all pages. Each page usually has 24 lines.
- Allow the user to configure how much information about a link to present in the

content view (when a link receives focus). For instance, allow the user to choose between "Display links using hyperlink text" or "Display links by title (if present)", with an option to toggle between the two views. For a link without a title, use the link text.

- For links with non-text content such as images, make available a text equivalent as follows:
    1. If the author has specified a non-empty text equivalent for the image (e.g., "alt" in HTML), use that as the link text;
    2. Otherwise, use the link title if available;
    3. Otherwise, use title information of the designated Web resource (e.g., the TITLE element of HTML for links to HTML documents).
    4. Otherwise, render part of the filename or URI of the designated Web resource .
    5. Otherwise, insert a generic placeholder (e.g., [LINK]) in place of the image.
- For an image in link content, ensure that the user has access to the link and any long description associated with the image.
- Ensure that all information about a link is available in a device-independent manner. For example, do not rely solely on fonts or colors to alert the user whether or not the link has previously been followed. Allow the user to configure how information will be presented (colors, sounds, status bar messages, some combination, etc.).
- If the user activates a broken link, leave the viewport where it is and alert the user (e.g., in the status bar and with a graphical or audio alert). Moving the viewport suggests that a link is not broken, which may disorient the user.
- If the focus is used to select active elements, implement the ':hover', ':active', and ':focus' pseudo-classes of CSS 2 ([CSS2] , section 5.11.3). This allows users to modify content focus presentation with user style sheets. Use them in conjunction with the CSS 2 ':before' pseudo-elements ([CSS2] , section 5.12.3) to clearly indicate that something is a link (e.g., 'A:before { content : "LINK:" }'). Refer also to techniques for generated content .
- Do not mark all local links (to anchors in the same page) as visited when the page has been visited.



As shown in the following image, JAWS for Windows [JFW] offers a view for configuring a number of rendering features, notably some concerning link types, text link verbosity, image map link verbosity, graphical link verbosity, and internal links.

3.4 List techniques

User agents can make lists accessible by ensuring that list structure – and in particular, embedded list structure – is available through navigation and rendering.

- Allow users to turn on "contextual" rendering of lists (even for unordered "bullet" lists). Use compound numbers (or letters, numbers, etc.) to introduce each list item (e.g., "1, 1.1, 1.2, 1.2.1, 1.3, 2, 2.1"). This provides more context and does not rely on the information conveyed by a graphical rendering, as in:

```
1.
  1.
  2.
    1.
  3.
2.
  1.
```

which might be serialized for speech or Braille as "1, 1, 2, 1, 2, 3, 2, 1".

- Specify list numbering styles in CSS. Refer to the section generated content, automatic numbering, and lists in CSS ([CSS2] , section 12).

  Example.

  The following CSS 2 style sheet (taken from CSS 2, section 12.5) shows how to specify compound numbers for nested lists created with either UL or OL elements. Items are numbered as "1", "1.1", "1.1.1", etc.

  ```
  <STYLE type="text/css">
     UL, OL { counter-reset: item }
     LI { display: block }
     LI:before { content: counters(item, "."); counter-increment: item }
  </STYLE>
  ```

  End example.

3.5 Table techniques

The HTML TABLE element was designed represent relationships among data ("data" tables). Even when authored well and used according to specification, tables may pose problems for users with disabilities for a number of reasons:

- Users who access a table serially (e.g., as speech or Braille) may have difficulty grasping the relationships among cells, especially for large and complex tables.
- Users who with cognitive disabilities may have trouble grasping or remembering relationships between cells and headers, especially for large and complex tables.
- Users of screen magnifiers or with physical disabilities may have difficulties navigating to the desired cells of a table.

For these situations, user agents may assist these users by providing table navigation mechanisms and supplying context that is present in a two-dimensional rendering (e.g., the cells surrounding a given cell).

To complicate matters, many authors use tables to lay out Web content ("layout" tables). Not only are table structures used to lay out objects on the screen, table elements such as TH (table header) in HTML are used to font styling rather than to indicate a true table header. These practices make it difficult for assistive technologies to rely on markup to convey document structure. Consequently, assistive technologies often must resort to interpreting the rendered content , even though the rendered content has "lost" information encoded in the markup. For instance, when an assistive technology "reads" a table from its graphical rendering, 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.
```

Screen readers that read rendered content line by line would read the table cells incorrectly as "This is the top left cell This is the top right cell". So that assistive technologies are not required to gather incomplete information from renderings, these guidelines require that user agents provide access to document source through an API (refer to checkpoint 5.3).

The following sections discuss techniques for providing improved access to tables.

3.5.1 Table metadata

Users of screen readers or other serial access devices cannot gather information "at a glance" about a two-dimensional table. User agents can make tables more accessible by providing the user with table metadata such as the following:

- The table caption (the CAPTION element in HTML) or summary information (the "summary" attribute in HTML).
- The number of column groups and columns. Note that the number of columns may change according to the row. Also, some parts of a table may have two dimensions, others three, others four, etc. Project dimensionality higher than two onto two when rendering information.
- The number of row groups and rows, in particular information about table headers and footers.
- Which rows contain header information (whether at the top or bottom of the table).
- Which columns contain header information (whether at the left or right of the table).
- Whether there are subheads.
- How many rows or columns a header spans.

When navigating, quick access to table metadata will allow users to decide whether to navigate within the table or skip over it. Other techniques:

- Allow users to query table summary information from inside a cell.
- Allow the user to choose different levels of detail for the summary (e.g., brief table summary and a more detailed summary).
- Allow the user to configure navigation so that table metadata is not (re-)rendered each time the user enters the table.

3.5.2 Linear rendering of tables

A linear rendering of tables -- cells presented one at a time, row by row or column by column -- may be useful, but generally only for simple tables. For more complex tables, user agents need to convey more information about relationships among cells and their headers.

Note: The following techniques apply to columns as well as rows. The elements listed in this section are HTML 4.01 table elements ([HTML4] , section 11).

- Provide access to one row at a time, beginning with any column header. If a header is associated with more than one row, offer that header for each row concerned.
- Render cells with their associated headers. Allow the user to configure how often headers are rendered (e.g., by implementing the 'speak-header' property in CSS 2 [CSS2] , section 17.7.1). Note also that the "abbr" attribute in HTML 4.01 specifies abbreviated headers for speech and other rendering ([HTML4] , section 11.2.6). Refer also to information about cell headers later in this section.
- Provide access to cell content as marked up in the document source.
- Refer to techniques for authoring accessible tables in "Techniques for Web Content Accessibility Guidelines 1.0" [WCAG10-TECHS] .

3.5.3 Cell rendering

The most important aspect of rendering a table cell is that the cell's contents be rendered faithfully and be identifiable as the contents of a single cell. However, user agents may provide additional information to help orient the user:

- Render the row and column position of the cell in the table.
- Indicate how many rows and columns a cell spans.
- Since the contents of a cell in a data table may only be comprehensible in context (i.e., with associated header information, row/column position, neighboring cell information etc.), allow users to navigate to cells and query them for this information.
- For HTML tables, refer to the section on associating header information with data cells of HTML 4.01 ([HTML4] , section 11.4.1).
- In a table with a leading row and column of TH cells, the interpretation of the corner cell as an empty TD or TH should not contribute to the set of headings for cells in that row and column.
- For nested tables, render information about the level of nesting.
- Since a cell may belong to N different dimensions in a multi-dimensional table, provide information about headers from each dimension.

3.5.4 Cell header algorithm

Properly constructed data tables distinguish header cells from data cells. How headers are associated with table cells depends on the markup language. The following algorithm is based on the HTML 4.01 algorithm to calculate header information ([HTML4] , section 11.4.3). For the sake of brevity, it assumes a left-to-right ordering, but will work for right-to-left tables as well (refer to the "dir" attribute of HTML 4.01 [HTML4] , section 8.2). For a given cell:

- Search left from the cell's position to find row header (TH) cells. Then search upwards from the cell's position to find column header cells. The search in a given direction stops when the edge of the table is reached or when a data cell is found after a header cell. If no headers are found in either direction (left or up), search in the other directions (right or down).
- Allow the user to configure where the header text comes from. For example, in HTML 4, either the header cell's text content or the value of the "abbr" attribute value ([HTML4] , section 11.2.6).
- Insert row headers into the list in the (left-to-right) order they appear in the table. Include values implicitly resulting from header cells in prior rows with `rowspan="R"`, sufficient to extend into the current row.
- Insert column headers after row headers, in the (top-to-bottom) order they appear in the table. Include values implicitly resulting from header cells in other columns with `colspan="C"`, sufficient to extend into the current column containing the TD cell.
- If a header cell has a value for the "headers" attribute, then insert it into the list and stop the search for the current direction.
- Treat cells with a value for the "axis" attribute as header cells.
- Be sure to take into account header cells that span several rows or columns.

Not all data tables include proper header markup, which the user agent may be able to detect. Some repair strategies for finding header information include the following:

- Consider that the top or bottom row contains header information.
- Consider that the leftmost or rightmost column in a column group contains header information.
- If cells in an edge row or column span more than one row or column, consider the following row or column to contain header information as well.
- When trying to guess table structure, present several solutions to the user.

Other repair issues to consider:

- TH cells on both the left and right of the table need to be considered.
- 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.
- An internal TH surrounded by TDs makes it difficult to know whether the header applies to cells to its left or right in the same row (or in both directions) or cells

above or below it in the same column (or in both directions).
- Finding column header cells assumes they are all above the TD cell to which they apply.
- A TH with "colspan" set needs to be included in the list of THs for the N-1 columns to its right.

3.5.5 Table navigation

To permit efficient access to tables, user agents should allow users to navigate to tables and within tables, to select individual cells, and to query them for information about the cell and the table as a whole.

- Allow users to navigate to a table, down to one of its cells, and back up to the table level. This should work recursively for nested tables.
- Allow users to navigate to a cell by its row and column position.
- Allow users to navigate to all cells under a given header.
- Allow users to navigate row by row or column by column.
- Allow users to navigate to the cells around the current cell.
- Allow users to navigate to the first or last cell of a row, column, or the table.
- Allow users to navigate from a cell directly to its related headers (if it's possible to navigate to the headers).
- Allow the user to search for text content within a table (i.e., without searching outside of the table). Allow the user to search for text content within specific rows or columns, row groups or column groups, or limited by associated headers.
- Notify the user when the navigation reaches a table edge and when a cell contains another table.
- Allow relative and direct navigation. For example, entering "-3, 20" might mean "left three cells, up 20 cells").
- Allow navigation of table headers or footers only.
- Consider the issues raised by navigation to or from a cell that spans more than one row or column.
- For examples of table navigation, refer to the table navigation script from the Trace Research Center [TABLENAV] .

3.6 Image map techniques

One way to make an image map accessible is to render the links it contains as text links. This allows assistive technologies to render the links a speech or Braille, and benefits users with slow access to the Web and users of small Web devices that do not support images but can support hypertext. User agents may allow users to toggle back and forth between a graphical mode for image maps and a text mode.

To construct a text version of an image map in HTML:

- If the content of the MAP element includes links, use them.
- Otherwise, for each AREA in the map, if a (non-null) text equivalent is available (the "alt" attribute), use it as the content of a generated link.
- When the author has specified a null text equivalent, do not render the link.
- When the author has not specified a text equivalent, render (for example) "Map area" followed by part of the URI of the link.

Furthermore, user agents that render a text image map instead of an image may preface the text image map with inline metadata such as:

- a string that announces the image map (e.g., "Start map")
- any text equivalent associated with the image (e.g., "alt" for IMG).
- the number of links in the map.

Allow users to suppress, shrink, and expand text versions of image maps so that they may quickly navigate to an image map (which may be, for example, a navigation tool bar) and decide whether to "expand" it and follow the links of the map. The metadata listed above will allow users to decide whether to expand the map. Ensure that the user can expand and shrink the map and navigate its links using the keyboard and other input devices.

3.7 Frame techniques

Frames were originally designed so that authors could divide up graphic real estate and allow the pieces to change independently (e.g., selecting an entry in a table of contents in one frame changes the contents of a second frame). While frames are not inherently inaccessible, they raise some accessibility issues:

- Alternatives to frame content. Some users cannot make use of frames because they cannot grasp the (spatial or logical) relationships conveyed by frame layout. Others cannot use them because their user agents or assistive technology does not support them or makes access difficult (e.g., users with screen readers or screen magnifiers).
- Navigation. Users must be able to navigate from frame to frame in a device independent manner.
- Orientation. Users need to know what frame they are in (thus, frames must be titled), what other frames are available, and how the frames of a frameset are organized.
- Dynamic changes. Users need to know how the changes they cause in one frame affect other frames.

To name a frame in HTML, use the following algorithm:

1. Use the "title" attribute on FRAME, or if not present,
2. Use the "name" attribute on FRAME, or if not present,
3. Use title information of the referenced frame source (e.g., the TITLE element of the source HTML document), or

4. Use title information of the referenced long description (e.g., what "longdesc" refers to in HTML), or
5. Use frame context (e.g., "Frame 2.1.3" to indicate the path to this frame in nested framesets).

To make frames accessible, user agents should do the following:

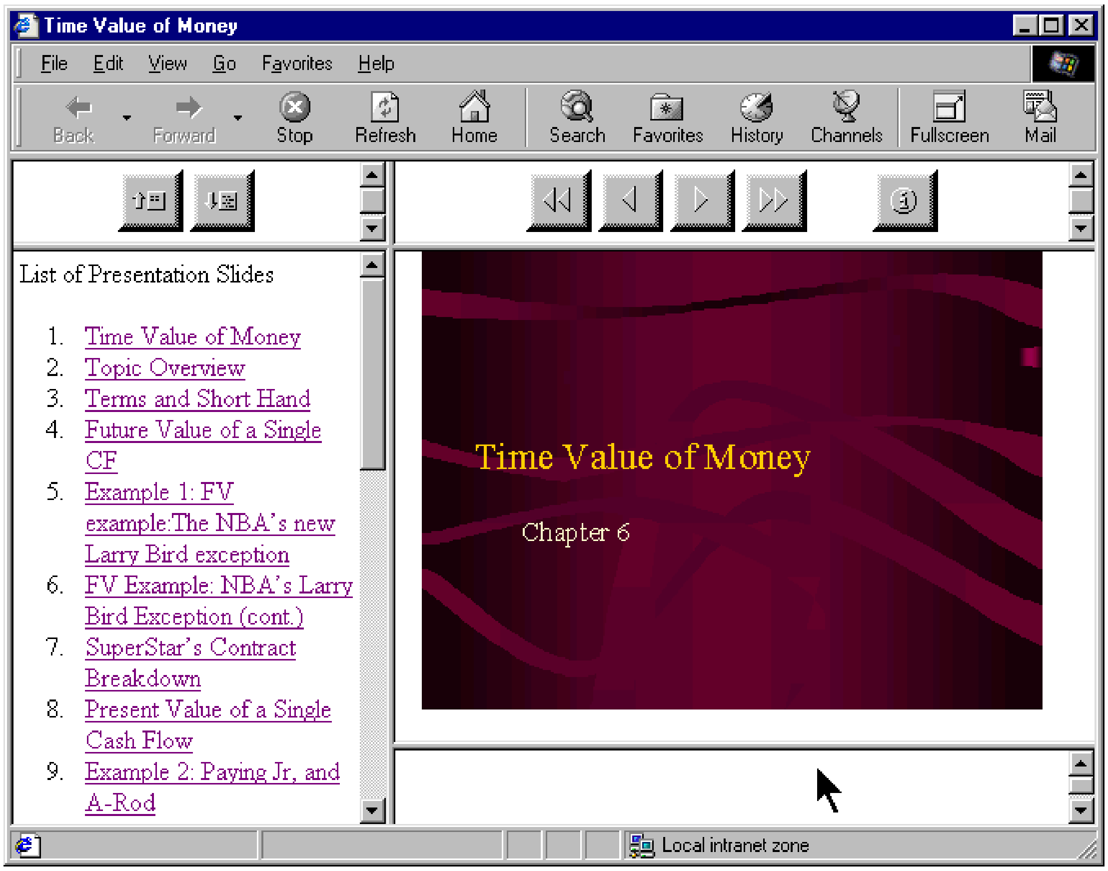
- Make available the author-specified frame equivalents (e.g., provided by the HTML 4.01 NOFRAMES element ([HTML4] , section 16.4.1).
- Here is a technique for the case of a frameset that does not contain a NOFRAMES equivalent but the individual frames have associated long descriptions ("longdesc"):
  1. For each frameset, render the frameset title as an H1 heading.
  2. For each frame, render the frame title in an H2 heading, followed by the content of the associated long description.
  3. Create a navigable table of contents according to the (possibly nested) frameset structure. Each entry in the table of contents should link to a frameset or frame. The end of the content used for each frame should include a link back to this table of contents.
- Notify the user when the viewport contains a frameset.
- Render a frameset as a list of links to named frames so the user can identify the number of frames. The list of links may be nested if framesets are nested.
- Provide information about the number of frames in the frameset.
- Highlight the current frameset (e.g., with a thick border, by displaying the name of the current frameset in the status bar, etc.)
- Allow the user to query the current frame for metadata about the frame. Make available the frame title for speech synthesizers and Braille displays. Users may also use information about the number of images and words in the frame to guess the purpose of the frame. For example, few images and few words is probably a title, more words is probably an index, many words is probably text area.
- Allow navigation between frames (forward and backward through the nested structure, return to global list of links to frames). **Note:** Recall that the user must be able to navigate frames through all supported input devices.
- Allow navigation to frame equivalents.
- Allow the user to bookmark the current frame.
- Notify the user when an action in one frame causes the content of another frame to change. Allow the user to navigate quickly to the frame(s) that changed.
- Authors can suppress scrolling of frames with `scrolling="no"`. In this case, the user agent must make available content that is not in the viewport.
- The user agent may ignore some attributes of the FRAME element of HTML 4.01 ([HTML4] , section 16.2.2): "noresize", "scrolling", and "frameborder".

Consider renderings of the following document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML lang="en">
<HEAD>
  <META http-equiv="Content-Type"
          content="text/html; charset=iso-8859-1">
  <TITLE>Time Value of Money</TITLE>
</HEAD>

<FRAMESET COLS="*, 388">
  <FRAMESET ROWS="51, *">
    <FRAME src="sizebtn" marginheight="5" marginwidth="1"
       name="Size buttons" title="Size buttons">
    <FRAME src="outlinec" marginheight="4" marginwidth="4"
       name="Presentation Outline"
           title="Presentation Outline">
  </FRAMESET>

  <FRAMESET ROWS="51, 280, *">
    <FRAME src="navbtn" marginheight="5" marginwidth="1"
       name="Navigation buttons"
       title="Navigation buttons">
    <FRAME src="slide001" marginheight="0" marginwidth="0"
      name="Slide Image" title="Slide Image">
    <FRAME src="note001" name="Notes" title="Notes">
  </FRAMESET>
<NOFRAMES>
<P>List of Presentation Slides</P>
<OL>
<LI><A HREF="slide001">Time Value of Money</A>
<LI><A HREF="slide002">Topic Overview</A>
<LI><A HREF="slide003">Terms and Short Hand</A>
<LI><A HREF="slide004">Future Value of a Single CF</A>
<LI><A HREF="slide005">Example 1: FV example:The
NBAŸs new Larry Bird exception</A>
<LI><A HREF="slide006">FV Example: NBAŸs Larry
Bird Exception (cont.)</A>
<LI><A HREF="slide007">SuperStarŸs Contract
Breakdown</A>
<LI><A HREF="slide008">Present Value of a Single
Cash Flow</A>
<LI><A HREF="slide009">Example 2: Paying Jr, and
A-Rod</A>
<LI><A HREF="slide010">Example 3: Finding Rate of
Return or Interest Rate</A>
<LI><A HREF="slide011">Annuities</A>
<LI><A HREF="slide012">FV of Annuities</A>
<LI><A HREF="slide013">PV of Annuities</A>
<LI><A HREF="slide014">Example 4: Invest Early in
an IRA</A>
<LI><A HREF="slide015">Example 4 Solution</A>
<LI><A HREF="slide016">Example 5: Lotto Fever
</A>
<LI><A HREF="slide017">Uneven Cash Flows: Example
6:Fun with the CF function</A>
<LI><A HREF="slide018">Example 6 CF worksheet inputs</A>
```

```
<LI><A HREF="slide019">CF inputs continued</A>
<LI><A HREF="slide020">Non-Annual Interest
Compounding</A>
<LI><A HREF="slide021">Example 7: What rate are
you really paying?</A>
<LI><A HREF="slide022">Nominal to EAR Calculator</A>
<LI><A HREF="slide023">Continuous Interest Compounding</A>
<LI><A HREF="slide024">FV and PV with non-annual
interest compounding</A>
<LI><A HREF="slide025">Non-annual annuities</A>
<LI><A HREF="slide026">Example 8: Finding Monthly
Mortgage Payment</A>
<LI><A HREF="slide027">solution to Example 8</A>
</OL>
</NOFRAMES>
</FRAMESET>
</HTML>
```

The following examples show how some user agents handle this frameset.

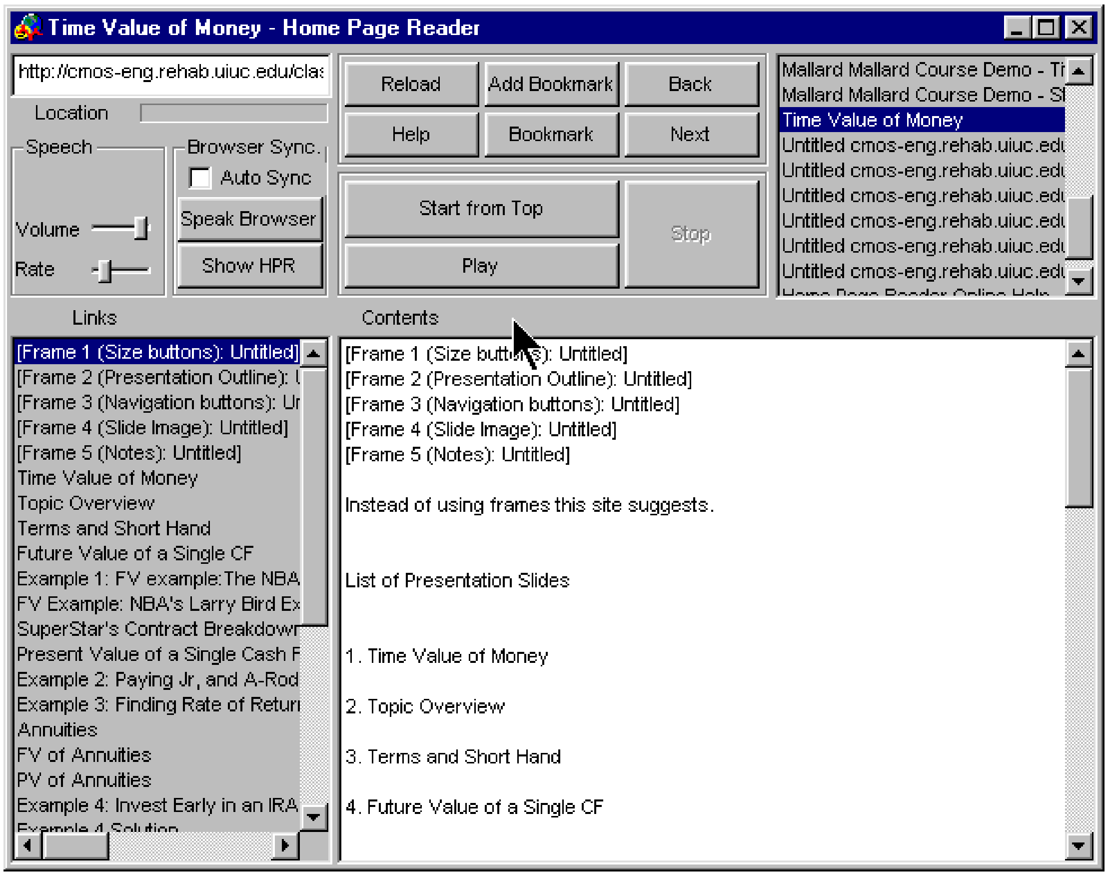


Rendering of a frameset by Internet Explorer [IE-WIN] .

Rendering by Lynx [LYNX] :

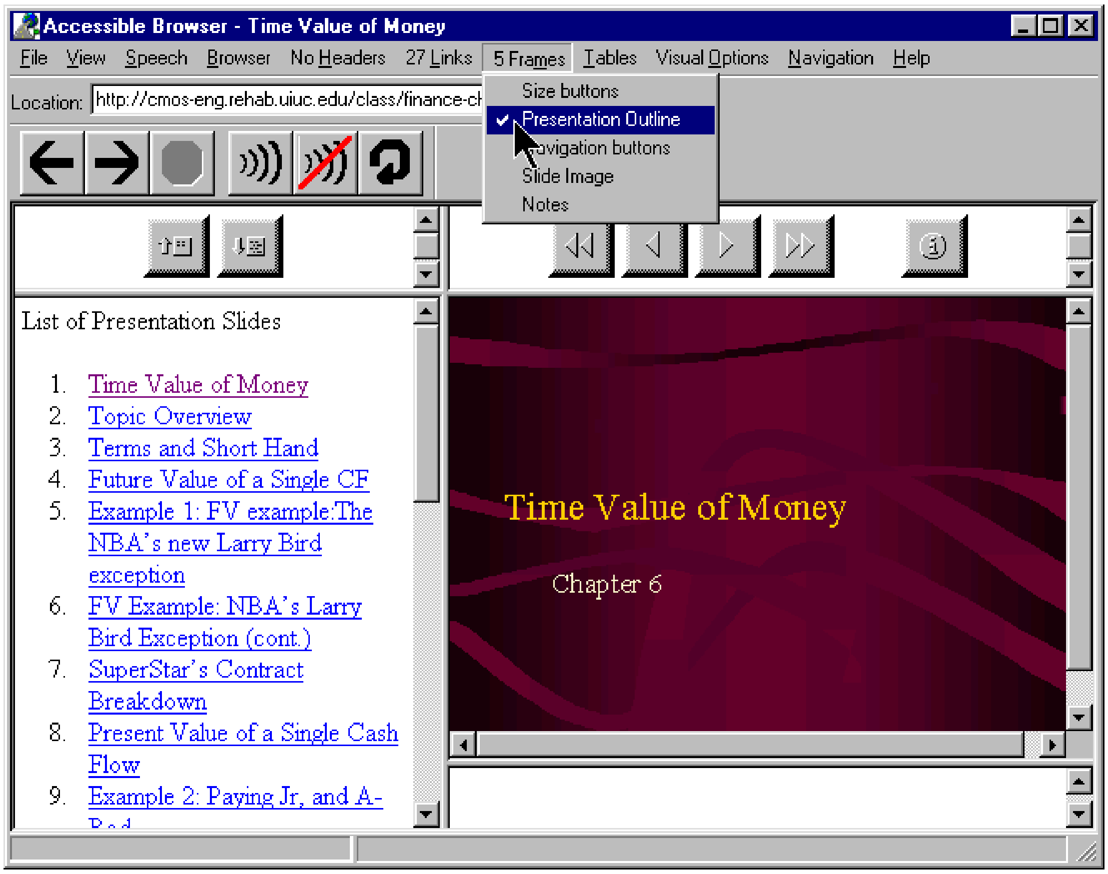
```
                              Time Value of Money

FRAME: Size buttons
FRAME: Presentation Outline
FRAME: Navigation buttons
FRAME: Slide Image
FRAME: Notes

List of Presentation Slides
 1. Time Value of Money
 2. Topic Overview
 3. Terms and Short Hand
 4. Future Value of a Single CF
 5. Example 1: FV example:The NBA's new Larry Bird exception
 6. FV Example: NBA's Larry Bird Exception (cont.)
 7. SuperStar's Contract Breakdown
 8. Present Value of a Single Cash Flow
 9. Example 2: Paying Jr, and A-Rod
10. Example 3: Finding Rate of Return or Interest Rate
11. Annuities
12. FV of Annuities
13. PV of Annuities
```

```
14. Example 4: Invest Early in an IRA
15. Example 4 Solution
16. Example 5: Lotto Fever
17. Uneven Cash Flows: Example 6:Fun with the CF function
18. Example 6 CF worksheet inputs
19. CF inputs continued
20. Non-Annual Interest Compounding
21. Example 7: What rate are you really paying?
22. Nominal to EAR Calculator
23. Continuous Interest Compounding
24. FV and PV with non-annual interest compounding
25. Non-annual annuities
26. Example 8: Finding Monthly Mortgage Payment
27. solution to Example 8
```



Rendering of a frameset by Home Page Reader [HPR] .

User agents may also indicate the number of frames in a document and which frame is the current frame via the menu bar or popup menus. Users can configure the user agent to include a FRAMES menu item in their menu bar. The menu bar makes the information highly visible to all users and is very accessible to assistive technologies.



In this image of the Accessible Web Browser [AWB] , the menu bar indicates the number of frames and uses a check mark next to the name of the current frame.

3.8 Form techniques

To make forms accessible, user agents need to ensure that users may interact with them in a device-independent manner, that users can navigate to the various form controls, and that information about the form and its controls is available on demand.

3.8.1 Form navigation techniques

- Allow users to navigate to forms and to all controls within a form (refer also to table navigation techniques). Opera [OPERA] and Navigator [NAVIGATOR] provide such functionality in a non-interactive manner, a "form navigation" keyboard commands. When invoked, these "form navigation" commands move the user agent's current focus to the first form control (if any) in the document.
- If there are no forms in a document and the user attempts to navigate to a form, alert the user.
- Provide a navigable, structured view of form controls (e.g., those grouped by LEGEND or OPTGROUP in HTML) along with their labels.
- For labels explicitly associated with form controls (e.g., "for" attribute on LABEL

in HTML), make available label information when the user navigates among the form controls.
- As the user navigates to a form control, provide information about whether the control must be activated before form submission.
- Allow the user to navigate away from a menu without selecting any option (e.g., by pressing the **`Escape`** key).
- As the user navigates to a form control, provide information (e.g., through context-sensitive help) about how the user can activate the control. Provide information about what is required for each form control. Lynx [LYNX] conveys this information by providing information about the currently selected form control via a status line message:
  - Radio Button: Use right-arrow or **`Return`** to toggle
  - Checkbox Field: Use right-arrow or **`Return`** to toggle
  - Option List: Press return and use arrow keys and return to select option
  - Text Entry Field: Enter Text. Use **`Up`** or **`Down`** arrows or **`Tab`** to move off
  - Textarea: Enter text. **`Up`** or **`Down`** arrows or **`Tab`** to move off (^Ve for editor) **Note:** The ^Ve (caret-V, e) command, included in the TEXTAREA status line message, enables the user to invoke an external editor defined in the local Lynx configuration file (`lynx.cfg`).

3.8.2 Form orientation techniques

Provide the following information about forms on demand:

- The number of forms in the document.
- The percentage of a form that has already been filled out. This will help users with serial access to form controls know whether they have completed the form. Otherwise, users who encounter a submit button that is not the last control of the form might inadvertently submit the incomplete form.

3.8.3 Form control orientation techniques

Provide the following information about the controls in a form on demand (e.g., for the control with focus):

- Indicate the number of controls in the form.
- Indicate the number of controls that have not yet been completed.
- Provide a list of controls that must be activated before form submission.
- Provide information about the order of form controls (e.g., as specified by "tabindex" in HTML). This is important since:
  1. Most forms are visually oriented, employing changes in font size and color.
  2. Users who access forms serially need to know they have supplied all the necessary information before submitting the form.
- Provide information about which control has focus (e.g., "control X of Y for the form named "MyForm"). The form name is very important for documents that contain more than one form. This will help users with serial access to form controls know whether they have completed the form.

- Allow the user to query a form control for information about title, value, grouping, type, status, and position.
- When a group of radio buttons receives content focus , identify the radio button with content focus as "Radio Button X of Y", where "Y" represents the total number of radio buttons in the group. HTML 4.01 specifies the FIELDSET element ([HTML4] , section 17.10), which allows authors to group thematically related controls and labels. The LEGEND element ([HTML4] , section 17.10) assigns a caption to a FIELDSET. For example, the LEGEND element might identify a FIELDSET of radio buttons as "Connection Rate". Each button could have a LABEL element ([HTML4] , section 17.9.1) stating a rate. When it receives content focus, identify the radio button as "Connection Rate: Radio button X of Y: 28.8kpbs", where "Y" represents the total number of radio buttons in the grouping and "28.8kbps" is the information contained in the LABEL.
- Allow the user to invoke an external editor instead of editing directly in a TEXTAREA control. This allows users to use all the features of the external editor: macros, spell-checkers, validators, known input configurations, backups and local copies, etc.
- Provide an option for transforming menus into checkboxes or radio buttons. In the transformation, retain the accessibility information specified by the author for the original form controls. Preserve the labels provided for the OPTGROUP and each individual OPTION, and re-associate them with the generated checkboxes. The LABEL defined for the OPTGROUP should be converted into a LEGEND for the result FIELDSET, and each checkbox should retain the LABEL defined for the corresponding OPTION. Lynx [LYNX] does this for HTML SELECT elements that have the "multiple" attribute specified.

3.8.4 Form submission techniques

Some users do not want forms to be submitted without their consent. The following techniques address user control of form submissions:

- Allow the user to turn off scripts, as authors may write scripts that submit a form when particular events occur (e.g., "onchange" events). Be aware of this type of practice:

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

  As soon as the user attempts to navigate the menu, the "switchpage" function opens a document in a new viewport. Try to avoid orientation problems that may be caused by scripts bound to form controls.

- Offer a configuration to prevent (or allow) automatic submission of forms.
- Allow the user to request confirmation before any form submission not initiated by the user . This should be the default setting. Allow the user to suppress future prompts or to change the setting to "always/never/prompt".
- Be aware that users may inadvertently pressing the **`Return`** or **`Enter`** key and accidentally submit a form.

3.9 Generated content techniques

User agents may help orient users by generating additional content that "announces" a context change. This may be done through CSS 2 [CSS2] style sheets using a combination of selectors (including the ':before' and ':after' pseudo-elements described in section 12.1) and the 'content' property (section 12.2).

For instance, the user might choose to hear "language:German" when the natural language changes to German and "language:default" when it changes back. This may be implemented in CSS 2 with the ':before' and ':after' pseudo-elements ([CSS2] , section 5.12.3)

For example, with the following definition in the stylesheet:

```
[lang|=es]:before { content: "start Spanish "; }
[lang|=es]:after  { content: " end Spanish"; }
```

the following HTML example:

```
<P lang="es" class="Spanish">
 <A href="foo_esp.html"
    hreflang="es">Esta pagina en español</A></P>
```

might be spoken "start Spanish _Esta pagina en espanol_ end Spanish". Refer also to information on matching attributes and attribute values useful for language matching in CSS 2 ([CSS2] , section 5.8.1).

The following example uses style sheets to distinguish visited from unvisited links with color and a text prefix.

The phrase "Visited link:" is inserted before every visited link:

```
A:link           { color: red }     /* For unvisited links */
A:visited        { color: green }   /* For visited links */
A:visited:before { content: "Visited link: "; }
```

To hide content, use the CSS 'display' or 'visibility' properties ([CSS2] , sections 9.2.5 and 11.2, respectively). The 'display' property suppresses rendering of an entire subtree. The 'visibility' property causes the user agent to generate a rendering structure, but display it invisibly (which means it takes up space, but cannot be seen).

The following XSLT style sheet (excerpted from the XSLT Recommendation [XSLT] , Section 7.7) shows how to one might number H4 elements in HTML with a three-part label.

Example.

```
<xsl:template match="H4">
   <fo:block>
     <xsl:number level="any" from="H1" count="H2"/>
     <xsl:text>.</xsl:text>
     <xsl:number level="any" from="H2" count="H3"/>
     <xsl:text>.</xsl:text>
     <xsl:number level="any" from="H3" count="H4"/>
     <xsl:text> </xsl:text>
     <xsl:apply-templates/>
   </fo:block>
  </xsl:template>
```

End example.

3.10 Script and applet techniques

User agents must make dynamic content accessible to users who may be disoriented by changes in content, who may have a physical disability that prevents them from interacting with a document within a time interval specified by the author, or whose user agent does not support scripts or applets. Not only must user agents make available equivalent alternatives to dynamic content, they must allow users to turn off scripts, to stop animations, adjust timing parameters, etc.

3.10.1 Script techniques

Certain elements of a markup 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 4.01 ([HTML4] , section 18.2.3), intrinsic events are specified by 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".

Techniques for providing access to scripts include the following:

- Allow the user to configure the user agent so that mouseover/mouseout events may be triggered by (and trigger) focus/blur events. Similarly, allow the user to use a key command, such as "enter" and "**`Shift`**-enter" to trigger "onclick" and "ondblclick" events.
- Implement DOM 2 [DOM2] events with a single activation event and provide a method for triggering that event from each supported input device or input API. These should be the same as the click events and mappings provided above (but note that a user agent which is also an editor may wish to use single click events for moving a system caret, and want to provide a different behavior to activate using the mouse). For example, Amaya [AMAYA] uses a "doAction" command for activating links and form controls, which can be triggered either by the mouse (and it is possible to set it for single-click or double-click) or by the keyboard (it is possible to set it for any key using Amaya's keyboard configuration)

- Allow the user to stop and start the flow of changes made by scripts. Prompt the user for confirmation of a pending change. **Note:** Some user agents allow users to turn off scripts for security reasons.
- Document the effects of known important scripts to give users an idea in advance of what they do. Make script source available to users so that those familiar with the scripting language may be able to understand their effects.

3.10.2 Applet techniques

When a user agent loads an applet, it should support the Java system conventions for loading an assistive technology (refer to the appendix on loading assistive technologies for DOM access). If the user is accessing the applet through an assistive technology, the assistive technology should notify the user when the applet receives content focus as this will likely result in the launch of an associated plug-in or browser-specific Java Virtual Machine. The user agent then needs to turn control of the applet over to the assistive technology. User agents must make available equivalent alternatives to the assistive technology. Applets generally include an application frame that provides title information.

3.11 Input configuration techniques

Many people benefit from direct access to important user agent functionalities (e.g., via a single key stroke or short voice command): users with poor physical control (who might mistakenly repeat a key stroke), users who fatigue easily (for whom key combinations involve significant effort), users who cannot remember key combinations, and any user who wants to operate the user agent quickly.

User agents that allow users to modify default input configurations must account for configuration information from several sources: user agent defaults, user preferences, author-specified configurations, and operating system conventions. In HTML, the author may specify keyboard bindings with the "accesskey" attribute ([HTML4] , section 17.11.2). Users generally specify their preferences through the user interface but may also do so programmatically or through a profile . The user agent may also consider user preferences set at the operating system level.

To the user, the most important information is the final configuration once all sources have been cascaded (combined) and all conflicts resolved. Knowing the default configuration is also important; checkpoint 11.3 requires that the default configuration be documented. The user may also want to know how the current configuration differs from the default configuration and what configuration in the current viewport comes from the author. This information may also be useful to technical support personnel who may be assisting users.

- The user interfaces for viewing and editing the input configuration may be combined, but need not be. When a single interface is available to the user, allow the user to apply filters to the list of bindings (e.g., author-specified only, user agent default, user preference, final configuration, etc.).
- The user interfaces for viewing and editing the input configuration must be

accessible: do not rely on color alone to convey information, use standard controls, allow device-independent input and output, etc.
- In the user interface, associate with each binding a short text description of the function to be activated. For example, if "**`Control-P`**" maps to a print functionality, a short description might be "Print" or "Print setup". For author-specified configurations, use available information (e.g., "title") or use generic descriptions of what action will be triggered (e.g., "Follow the link with this link text").
- Allow users to query user interface controls for pertinent input configuration information (e.g., what key will activate the functionality).

3.11.1 Resolution of input configuration conflicts

In general, user preferences should override other configurations, however this may not always be desirable. For example, users should be prevented from configuring the user agent in a way that would interfere with important functionalities such as quitting the user agent or reconfiguring it.

Some possible options user agents may make available to the user to resolve conflicts include:

- Allow author configurations to override other configurations and alert the user when this happens.
- Do not allow author configurations to override other configurations. Alert the user when an author-specified binding has been overridden and provide access to the author-specified control through other means (e.g., an unused binding, a menu, in a list of all author-specified bindings, etc.)
- Author-specified keyboard bindings in combination with the user agent's trigger mechanism may conflict with system conventions. For example, Internet Explorer [IE-WIN] in Windows uses the **`Alt`** key as the trigger key for author-specified bindings. If the author has specified a configuration with the characters "h" or "f", this will interfere with the system conventions for accessing help and the file menu. In addition to the previous two options for handling conflicts, the user agent may allow the user to choose another trigger key (either globally or on a per-document basis when conflicts are detected).

3.11.2 Invocation through the input configuration

Users may want to use a keyboard or voice binding to shift focus without actually triggering the associated functionality (refer to parallel behavior described for navigation of active elements in the section on sequential navigation techniques). First-time users may want to access additional information before deciding whether to activate a control. More experienced users or those familiar with a page may want to select and activate in one step. Therefore, the user agent may provide the user with the following options:

1. On invocation of the input binding, move focus to the associated active element, but do not activate it.
2. On invocation of the input binding, move focus to the associated active element and prompt the user with information that will allow the user to decide whether to activate the element (e.g., link title or text). Allow the user to suppress future prompts for this particular input binding.
3. On invocation of the input binding, move focus to the associated active element and activate it.

3.12 Speech techniques

The following techniques apply user agents that render content as synthesized speech. For information about speech recognition and accessibility, refer to "Speak to Write" [SPEAK2WRITE] .

- Since user agents that render content as speech do not always pronounce it correctly, they should provide additional context to facilitate understanding. Techniques include:
  - Spelling words
  - Indicating punctuation, capitalization, etc.
  - Allowing users to repeat words alone and in context.
  - Using auditory nuances – including pitch, articulation model, volume, and orientation – to convey meaning the way fonts, spacing, and borders do in graphical media.
  - Generating context. For example, a user agent might speak the word "link" before a link, "heading" before the text content of a heading or "item 1.4" before a list item.
  - Rendering text according in the appropriate natural language .
- User agents that synthesize speech should implement the CSS 2 aural style sheet properties ([CSS2] , section 19) to allow users to configure speech rate, volume, and pitch.
- User agents that provide accessible solutions for images should, by default, provide *no* information about images for which the author has provided no text equivalent , otherwise information may clutter the user's view of the content and cause confusion. The user should be able to turn off this option.
- User agents may recognize different natural languages and be able to render content according to language markup defined for a certain part of the document. For instance, a screen reader might change the pronunciation of spoken text according to the language definition. This is usually desired and done according to the capabilities of the tool. Some specialized tools might give some finer user control for the pronunciation as well. **Note:** A user agent may not support all languages.
- The following techniques for speaking data tables are adapted from the "Tape Recording Manual" produced by the National Braille Association [NBA] :
  1. Read the title, source, captions and any explanatory keys.
  2. Describe the structure of the table. Include the number of columns, the

headers of each column and any associated sub-columns, reading from left to right. The subhead is not considered a column. If column heads have footnotes, read them following each header.
3. Explain whether the table will be read by rows (horizontally) or by columns (vertically). The horizontal reading is usual but, in some cases, the vertical reading better conveys the content. On rare occasions it is necessary to read a table both ways.
4. Repeat the column headers with the figures under them for the first two rows. If the table is long, repeat the headers every fifth row. Always repeat them during the reading of the last row.
5. Indicate the last row by saying, "and finally . . . " or "last row ..."
6. At the completion of the reading say "End table X." If the table appeared on a page other than the one you were recording, add "Returning to text on page Y."

4 Appendix: Accessibility features of some operating systems

Several mainstream operating systems now include built-in accessibility features 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
: StickyKeys allows users who have difficulties with pressing several keys simultaneously to press and release sequentially each key of the configuration.

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

RepeatKeys
: RepeatKeys allows users to set how fast a key repeats ("repeat rate") when the key is held pressed. It also allows users to control how quickly the key starts to repeat after the key has been pressed ("delay until repeat"). Users can also turn off key repeating.

SlowKeys
: SlowKeys instructs the computer not to accept a key as pressed until it has been pressed and held down for more than a user-configurable length of time.

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

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

SoundSentry
: SoundSentry monitors the operating system and applications for sounds in order to provide a graphical indication when a sound is being played. Older versions of SoundSentry 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
: ShowSounds are user settings or software switches that cause audio to be presented using both audio and graphics. Applications may use these switches as the basis of user preferences.

HighContrast
: HighContrast sets fonts and colors designed to make the screen easier to read.

TimeOut
: TimeOut turns off built-in accessibility features automatically if the computer remains idle for a user-configurable length of time. This is useful for computers in public settings such as a 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
: SerialKeys allows 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, infra-red 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

To find out about built-in accessibility features on Windows platforms, ask the system via the "SystemParametersInfo" function. Please refer to "Software accessibility guidelines for Windows applications" [MS-ENABLE] for more information.

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

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

- StickyKeys: modifier keys include **`Shift`**, **`Control`**, and **`Alt`**.
- FilterKeys: grouping term for SlowKeys, RepeatKeys, and BounceKeys.
- MouseKeys
- ToggleKeys
- SoundSentry
- ShowSounds
- Automatic reset: term used for TimeOut
- High Contrast
- SerialKeys

Additional accessibility features available in Windows 98:

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

Accessibility Wizard
: The Accessibility Wizard is a setup tool to assist users with the configuration of system accessibility features.

Apple Macintosh operating system

The following accessibility features can be adjusted from the Easy Access Control panel. **Note:** The Apple naming convention for accessibility features is to put spaces between the terms (e.g., "Sticky Keys" instead of "StickyKeys").

- Sticky Keys: modifier keys include the **`Shift`**, **`Command`** (**`Open apple`**), **`Option`** (**`Alt`**), and **`Control`** keys.
- Slow Keys
- Mouse Keys

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

- Key Repeat Rate (part of RepeatKeys)
- Delay Unit Repeat (part of RepeatKeys)

The following accessibility feature can be adjusted from the Sound or Monitors and Sound Control Panel (depending on system version).

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

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

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

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

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

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 **`Alt`** keys.
- Keyboard Response Group: grouping term for SlowKeys, RepeatKeys, and BounceKeys
- MouseKeys
- ToggleKeys
- SoundSentry (incorrectly named ShowSounds)
- SerialKeys
- TimeOut

5 Appendix: Loading assistive technologies for access to the document object model

Many of the checkpoints in the guidelines require a "host" user agent to communicate information about content and the user interface to assistive technologies. This appendix explains how developers can ensure the timely exchange of this information (refer to checkpoint 5.6). The techniques described here include:

1. Loading the entire assistive technology in the address space of the host user agent;
2. Loading part of the assistive technology in the address space of the host user agent (e.g., piece of stub code, a dynamically linked library (DLL), a browser helper object , etc.);
3. Out-of-process access to the document object model .

The first two techniques are similar, differing in the amount of, or capability of, the assistive technology loaded in the same process or address space as the host user agent. These techniques are likely to provide faster access to the document object model since they will not be subject to inter-process communication overhead.

Note: This appendix does not address specialized user agents that offer assistive technology functions natively (e.g., pwWebSpeak [PWWEBSPEAK]).

Loading assistive technologies for direct navigation of the document object model

First, the host user agent needs to know which assistive technology to load. One technique for this is to store a reference to an assistive technology in a system registry file or, in the case of Java, a properties file. Registry files are common among many operating system platforms:

- Windows: use the system registry file
- IBM OS/2: use the `system.ini`
- On client/server systems: use a system registry server that an application running on the network client computer can query.
- In Sun Java 2, use the "accessibility.properties" file, which 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.

Here is an example entry for Java:

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

In Microsoft 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/assistive technology pair.

Here is an example entry for Windows:

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

Attaching the assistive technologies to the document object model

Once the assistive technology has been registered, any other user agent can determine whether it needs to be loaded and then load it. Once loaded, the assistive technology can monitor the document object model as needed.

On a non-Java platform, a technique to do this would be to create a separate thread with a reference to the document object model using a DLL. This new thread will load the DLL and call a specified DLL entry name with a pointer to the document object model interface. The assistive technology process will then run as long as required.

The assistive technology has the option to either:

- communicate with a main assistive technology of its own and process the document object model as a caching mechanism for the main assistive technology, or
- act as a bridge to the document object model 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 navigation to its client area document object model, but also multiple document object models that it is processing and an event model for monitoring them.

Java's direct access

Java can facilitate timely access to accessibility components. In this example, an assistive technology running on a separate thread monitors user interface events such as focus changes. When focus changes, the assistive technology is notified of which component object has focus. The assistive technology 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 assistive technology has direct access to component specific methods as well as those provided for by the Java Accessibility API. There is no reason that a document object model interface to user agent components could not be provided via Java.

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, `AccessEngine`, 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`, has implemented a document object model interface, the assistive technology will have direct access to the document object model 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 document object model component
            if (o instanceof DOM) {
                ...
            }
        }
        public void focusLost(FocusEvent theEvent) {
            // Do Nothing
        }
    }
```

In this example, the assistive technology has the option of running stand-alone 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 the document object model

In order to attach to a running instance of Internet Explorer 4.0, you can use a Browser Helper Object ([BHO]), which is a DLL that will attach itself to every new instance of Internet Explorer 4.0 [IE-WIN] (only if you explicitly run iexplore.exe). You can use this feature to gain access to the object model of Internet Explorer and to monitor events. This can be tremendously helpful when many method calls need to be made to IE, as each call will be executed 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.

Java access bridge

To provide native Microsoft Windows assistive technologies access to Java applications without creating a Java native solution, Sun Microsystems provides the "Java Access Bridge." This bridge is loaded as an assistive technology as described in the section on loading assistive technologies for direct navigation of the document object model . 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 assistive technology 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 for indirect access to the document object model

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 document object model that might lead a developer to wish to access it from their own process or memory address space. One obvious protection this method provides is that, if the user agent fails, it does not disable the user's assistive technology as well. Another consideration would be legacy systems, where the user relies on their assistive technology for access to software other than the user agent, and thus would have their application loaded all the time.

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

The following example illustrates the use of COM to access the IE object model. This is an example of how to use COM to get a pointer to the `WebBrowser2` module, which in turn enables access to an interface/pointer to the document object, or IE document object model for the content.

```
/* 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 a working example of this method, refer to HelpDB [HELPDB] .

6 Appendix: Glossary

Active element
: An active element is an element with behaviors that may be ***activated*** (or "triggered") either through the user interface or through an API (e.g., by using scripts). Some element instances may be active at times but not at others (e.g., they may be "deactivated" through scripts, or they are only active for a period of time determined by the author). Which elements are active depends on the document language and whether the features are supported by the user agent. In HTML 4.01 [HTML4] documents, for example, active elements include links, image maps, form controls, element instances with a value for the "longdesc" attribute, and element instances with scripts (event handlers) explicitly associated with them (e.g., through the various "on" attributes). Most systems use the content focus to navigate active elements and identify which is to be activated. An active element's behavior may be triggered through any number of mechanisms, including the mouse, keyboard, an API , etc. The effect of activation depends on the element. For instance, when a link is activated, the user agent generally retrieves the linked Web resource . When a form control is activated, it may change state (e.g., check boxes) or may take user input (e.g., a text field). Refer also to the definition of event handler .

Application Programming Interface (API)
: An application programming interface (API) defines how communication may take place between applications.

Assistive technology
: In the context of this document, an assistive technology is a user agent that:

1. relies on services (such as retrieving Web resources , parsing markup, etc.) provided by one or more other "host" user agents. Assistive technologies communicate data and messages with host user agents by using and monitoring APIs .
2. provides services beyond those offered by the host user agents to meet the requirements of a users with disabilities. Additional services include alternative renderings (e.g., as synthesized speech or magnified content), alternative input methods (e.g., voice), additional navigation or orientation mechanisms, content transformations (e.g., to make tables more accessible), etc.

For example, screen reader software is an assistive technology because it relies on browsers or other software to enable Web access, particularly for people with visual and learning disabilities.

Examples of assistive technologies that are important in the context of this document include the following:

- screen magnifiers, which are used by people with visual disabilities to enlarge and change colors on the screen to improve the visual readability of text and images.
- screen readers, which are used by people who are blind or have reading disabilities to read textual information through synthesized speech or Braille

displays.

- speech recognition software, which may be used by people who have some physical disabilities.
- alternative keyboards, which are used by people with certain physical disabilities to simulate the keyboard.
- alternative pointing devices, which are used by people with certain physical disabilities to simulate mouse pointing and button activations.

Beyond this document, assistive technologies consist of software or hardware that has been specifically designed to assist people with disabilities in carrying out daily activities, e.g., wheelchairs, reading machines, devices for grasping, text telephones, vibrating pagers, etc.

Attribute

This document uses the term "attribute" in the XML sense: an element may have a set of attribute specifications (refer to the XML 1.0 specification [XML] section 3).

Audio, Audio object

An audio object is output from an audio viewport .

Audio-only presentation

An audio-only presentation is a presentation consisting exclusively of one or more audio tracks presented concurrently or in series. Examples of an audio-only presentation include a musical performance, a radio-style news broadcast, and a book reading.

Audio track

An audio track is an audio object that is intended as a whole or partial presentation . An audio track can, but does not necessarily correspond to a single audio channel (left or right audio channel).

Auditory description

An auditory description is either a prerecorded human voice or a synthesized voice (recorded or generated dynamically) describing the key visual elements of a movie or animation. The auditory description is synchronized with the audio track of the presentation, usually during natural pauses in the audio track . Auditory descriptions include information about actions, body language, graphics, and scene changes.

Author styles

Authors styles are style property values that come from a document, its associated style sheets, or are generated by the server.

Captions

Captions (or sometimes "closed captions") are text transcripts that are synchronized with other audio or visual tracks. Captions convey information about spoken words and non-spoken sounds such as sound effects. They benefit people who are deaf or hard-of-hearing, and anyone who cannot hear the audio (e.g., someone in a noisy environment). Captions are generally rendered graphically above, below, or superimposed over video. **Note:** Other terms that include the word "caption" may have different meanings in this document. For instance, a "table caption" is a title for the table, often positioned graphically above or below the table. In this document, the intended meaning of "caption" will be clear from context.

Collated text transcript

A collated text transcript is a text equivalent of a movie or animation. More specifically, it is the combination of the text transcript of the audio track and the text equivalent of the visual track. For example, a collated text transcript typically includes segments of spoken dialogue interspersed with text descriptions of the key visual elements of a presentation (actions, body language, graphics, and scene changes). Refer also to the definitions of text transcript and auditory description . Collated text transcripts are essential for individuals who are deaf-blind.

Configure and ***Control***

In the context of this document, both the terms "control" and "configure" share in common the idea of governance such as a user may exercise over interface layout, user agent behavior, rendering style, and other parameters required by this document. Generally, the difference in the terms centers on the idea of *persistence*. When a user makes a change by "controlling" a setting, that change usually does not persist beyond that user session. On the other hand, when a user "configures" a setting, that setting typically persists into later user sessions. Furthermore, the term "control" typically means that the change can be made easily (such as through a keyboard shortcut) and that the results of the change occur immediately, whereas the term "configure" typically means that making the change requires more time and effort (such as making the change via a series of menus leading to a dialog box, via style sheets or scripts, etc.) and that the results of the change may not take effect immediately (e.g., due to time spent reinitializing the system, initiating a new session, rebooting the system). Configuration settings may be stored in a profile . The range and granularity of the changes that can be controlled or configured by the user may depend on system or hardware limitations.

Both configuration and control may apply at different "levels": across Web resources (i.e., at the user agent level, or inherited from the system), to the entirety of a Web resource, or to components of a Web resource (e.g., on a per-element basis). For example, users may configure the user agent to apply the same font family across Web resources, so that all text content is displayed by default using that font family. Or, the user may wish to configure the rendering of a particular element type, which may be done through style sheets. Or, the user may wish to control the text size dynamically (zooming in and out) for a given document, without affecting the Web resource-level configuration. Or, the user may wish to control the text size dynamically for a given element, e.g., by navigating to the element and zooming in on it.

Note: In this document, the noun "control" means "user interface component" or "form component".

Content

In this specification, the term "content" is used in two ways:

1. Content refers to the document object as a whole or in parts. Phrases such as "content type", "text content", and "language of content" refer to this usage. When used in this sense, the term content encompasses equivalent

alternatives . Refer also to the definition of rendered content . and other accessibility information.
2. Content refers to the content of an HTML or XML element, in the sense employed by the XML 1.0 specification ([XML] , section 3.1): "The text between the start-tag and end-tag is called the element's content." Context should indicate that the term content is being used in this sense.

Device-independence
: Device-independence refers to the ability to make use of software with any supported input or output device. User agents should follow operating system conventions and use standard system APIs for input and output.

Document Object, ***Document Object Model***
: The document object is the user agent's representation of data (e.g., a document). This data generally comes from the document source , but may also be generated (from style sheets, scripts, transformations, etc.) or produced as a result of preferences set within the user agent. Some data that is part of the document object is routinely rendered (e.g., in HTML, what appears between the start and end tags of elements and the values of attributes such as "alt", "title", and "summary"). Other parts of the document object are generally processed invisibly by the user agent, such as DTD-defined names of element types and attributes, and other attribute values such as "href", "id", etc. These guidelines require that users have access to both types of data through the user interface.
A document object model is the abstraction that governs the construction of the user agent's document object. The document object model employed by different user agents will vary in implementation and sometimes in scope. Nevertheless, this document calls for developers of user agents to adhere to the ***W3C Document Object Model (DOM)***, which specifies a standard interface for accessing HTML and XML content. This standard interface allows authors to access and modify the document with a scripting language (e.g., JavaScript) in a consistent manner across different scripting languages. As a standard interface, use of a W3C DOM makes it easier not just for authors but for assistive technology developers to extract information and render it in ways most suited to the needs of particular users. The relevant W3C DOM Recommendations are listed in the references . In this specification, the acronym "DOM" refers to the W3C DOM.

Document Source, ***Document Source View***
: In this document, the term document source refers to the data that the user agent receives as the direct result of a request for a Web resource . A document source view represents all or part of a document in a way that exposes the markup language(s) used to build the Web resource. A source view often presents textual representations of content. Refer also to the definition of content .

Documentation
: Documentation refers to **all** information provided by the vendor about a product, including all product manuals, installation instructions, the help system, and tutorials.

Element

This document uses the term "element" both in the XML sense (an element is a syntactic construct as described in the XML 1.0 specification [XML] , section 3) and more generally to mean a type of content (such as video or sound) or a logical construct (such as a header or list).

Equivalent alternatives for content

Since content in some forms is not always accessible to users with disabilities, authors must provide equivalent alternatives for inaccessible content. In the context of this document, the equivalent must fulfill essentially the same function for the person with a disability (at least insofar as is feasible, given the nature of the disability and the state of technology), as the "primary" content does for the person without any disability. For example, the text "The Full Moon" might convey the same information as an image of a full moon when presented to users. Note that equivalent information focuses on fulfilling the same function. If the image is part of a link and understanding the image is crucial to guessing the link target, an equivalent must also give users an idea of the link target. Equivalent alternatives of content include ***text equivalents*** (long and short, synchronized and unsynchronized) and non-text equivalents (e.g., an auditory description , or a visual track that shows a sign language translation of a written text, etc.). Please also consult the Web Content Accessibility Guidelines 1.0 [WCAG10] and its associated Techniques document [WCAG10-TECHS] . Each markup language defines its own mechanisms for specifying equivalent alternatives. For instance, in HTML 4.01 [HTML4] or SMIL 1.0 [SMIL] , the "alt" attribute specifies alternative text for many elements. In HTML 4.01, authors may provide alternatives in attribute values (e.g., the "summary" attribute for the TABLE element), in element content (e.g., OBJECT for external content it specifies, NOFRAMES for frame alternatives, and NOSCRIPT for script alternatives), and in prose.

Events and scripting, event handler

User agents often perform a task when a certain event occurs, caused by user interaction (e.g., mouse motion or a key press), a request from the operating system, etc. Some markup languages allow authors to specify that a script, called an ***event handler***, be executed when a specific event occurs, such as document loading and unloading, mouse press or hover events, keyboard events, and other user interface events. **Note:** The combination of HTML, style sheets, the Document Object Model (DOM) , and scripting is commonly referred to as "Dynamic HTML" or DHTML. However, as there is no W3C specification that formally defines DHTML, this document only refers to event handlers and scripts.

Focus, ***content focus***, ***user interface focus***, ***current focus***

The notion of focus refers to two identifying mechanisms of user agents:

1. The "content focus" designates an active element in a document. A viewport has at most one content focus.
2. The "user interface focus" designates a control of the user interface that will respond to user input (e.g., a radio button, text box, menu, etc.).

The term "focus" encompasses both types of focus. Where one is meant specifically in this document, it is identified.

When several viewports coexist, each may have a content and user interface focus. At all times, only one content focus **or** one user interface focus is active, called the current focus. The current focus responds to user input and may be toggled between content focus and user interface focus through the keyboard, pointing device, etc. Both the content and user interface focus may be highlighted . Refer also to the definition of point of regard .

Graphical

In this document, the term graphical refers to information (text, graphics, colors, etc.) rendered for visual consumption.

Highlight

A highlight mechanism emphasizes selected or focused content. For example, graphical highlight mechanisms include dotted boxes, underlining, and reverse video. Synthesized speech highlight mechanisms include alterations of voice pitch and volume.

Input configuration

An input configuration is the mapping of user agent functionalities to some user interface trigger mechanisms (e.g., menus, buttons, keyboard keys, voice commands, etc.). The default input configuration is the mapping the user finds after installation of the software; it must be included in the user agent documentation .

Multimedia Presentation

For the purposes of this document, a multimedia presentation is a presentation that is not a visual-only presentation , audio-only presentation , or tactile-only presentation . In a "classic" multimedia presentation (e.g., a movie that has sound track or an animation with accompanying audio), at least one visual track is closely synchronized with at least one audio track .

Native support

A user agent supports a feature natively if it does not require another piece of software (e.g., plug-in or external program) for support. Operating system features adopted by the user agent to meet the requirements of this document are considered part of native support. User agents may, but are not required to, provide access to adopted operating system features through the user agent's user interface or programmatic means. For example, if the user agent relies on the operating system's audio control features to meet some requirements of this document, the user agent is not required to include those controls in its native user interface. If an adopted operating system feature is not accessible through the operating system's user interface, then the user agent must provide an alternative accessible solution.

Natural language

Natural language is spoken, written, or signed human language such as French, Japanese, and American Sign Language. On the Web, the natural language of content may be specified by markup or HTTP headers. Some examples include the "lang" attribute in HTML 4.01 ([HTML4] section 8.1), the "xml:lang" attribute in XML 1.0 ([XML] , section 2.12), the HTML 4.01 "hreflang" attribute for links in HTML 4.01 ([HTML4] , section 12.1.5), the HTTP Content-Language header

([RFC2616] , section 14.12) and the Accept-Language request header ([RFC2616] , section 14.4).

Point of regard

The point of regard of a viewport is its position in rendered content . What is meant precisely by "the point of regard" may vary since users may be viewing rendered content with browsers that render in various ways (graphically , as speech, as Braille, etc.). Depending on the user agent and browsing context, it may refer to a two dimensional area (e.g., for graphical rendering) or a single point (e.g., for aural rendering or voice browsing). The point of regard may also refer to a particular moment in time for content that changes over time (e.g., an audio-only presentation). User agents may use the focus , selection , or other means to designate the point of regard. A user agent should not change the point of regard unexpectedly as this may disorient the user.

Presentation

In this document, the term presentation refers to a collection of information, consisting of one or more Web resources , intended to be rendered simultaneously, and identified by a single URI. In general, a presentation has an inherent time component (i.e., it’s not just a static "Web page" (refer to the definition of "Web page" in "Web Characterization Terminology and Definitions Sheet" [WEBCHAR]).

Profile

A profile is a named and persistent representation of user preferences that may be used to configure a user agent. Preferences include input configurations, style preferences, etc. On systems with distinct user accounts, profiles enable users to reconfigure software quickly when they log on, and they may be shared by several users. Platform-independent profiles are useful for those who use the same user agent on different platforms.

Properties, values, and defaults

A user agent renders a document by applying formatting algorithms and style information to the document’s elements. Formatting depends on a number of factors, including where the document is rendered: on screen, on paper, through speakers, on a Braille display, on a mobile device, etc. Style information (e.g., fonts, colors, voice inflection, etc.) may come from the elements themselves (e.g., certain font and phrase elements in HTML), from style sheets, or from user agent settings. For the purposes of these guidelines, each formatting or style option is governed by a property and each property may take one value from a set of legal values. Generally in this document, the term "property" has the meaning defined in CSS 2 ([CSS2] , section 3). A reference to "styles" in this document means a set of style-related properties.

The value given to a property by a user agent when it is installed is called the property’s ***default value***.

Recognize

A user agent is said to recognize markup, content types, or rendering effects when it can identify the information. Recognition may occur through built-in mechanisms, Document Type Definitions (DTDs) style sheets, HTTP headers, and other means. An example of failure of recognition is that HTML 3.2 user agents may not recognize the new elements or attributes of HTML 4.01 [HTML4] . While a user agent may recognize blinking content specified by elements or attributes, it may not recognize blinking in an applet. The Techniques document [UAAG10-TECHS] lists some markup known to affect accessibility that should be recognized by user agents.

Rendered content

The rendered content is that part of content rendered in a given viewport (whether visual, audio, or tactile).

Selection*, *current selection

The selection generally identifies a range of content (e.g., text, images, etc.) in a document. The selection may be structured (based on the document tree) or unstructured (e.g., text-based). Content may be selected through user interaction, scripts, etc. The selection may be used for a variety of purposes: for cut and paste operations, to designate a specific element in a document, to identify what a screen reader should read, etc.

The selection may be set by the user (e.g., by a pointing device or the keyboard) or through an application programming interface (API). A viewport has at most one selection (though the selection may be rendered graphically as discontinuous text fragments). When several viewports coexist, each may have a selection, but only one is active, called the current selection.

On the screen, the selection may be highlighted using colors, fonts, graphics, magnification, etc. The selection may also be rendered as inflected speech, for example.

Standard device APIs

Operating systems are designed to be used by default with devices such as pointing devices, keyboards, voice input, etc. The operating system (or windowing system) provides "standard APIs " for these devices. On desktop computers today, the standard input APIs are for the mouse and keyboard. For touch screen devices or mobile devices, standard input APIs may include stylus, buttons, voice, etc. The graphical display and sound card are considered standard ouput devices for a graphical desktop computer environment, and each has a standard API.

Synchronize

In this document, the term synchronize has two meanings:

1. The time-coordination of two or more presentation components (e.g., in a multimedia presentation, a visual track with captions). For Web content developers, the requirement to synchronize means to provide the data that will permit sensible time-coordinated rendering by a user agent. For example, Web content developer can ensure that the segments of caption text are neither too long nor too short, and that they map to segments of the visual track that are appropriate in length. For user agent developers, the requirement to synchronize means to present the content in a sensible time-coordinated fashion under a wide range of circumstances including technology constraints (e.g., small text-only displays), user limitations (slow reading speeds, large font sizes, high need for review or repeat functions), and content that is sub-optimal in terms of accessibility.

2. The coordination of user interface changes (e.g., focus changes) among two or more viewports .

Text transcript
A text transcript is a text equivalent of audio information (e.g., an audio-only presentation or the audio track of a movie or animation). It provides text for both spoken words and non-spoken sounds such as sound effects. Text transcripts make audio information accessible to people who have hearing disabilities and to people who cannot play the audio. Text transcripts are usually pre-written but may be generated on the fly (e.g., by speech-to-text converters). Refer also to the definitions of captions and collated text transcripts .

User agent
A user agent is software that retrieves and renders Web content, including text, graphics, sounds, video, images, and other content types. A user agent may require additional user agents that handle some types of content. For instance, a browser may run a separate program or plug-in to render sound or video. User agents include graphical desktop browsers, multimedia players, text browsers, voice browsers, and assistive technologies such as screen readers, screen magnifiers, speech synthesizers, onscreen keyboards, and voice input software.

Tactile object
A tactile object is output from a tactile viewport Tactile objects include text (rendered as Braille) and graphics (rendered as raised-line drawings).

Tactile-only presentation
A tactile-only presentation is a presentation consisting exclusively of one or more tactile tracks presented concurrently or in series.

Tactile track
A tactile track is a tactile object that is intended as a whole or partial presentation . This does not necessarily correspond to a single physical or logical track on the storage or delivery media.

User agent default styles
User agent default styles are style property values applied in the absence of any author or user styles. Some markup languages specify a default rendering for documents in that markup language. Other specifications may not specify default styles. For example, XML 1.0 [XML] does not specify default styles for XML documents. HTML 4 [HTML4] does not specify default styles for HTML documents, but the CSS 2 [CSS2] specification suggests a sample default style sheet for HTML 4 based on current practice.

User interface
For the purposes of this document, user interface includes both:

1. the "***user agent user interface***", i.e., the controls and mechanisms offered by the user agent for user interaction, such as menus, buttons, keyboard access, etc.
2. the "content user interface", i.e., the active elements that are part of content, such as form controls, links, applets, etc. that are implemented natively .

The document distinguishes them only where required for clarity.

User styles

User styles are style property values that come from user interface settings, user style sheets, or other user interactions.

User-initiated, user agent initiated

An action initiated by the user is one that results from user operation of the user interface. An action initiated by the user agent is one that results from the execution of a script (e.g., an event handler bound to an event not triggered through the user interface), from operating system conditions, or from built-in user agent behavior.

Visual object

A visual object is output from a visual viewport . Visual objects include graphics, text, and visual portions of movies and animations.

Visual-only presentation

A visual-only presentation is a presentation consisting exclusively of one or more visual tracks presented concurrently or in series.

Visual track

A visual track is a visual object that is intended as a whole or partial presentation . A visual track does not necessarily correspond to a single physical or software object. A visual track can be text-based or graphic, static or animated.

Views, viewports, and current viewport

User agents may handle different types of content : markup language, sound, video, etc. The user views rendered content through a ***viewport***, which may be a window, a frame, a piece of paper, a speaker, a virtual magnifying glass, etc. A viewport may contain another viewport (e.g., nested frames). Viewports do not include user interface controls such as prompts, menus, alerts, etc.

The viewport that contains both the current focus and the current selection is called the ***current viewport***. The current viewport is generally highlighted when several viewports coexist. A user agent should provide mechanisms for accessing all content that can be presented by each viewport (e.g., scrolling mechanisms, advance and rewind, etc.).

User agents may render the same content in a variety of ways; each rendering is called a ***view***. For instance, a user agent may allow users to view an entire document or just a list of the document's headers. These are two different views of the document.

Web resource

The term "Web resource" is used in this document in accordance with Web Characterization Terminology and Definitions Sheet [WEBCHAR] to mean anything that has identity on the Web. A Web resource is identified by a URI.

7 Acknowledgments

The active participants of the User Agent Guidelines Working Group who produced this document were: James Allan, Denis Anson, Kitch Barnicle, Harvey Bingham, Dick Brown, Al Gilman, Jon Gunderson, Eric Hansen, Ian Jacobs, Marja-Riitta Koivunen, Tim Lacy, Charles McCathieNevile, Mark Novak, David Poehlman, Mickey Quenzer, Gregory Rosmaita, Madeleine Rothberg, and Rich Schwerdtfeger.

Many thanks to the following people who have contributed through review and past participation: Paul Adelson, 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, Larry Goldberg, Glen Gordon, John Grotting, Markku Hakkinen, Earle Harrison, Chris Hasser, Kathy Hewitt, Philipp Hoschka, Masayasu Ishikawa, Phill Jenkins, Earl Johnson, Jan Kärrman (for help with html2ps), Leonard Kasday, George Kerscher, Peter Korn, Josh Krieger, Catherine Laws, Greg Lowney, Susan Lesch, Scott Luebking, William Loughborough, Napoleon Maou, Peter Meijer, Karen Moses, Masafumi Nakane, Charles Oppermann, Mike Paciello, David Pawson, Michael Pederson, Helen Petrie, Michael Pieper, Jan Richards, Hans Riesebos, Joe Roeder, Lakespur L. Roca, Lloyd Rutledge, Liam Quinn, T.V. Raman, Robert Savellis, 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.

8 References

For the latest version of any W3C specification please consult the list of W3C Technical Reports at http://www.w3.org/TR.

[AERT]
"Techniques For Accessibility Evaluation And Repair Tools," C. Ridpath, W. Chisholm, eds. This is a W3C Working Draft as of publication of the current document. The latest version of this document is available at http://www.w3.org/TR/AERT.

[ATAG10]
"Authoring Tool Accessibility Guidelines 1.0", J. Treviranus, C. McCathieNevile, I. Jacobs, and J. Richards, eds., 3 February 2000. This ATAG 1.0 Recommendation is http://www.w3.org/TR/2000/REC-ATAG10-20000203.

[ATAG10-TECHS]
"Techniques for Authoring Tool Accessibility Guidelines 1.0", J. Treviranus, C. McCathieNevile,, I. Jacobs, and J. Richards, eds. The latest version of this techniques document is http://www.w3.org/TR/ATAG10-TECHS/.

[CHARMOD]
"Character Model for the World Wide Web", M. Dürst, 25 February 1999. This W3C Working Draft is http://www.w3.org/TR/1999/WD-charmod-19990225.

[CSS-ACCESS]
"Accessibility Features of CSS", I. Jacobs, J. Brewer, 4 August 1999. This W3C note is http://www.w3.org/1999/08/NOTE-CSS-access-19990804. The latest version of the Accessibility Features of CSS is available at http://www.w3.org/TR/CSS-access.

[CSS1]
"CSS, level 1 Recommendation", B. Bos, H. Wium Lie, eds., 17 December 1996, revised 11 January 1999. This CSS 1 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., 12 May 1998. This CSS 2 Recommendation is http://www.w3.org/TR/1998/REC-CSS2-19980512.

[DOM2]
"Document Object Model (DOM) Level 2 Specification", L. Wood, A. Le Hors, V. Apparao, L. Cable, M. Champion, J. Kesselman, P. Le Hégaret, T. Pixley, J. Robie, P. Sharpe, C. Wilson, eds. The latest version of the specification is available at: http://www.w3.org/TR/DOM-Level-2.

[HTML4]
"HTML 4.01 Recommendation", D. Raggett, A. Le Hors, and I. Jacobs, eds., 24 December 1999. This HTML 4.01 Recommendation is http://www.w3.org/TR/1999/REC-html401-19991224.

[MATHML]
"Mathematical Markup Language", P. Ion and R. Miner, eds., 7 April 1998. This MathML 1.0 Recommendation is

http://www.w3.org/TR/1998/REC-MathML-19980407.

[MICROPAYMENT]
"Common Markup for micropayment per-fee-links", T. Michel, ed. The latest version of this W3C Working Draft is available at http://www.w3.org/TR/Micropayment-Markup.

[PNG]
"PNG (Portable Network Graphics) Specification 1.0", T. Boutell, ed., 1 October 1996. This W3C Recommendation is http://www.w3.org/TR/REC-png.

[RFC2616]
"Hypertext Transfer Protocol -- HTTP/1.1, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999.

[SMIL]
"Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", P. Hoschka, ed., 15 June 1998. This SMIL 1.0 Recommendation is http://www.w3.org/TR/1998/REC-smil-19980615.

[SMIL-ACCESS]
"Accessibility Features of SMIL", M-R. Koivunen, I. Jacobs, 21 September 1999. This W3C note is http://www.w3.org/TR/1999/NOTE-SMIL-access-19990921. The latest version of the Accessibility Features of SMIL is available at http://www.w3.org/TR/SMIL-access.

[UAAG10]
"Techniques for User Agent Accessibility Guidelines 1.0," J. Gunderson, I. Jacobs, eds. The latest draft of the guidelines is available at http://www.w3.org/WAI/UA/UAAG10-TECHS/.

[UAAG10-TECHS]
"Techniques for User Agent Accessibility Guidelines 1.0," J. Gunderson, I. Jacobs, eds. The latest draft of the techniques document is available at http://www.w3.org/WAI/UA/UAAG10-TECHS/.

[WCAG10]
"Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, and I. Jacobs, eds., 5 May 1999. This WCAG 1.0 Recommendation is http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505.

[WCAG10-TECHS]
"Techniques for Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, and I. Jacobs, eds. The latest version of this document is available at http://www.w3.org/TR/WCAG10-TECHS.

[WEBCHAR]
"Web Characterization Terminology and Definitions Sheet", B. Lavoie, H. F. Nielsen, eds., 24 May 1999. This is a W3C Working Draft that defines some terms to establish a common understanding about key Web concepts.

[XHTML10]
"XHTML[tm] 1.0: The Extensible HyperText Markup Language", S. Pemberton, et al. The 26 January 2000 XHTML 1.0 Recommendation is http://www.w3.org/TR/2000/REC-xhtml1-20000126.

[XML]
"Extensible Markup Language (XML) 1.0.", T. Bray, J. Paoli, C.M.

Sperberg-McQueen, eds., 10 February 1998. This XML 1.0 Recommendation is http://www.w3.org/TR/1998/REC-xml-19980210.

[XSLT]
"XSL Transformations (XSLT) Version 1.0", J. Clark. The 16 November 1999 Recommendation is http://www.w3.org/TR/1999/REC-xslt-19991116.

9 Resources

Note: *W3C does not guarantee the stability of any of the following references outside of its control. These references are included for convenience. References to products are not endorsements of those products.*

9.1 Operating system and programming guidelines

[APPLE-HI]
Refer to the following guidelines from Apple:

- Information on accessibility guidelines for Macintosh applications.
- Inside Macintosh: Macintosh Human Interface Guidelines / Part 1 - Fundamentals Chapter 2 – General Design Considerations (Very General).
- Inside Macintosh: Mac OS 8 Control Manager Reference / Addresses Keyboard Focus.
- Inside Macintosh: Mac OS 8 Human Interface Guidelines / Chapter 3 - Dialog Box Guidelines / Keyboard Navigation and Focus.
- Inside Macintosh: Programmer's Guide to MacApp / Part 1 – MacApp Theory and Architecture / Chapter 8 – Displaying, Manipulating, and Printing Data / Cursor Handling
- Inside Macintosh: Programmer's Guide to MacApp / Part 1 – MacApp Theory and Architecture / Chapter 8 – Displaying, Manipulating, and Printing Data / Basic View TechnologyHighlighting in a View
- Inside Macintosh: Macintosh Human Interface Guidelines / Part 2 – The Interface Elements / Chapter 10 – Behaviors / Selecting
- Inside Macintosh: Imaging with QuickDraw / Highlighting
- Information on Apple's scripting model can be found at tn1095 and tn1164. Refer also to the Inside Macintosh chapter devoted to Inter-application Communication.

[BHO]
Browser Helper Objects: The Browser the Way You Want It, D. Esposito, January 1999. Refer also to http://support.microsoft.com/support/kb/articles/Q179/2/30.asp.

[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.. Refer to the IBM guidelines for software accessibility, IBM guidelines for Java accessibility.

[ICCCM]
"The Inter-Client communication conventions manual". A protocol for communication between clients in the X Window system. The file `icccm.ms` is a troff-formatted document.

[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.

[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.

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

[MOTIF]
The OSF/Motif Style Guide.

[MS-ENABLE]
Software accessibility guidelines for Windows applications. Refer also to Built-in accessibility features.

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

[MS-SOFTWARE]
"The Microsoft Windows Guidelines for Accessible Software Design". **Note:** This page summarizes the guidelines and includes links to the full guidelines in various formats (including plain text).

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

[NISO]
National Information Standards Organization. One activity pursued by this organization concerns Digital Talking Books. Refer to the "Digital Talking Book Features List" draft for more information.

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

[PHOTO-RDF]
"Describing and retrieving photos using RDF and HTTP", Y. Lafon and B. Bos. The 3 May 2000 version of the W3C Note is http://www.w3.org/TR/2000/NOTE-photo-rdf-20000503.

[SAMI]
Information on Synchronized Accessible Multimedia Interchange (SAMI) accessibility.

[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.

9.2 User agents and other tools

A list of alternative Web browsers (assistive technologies and other user agents designed for accessibility) is maintained at the WAI Web site.

[ALTIFIER]
The Altifier Tool generates "alt" text intelligently.

[AMAYA]
Amaya is W3C’s test-bed browser and editor.

[AWB]
The Accessible Web Browser senior project at the University of Illinois Champaign-Urbana.

[CSSVALIDATOR]
W3C’s CSS Validator service.

[G2]
The G2 player version 7 for Windows.

[HELPDB]
HelpDB is a test tool for Web table navigation.

[HPR]
Home Page Reader.

[IE-WIN]
Internet Explorer 5.0 for Windows 95, Windows 98, and Windows NT. Refer also to information on using COM with IE. Refer also to information about monitoring HTML events in the IE document object model.

[JAVAWEBLET]
Java Weblets are Java client programs that access the user agent's document object model.

[JFW]
JAWS for Windows.

[LYNX]
The Lynx Browser.

[MOZILLA]
The Mozilla browser.

[NAVIGATOR]
Netscape Navigator.

[OPERA]
The Opera Browser.

[PWWEBSPEAK]
pwWebSpeak.

[QUICKTIME]
The Quicktime player.

[TABLENAV]
A table navigation script from the Trace Research Center.

[VALIDATOR]
W3C's HTML/XML Validator service.

[WINDOWEYES]
Window-Eyes.

[WINVISION]
Winvision.

9.3 Accessibility resources

[BRAILLEFORMATS]
"Braille Formats: Principles of Print to Braille Transcription 1997" .

[NBA]
The National Braille Association.

[NBP]
The National Braille Press.

[RFBD]
Recording for the Blind and Dyslexic.

[SAPI]
Microsoft's Speech Application Programming Interface.

[SPEAK2WRITE]
Speak to Write is a site about using speech recognition to promote accessibility.

9.4 Standards resources

[ISO639]
"Codes for the representation of names of languages", ISO 639:1988. For more information, consult http://www.iso.ch/cate/d4766.html. Refer also to http://www.oasis-open.org/cover/iso639a.html.

[UNICODE]
The Unicode Consortium. "The Unicode Standard, Version 3.0", Reading, MA, Addison-Wesley Developers Press, 2000. ISBN 0-201-61633-5. Refer also to http://www.unicode.org/unicode/standard/versions/.