Techniques for User Agent Accessibility Guidelines 1.0

W3C Working Draft, 12 September 2001

This version:
http://www.w3.org/TR/2001/WD-UAAG10-TECHS-20010912/
(Formats: single HTML, plain text, gzip PostScript, gzip PDF, gzip tar file of HTML, zip archive of HTML)
Latest version:
http://www.w3.org/TR/UAAG10-TECHS/
Previous version:
http://www.w3.org/TR/2001/WD-UAAG10-TECHS-20010622/

Editors:
Ian Jacobs, W3C
Jon Gunderson, University of Illinois at Urbana-Champaign
Eric Hansen, Educational Testing Service
Authors and Contributors:
See acknowledgements .

Copyright © 1999 - 2001 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This document provides techniques for satisfying the checkpoints defined in "User Agent Accessibility Guidelines 1.0" *[UAAG10]* . These techniques address key aspects of 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).

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 a definitive set of requirements for satisfying a checkpoint.

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 12 September 2001 Working Draft of "Techniques for User Agent Accessibility Guidelines 1.0". 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 User Agent Accessibility Guidelines Working Group (UAWG).

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

A list of changes to this document is available.

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 Accessibility Guidelines Working Group are described in the charter.

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

Table of contents

Abstract 1
Status of this document 2
1 The user agent accessibility guidelines 6
1. Support input and output device-independence. 7
2. Ensure user access to all content. 11
3. Allow configuration not to render some content that may reduce accessibility. 27
4. Ensure user control of rendering. 34
5. Ensure user control of user interface behavior. 50
6. Implement interoperable application programming interfaces. . . 55
7. Observe operating environment conventions. 65
8. Implement specifications that benefit accessibility. 69
9. Provide navigation mechanisms. 72
10. Orient the user. 84
11. Allow configuration and customization. 98
12. Provide accessible user agent documentation and help. . . . 104
2 Accessibility topics 109
2.1 Access to content 109
2.2 User control of rendering and style 111
2.3 Link techniques 111
2.4 List techniques 113
2.5 Table techniques 114
2.6 Image map techniques 118
2.7 Frame techniques 119
2.8 Form techniques 125
2.9 Generated content techniques 128
2.10 Content repair techniques 129
2.11 Script and applet techniques 129
2.12 Input configuration techniques 130
2.13 Synthesized speech techniques 132
2.14 Techniques for reducing dependency on spatial interactions . . 133
2.15 Accessibility and internationalization techniques 134
2.16 Appendix: Impact matrix 134
2.17 Appendix: Accessibility features of some operating systems . . 135
2.18 Appendix: Loading assistive technologies for access to the document object model 139
3 Glossary 145
4 References 163
4.1 How to refer to this document 163
4.2 Normative references 164
4.3 Informative references 164

5 Resources 167
5.1 Operating system and programming guidelines 167
5.2 User agents and other tools 170
5.3 Accessibility resources 171
5.4 Standards resources 171
6 Acknowledgments 172

Related resources

"Techniques for User Agent Accessibility Guidelines 1.0" and the "User Agent Accessibility Guidelines 1.0" *[UAAG10]* are part of a series of accessibility guidelines published by the Web Accessibility Initiative (WAI). These documents explain the responsibilities of user agent developers in making the Web accessibility to users with disabilities. The series also includes the "Web Content Accessibility Guidelines 1.0" *[WCAG10]* (and techniques *[WCAG10-TECHS]*), which explain the responsibilities of authors, and the "Authoring Tool Accessibility Guidelines 1.0" *[ATAG10]* (and techniques *[ATAG10-TECHS]*), which explain the responsibilities of authoring tool developers.

The Web Accessibility Initiative provides other resources and educational materials to promote Web accessibility. Resources include information about accessibility policies, links to translations of WAI materials into languages other than English, information about specialized user agents and other tools, accessibility training resources, and more.

Differences from User Agent Accessibility Guidelines 1.0

In an effort to improve the readability of this document, some information from User Agent Accessibility Guidelines 1.0 has been copied here:

- each checkpoint (including the Notes following the checkpoints);
- the priority definitions ;
- the glossary ;
- the acknowledgments .

In an effort to reduce the size of the current document, some information that is in User Agent Accessibility Guidelines 1.0 has not been copied here:

- the introduction;
- the descriptions of how the guidelines and checkpoints are structured and organized;
- the prose of each guideline (i.e., the text after the guideline title and before the list of checkpoints);
- the conformance section (since one does not conform to the current document, only to User Agent Accessibility Guidelines 1.0).

The current document includes more (implementation-related) references than the same section inUser Agent Accessibility Guidelines 1.0, and includes an additional section on resources that should help implementors.

1 The user agent accessibility guidelines

This section lists each checkpoint of "User Agent Accessibility Guidelines 1.0" *[UAAG10]* along with some possible techniques for satisfying it. Each checkpoint definition includes a link to the checkpoint definition in "User Agent Accessibility Guidelines 1.0". Each checkpoint definition is followed by one or more of the following:

- **Notes and rationale**: Additional rationale and explanation of the checkpoint;
- **Who benefits**: Which users with disabilities are expected to benefit from user agents that satisfy the checkpoint;
- **Example techniques**: Some techniques to illustrate how a user agent might satisfy the requirements of the checkpoint Screen shots and other information about deployed user agents have been included as sample techniques. References to products are not endorsements of those products by W3C;
- **Doing more**: Techniques to achieve more than what is required by the checkpoint;
- **Related techniques**: Links to other techniques in section 3. The accessibility topics of section 3 generally apply to more than one checkpoint.
- **References**: References to other guidelines, specifications, or resources.

Note: Most of the techniques in this document are designed for graphical browsers and multimedia players running on desktop computers. 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.

Priorities

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

Priority 1 (P1)
: 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 (P2)
: 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 (P3)
: 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.

Note: This information about checkpoint priorities is included for convenience only. For detailed information about conformance to "User Agent Accessibility Guidelines 1.0" *[UAAG10]* , please refer to that document.

Guideline 1. Support input and output device-independence.

Checkpoints

1.1 Full keyboard access. (P1)

1. Ensure that the user can operate through keyboard input alone any user agent functionality available through the user interface .

For both content and user agent.

Note: User agents may support at least two types of keyboard access to functionalities: direct access (where user awareness of a location "in space" is not required, as is the case with keyboard shortcuts and navigation of user agent menus) and spatial access (where the user moves the pointing device "in space" via the keyboard). To satisfy this checkpoint, user agents are expected to provide a mix of both types of keyboard access. User agents should allow direct keyboard access where possible, and this may be redundant with spatial input techniques. Furthermore, the user agent should satisfy this requirement by offering a combination of keyboard-operable user interface controls (e.g., keyboard operable print menus and settings) and direct keyboard operation of user agent functionalities (e.g., a short cut to print the current page). As examples of functionalities, ensure that the user can interact with enabled elements , select content, navigate viewports, configure the user agent, access documentation, install the user agent, operate controls of the user interface, etc., all entirely through keyboard input. It is also possible to claim conformance to User Agent Accessibility Guidelines 1.0 *[UAAG10]* for full support through pointing device input and voice input. See the section on input modality labels in UAAG 1.0.

Notes and rationale:

1. It is up to the user agent developer to decide which functionalities are best served by direct keyboard access and which are best served by spatial access through the keyboard (or pointing device). The UAAG 1.0 does not discourage a pointing device interface, but it does require redundancy through the keyboard. In most cases, developers can allow operation of the user agent without relying on motion "through space"; this includes text selection (a text caret may be used to establish the start and end of the selection), region selection (allow the user to describe the coordinates or position of the region, e.g., relative to the viewport), drag-and-drop (allow the user to designate start and end points and then say "go"), etc.
2. For instance, the user must be able to do the following through the keyboard alone (or pointing device alone or voice alone):
   - Select content and operate on it. For example, if the user can select

rendered text with the mouse and make it the content of a new link by pushing a button, they also need to be able to do so through the keyboard and other supported devices. Other operations include cut, copy, and paste.
- Set the focus on viewports and on enabled elements.
- Install, configure, uninstall, and update the user agent software.
- Use the graphical user interface menus. Some users may wish to user the graphical user interface even if they cannot use or do not wish to use the pointing device.
- Fill out forms.
- Access documentation.

3. Suppose a user agent does not allow *complete* operation through the keyboard alone. It is still possible to claim conformance for the user agent in conjunction with a special module designed to "fill in the gap".

Who benefits:

1. Users with blindness are most likely to benefit from direct access through the keyboard, including navigation of user interface controls; this is a logical navigation, not a spatial navigation.
2. Users with physical disabilities are most likely to benefit from a combination of direct access and spatial access through the keyboard. For some users with physical disabilities, moving the pointing device through a physical mouse may be significantly more difficult than moving the pointing device with arrow keys, for example.
3. This checkpoint will also benefit users of many other alternative input devices (which make use of the keyboard API) and also anyone without a mouse.
4. While keyboard operation is expected to improve access for many users, operation by keyboard shortcuts alone may reduce accessibility (and usability) by requiring users to memorize a long list of shortcuts. Developers should provide mechanisms for contextual access to user agent functionalities (including keyboard-operable cascading mechanisms, context-sensitive help, keyboard operable configuration tabs, etc.) as well as direct access to those functionalities. See also checkpoint 11.5.

1.2 Activate event handlers. (P1)

1. For the element with content focus , allow the user to activate any explicitly associated input device event handlers through keyboard input alone.
2. The user agent is not required to allow activation of event handlers associated with a given device (e.g., the pointing device) in any order other than what the device itself allows.

Note: The requirements for this checkpoint refer to **any** explicitly associated input device event handlers associated with an element, independent of the input modalities for which the user agent conforms. For example, suppose that an element has an explicitly associated handler for pointing device events. Even when the user agent only conforms for keyboard input (and does not conform for the pointing device, for example), this checkpoint requires the user agent to allow the user to activate that handler with the keyboard. This checkpoint is an important special case of checkpoint 1.1. Please refer to the checkpoints of guideline 9 for more information about focus requirements.

Notes and rationale:

1. For example, users without a pointing device need to be able to activate form controls and links (including the links in a client-side image map).
2. Events triggered by a particular device generally follow a set pattern, and often in pairs: start/end, down/up, in/out. One would not expect a "key down" event for a given key to be followed by another "key down" event without an intervening "key up" event.

Who benefits:

1. Users with blindness or some users with a physical disability, and anyone without a pointing device.

Example techniques:

1. To preserve the expected order of events, provide a dynamically changing menu of available handlers. For example, an initial menu of handlers might only allow the user to trigger a "mousedown" event. Once triggered, the menu would not allow "mousedown" but would allow "mouseup" and "mouseover", etc.
2. For example, in HTML 4 *[HTML4]* , input device event handlers are described in section 18.2.3. They are: `onclick`, `ondblclick`, `onmousedown`, `onmouseover`, `onmouseout`, `onfocus`, `onblur`, `onkeypress`, `onkeydown`, and `onkeyup`.
3. In "Document Object Model (DOM) Level 2 Events Specification" *[DOM2EVENTS]* , focus and activation types are discussed in section 1.6.1. They are: `DOMFocusIn`, `DOMFocusOut`, and `DOMActivate`. These events are specified independent of a particular input device type.
4. In "Document Object Model (DOM) Level 2 Events Specification" *[DOM2EVENTS]* , mouse event types are discussed in section 1.6.2. They are: `click`, `mousedown`, `mouseup`, `mouseover`, `mousemove` and `mouseout`.
5. The DOM Level 2 Event specification does not provide a key event module.
6. Sequential technique: Add each input device event handler to the serial navigation order (refer to checkpoint 9.3). Alert the user when the user has navigated to an event handler, and allow activation. For example, an link that also has a onMouseOver and onMouseOut event handlers defined, might generate three "stops" in the navigation order: one for the link and two for the

event handlers. If this technique is used, allow configuration so that input device event handlers are not inserted in the navigation order.
7. Query technique: Allow the user to query the element with content focus for a menu of input device event handlers.
8. Descriptive information about handlers can allow assistive technologies to choose 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. See also checkpoint 6.3.
9. Using MSAA *[MSAA]* on the Windows platform:
   - Retrieve the node in the document object that has current focus.
   - Call the `IHTMLDocument4::fireEvent` method on that node.

Related techniques:

1. See image map techniques .

References:

1. For example, section 16.5 of the SVG 1.0 Candidate Recommendation *[SVG]* specifies processing order for user interface events.

1.3 Provide text messages. (P1)

1. Ensure that every message (e.g., prompt , alert , notification, etc.) that is a non-text element and is part of the user agent user interface has a text equivalent .

Note: For example, if the user is alerted of an event by an audio cue, a visually-rendered text equivalent in the status bar could satisfy this checkpoint. Per checkpoint 6.4, a text equivalent for each such message must be available through an API . See also checkpoint 6.5 for requirements for programmatic alert of changes to the user interface.

Notes and rationale:

1. User agents should use modality-specific messages in the user interface (e.g., graphical scroll bars, beeps, and flashes) as long as redundant mechanisms are available or possible. These redundant mechanisms will benefit all users, not just users with disabilities.

Who benefits:

1. Users with blindness, deafness, or who are hard of hearing. Mechanisms that are redundant to audio will benefit 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.

Example techniques:

1. Render text messages on the status bar of the graphical user interface. Allow users to query the viewport for this status information (in addition to having access through graphical rendering).
2. Make available information in a manner that allows other software to present it according to the user's preferences. For instance, if the graphical user agent uses proportional scroll bars to indicate the position of the viewport in content, make available this same information in text form. For instance, this will allow other software to render the proportion of content viewed as synthesized speech or as braille.

Doing more:

1. Allow configuration to render or not render status information (e.g., allow the user to hide the status bar).

Guideline 2. Ensure user access to all content.

Checkpoints

2.1 Render content according to specification. (P1)

1. Render content according to format specification (e.g., for a markup language or style sheet).
2. When a rendering requirement of another specification contradicts a requirement of the current document, the user agent may disregard the rendering requirement of the other specification and still satisfy this checkpoint.
3. Rendering requirements include format-defined interactions between author preferences and user preferences/capabilities (e.g., when to render the "`alt`" attribute in HTML, the rendering order of nested `OBJECT` elements in HTML, test attributes in SMIL, and the cascade in CSS2).

Note: If a conforming user agent does not render a content type, it should allow the user to choose a way to handle that content (e.g., by launching another application, by saving it to disk, etc.). The user agent is not required to satisfy this checkpoint for all implemented specifications; see the section on conformance and implementing specifications for more information.

Notes and rationale:

1. The right to disregard only applies when the rendering requirement of another specification contradicts the requirements of the current document; no exemption is granted if the other specification is consistent with or silent about a requirement made by the current document.

Who benefits:

1. Users with disabilities when specifications include features that promote accessibility (e.g., scalable graphics benefit users with low vision, style sheets allow users to override author and user style sheets).

Example techniques:

1. Provide access to attribute values (one at a time, not as a group). For instance, allow the user to select an 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 view of the text source.
2. When content changes dynamically (e.g., due to embedded scripts or content refresh), users need to have access to the content before and after the change.
3. 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 expansion in one location, look for another occurrence in content that does. User agents may also look for possible expansions (e.g., in parentheses) in surrounding context, though that is a less reliable technique.

Related techniques:

1. See the sections on access to content , link techniques , table techniques , frame techniques , and form techniques .

Doing more:

1. If the requirements of the current document contradict the rendering requirements of another specification, the user agent may offer a configuration to allow conformance to one or the other specification.

References:

1. Sections 10.4 ("Client Error 4xx") and 10.5 ("Server Error 5xx") of the HTTP/1.1 specification *[RFC2616]* 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.

2.2 Provide text view. (P1)

1. For content authored in text formats, provide a view of the text source . For the purposes of this document, text formats are defined to be:
   - all media objects given an Internet media type of "text" (e.g., text/plain, text/HTML, or text/*) as defined in RFC 2046 *[RFC2046]* , section 4.1.
   - all SGML and XML applications, regardless of Internet media type (e.g., HTML 4.01, XHTML 1.1, SMIL, SVG, etc.).

Note: A user agent would also satisfy this checkpoint by providing a source view for any text format, not just implemented text formats. The user agent is not required to satisfy this checkpoint for all implemented specifications; see the section on conformance and implementing specifications for more information.

Notes and rationale:

1. In general, user agent developers should not rely on a "source view" to convey information to users, most of whom are not familiar with markup languages. A source view is still important as a "last resort" to some users as content might not otherwise be accessible at all.

Who benefits:

1. Users with blindness, low vision, deafness, hard of hearing, and any user who requires the text source to understand the content.

Example techniques:

1. Make the text view *useful*. For instance, enable links (i.e., URIs), allowing searching and other navigation within the view.
2. A source view is an easily-implementable view that will help users inspect some types of content, such as style sheet fragments or scripts. This does not mean, however, that a source view of style sheets is the *best* user interface for reading or changing style sheets.

Doing more:

1. Provide a source view for any text format, not just implemented text formats.

2.3 Render conditional content. (P1)

1. Allow configuration to provide access to each piece of unrendered conditional content "C".
2. The configuration may be a switch that, for all content, turns on or off the access mechanisms described in the next provision.
3. When a specification does not explain how to provide access to this content, do so as follows:
   - If C is a summary, title, alternative, description, or expansion of another piece of content D, provide access through at least one of the following mechanisms:
     - (1a) render C in place of D;
     - (2a) render C in addition to D;
     - (3a) provide access to C by querying D. In this case, the user agent must also alert the user, on a per-element basis, to the existence of C (so that the user knows to query D);
     - (4a) allow the user to follow a link to C from the context of D.
   - Otherwise, provide access to C through at least one of the following mechanisms:
     - (1b) render a placeholder for C, and allow the user to view the original author-supplied content associated with each placeholder;
     - (2b) provide access to C by query (e.g., allow the user to query an element for its attributes). In this case, the user agent must also alert the user, on a per-element basis, to the existence of C;
     - (3b) allow the user to follow a link in context to C.
4. To satisfy this checkpoint, the user agent may provide access on a per-element basis (e.g., by allowing the user to query individual elements) or for all elements (e.g., by offering a configuration to render conditional content all the time).

For all content.

Note: For instance, an HTML user agent might allow users to query each element for access to conditional content supplied for the "`alt`", "`title`", and "`longdesc`" attributes. Or, the user agent might allow configuration so that the value of the "`alt`" attribute is rendered in place of all `IMG` elements (while other conditional content might be made available through another mechanism). See checkpoint 2.10 for additional placeholder requirements.

Notes and rationale:

1. There may be more than one piece of conditional content associated with another piece of content (e.g., multiple captions tracks associated with the visual track of a presentation).
2. Please note that the alert requirement of this checkpoint is per-element. A single resource-level alert (e.g., "there is conditional content somewhere here") does not satisfy the checkpoint, but may be part of a solution for satisfying this checkpoint. For example, the user agent might indicate the presence of

conditional content "somewhere" with menu in the toolbar. The menu items could provide both per-element alert and access to the content (e.g., by opening a viewport with the conditional content rendered).

Who benefits:

1. Any user for whom the author has provided conditional content for accessibility purposes. This includes: text equivalents for users with blindness or low vision, or users who are deaf-blind, and captions, for users who with deafness or who are hard of hearing.

Example techniques:

1. Allow users to choose more than one piece of conditional content at a given time. 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 synthesized speech.
2. In HTML 4 *[HTML4]* , conditional content mechanisms include the following:
   - For the IMG element (section 13.2): the "alt" (section 13.8), "title" (section 7.4.3), and "longdesc" (section 13.2) attributes. See 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).
3. Allow the user to configure how the user agent renders a long description (e.g., "longdesc" in HTML 4 *[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.
4. 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 conditional content

should be rendered. For example, within the specified geometry, by ignoring the specified geometry altogether, etc.

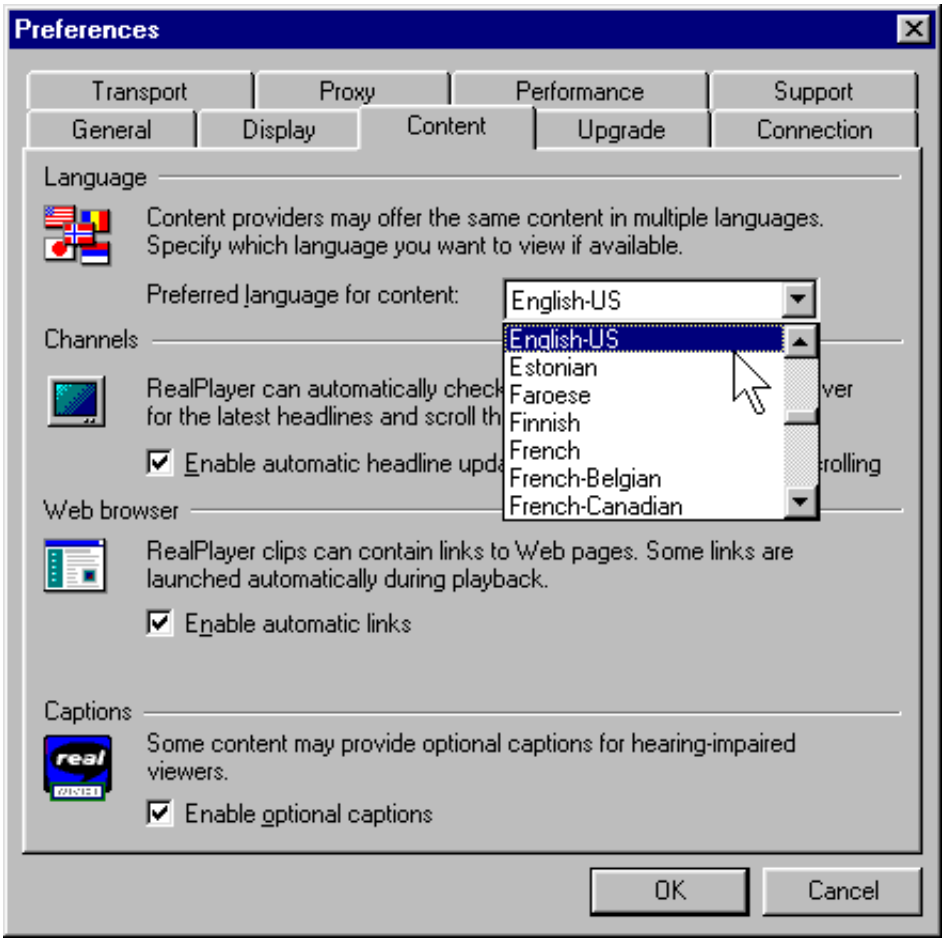
5. For multimedia presentations with several alternative tracks, ensure access to all tracks and allow the user to select individual tracks. The QuickTime player *[QUICKTIME]* allows users to turn on and off any number of tracks separately. For example, construct a list of all available tracks from short descriptions provided by the author (e.g., through the "title" attribute).
6. For multimedia presentations with several alternative tracks, allow users to choose 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).
7. 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. 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.
8. Make apparent through the user agent user interface which audio tracks are meant to be played separately.

Related techniques:

1. See the section on access to content .

Doing more:

1. Make information available with different levels of detail. For example, for a voice browser , offer two options for HTML `IMG` elements:
    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.
2. Allow the user to configure different natural language preferences for different types of conditional content (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 supplementary 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.



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

2.4 Allow time-independent interaction. (P1)

1. For rendered content where user input is only possible within a finite time interval controlled by the user agent, allow configuration to provide a view where user interaction is time-independent.
2. The user agent may satisfy this checkpoint by pausing processing automatically to allow for user input, and resuming processing on explicit user request . When this technique is used, pause at the end of each time interval where user input is possible. In the paused state:
   - Alert the user that the rendered content has been paused (e.g., highlight the "pause" button in a multimedia player's control panel).
   - Highlight which enabled elements are time-sensitive.
   - Allow the user to interact with the enabled elements .
   - Allow the user to resume on explicit user request (e.g., by pressing the

"play" button in a multimedia player's control panel; see also checkpoint 4.5).

3. The user agent may satisfy this checkpoint by generating a time-independent ("static") view, based on the original content , that offers the user the same opportunities for interaction. The static view should reflect the structure and flow of the original time-sensitive presentation; orientation cues will help users understand the context for various interaction opportunities.
4. When satisfying this checkpoint for a real-time presentation, the user agent may discard packets that continue to arrive after the construction of the time-independent view (e.g., when paused or after the construction of a static view).

Note: If the user agent satisfies this checkpoint by pausing automatically, it may be necessary to pause more than once when there are multiple opportunities for time-sensitive user interaction When pausing, pause synchronized content as well (whether rendered in the same or different viewports) per checkpoint 2.6. In SMIL 1.0 *[SMIL]* , for example, the "`begin`", "`end`", and "`dur`" attributes synchronize presentation components. This checkpoint does not apply when the user agent cannot recognize the time interval in the presentation format, or when the user agent cannot control the timing (e.g., because it is controlled by the server). See also checkpoint 3.5, which involves client-driven content refresh.

Notes and rationale:

1. The user agent could satisfy this checkpoint by allowing the user to step through an entire presentation manually (as one might advance frame by frame through a movie). However, this is likely to be tedious and lead to information loss, so the user agent should preserve as much of the flow and order of the original presentation as possible.
2. The requirement to pause at the *end* (rather than at the beginning) of a time-interval is to allow the user to review content that may change during the elapse of this time.
3. The configuration option is important because techniques used to satisfy this checkpoint may lead to information loss for some types of content (e.g., highly interactive real-time presentations).
4. When different streams of time-sensitive content are not synchronized (and rendered in the same or different viewports), the user agent is not required to pause the pieces all at once. The assumption is that both streams of content will be available at another time.

Who benefits:

1. Some users with a physical disability who may not have the time to interact with the content. Also, users who may be accessing the content serially (e.g,. users with blindness or some users with a physical disability) and require more time to reach the timed content

Example techniques:

1. Some HTML user agents recognize time intervals specified through the `META` element, although this usage is not defined in HTML 4 *[HTML4]* .
2. 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.
3. For a presentation that is not "live", allow the user to choose from a menu of available time-sensitive links (essentially making them time-independent).

Doing more:

1. Provide a view where time intervals are lengthened, but not infinitely (e.g., allow the user to multiple time intervals by 3, 5, and 10). Or, allow the user to add extra time (e.g., 10 seconds) to each time interval.
2. Allow the user to view a list of all media elements or links of the presentations sorted by start or end time or alphabetically.
3. Alert the user whenever pausing the user agent may lead to packet loss.

References:

1. Refer to section 4.2.4 of SMIL 1.0 *[SMIL]* for information about the SMIL time model.

2.5 Make captions, transcripts available. (P1)

1. Allow configuration or control to render text transcripts , collated text transcripts , captions , and auditory descriptions at the same time as the associated audio tracks and visual tracks .

For all content.

Note: This checkpoint is an important special case of checkpoint 2.1.

Notes and rationale:

1. Users may wish to a read transcript at the same time as a related visual or audio track and pause the visual or audio track while reading; see checkpoint 4.5.

Who benefits:

1. Users with blindness or low vision (auditory descriptions and text captions, etc.) and users with deafness or who are hard of hearing.

Example techniques:

1. Allow users to turn on and off auditory descriptions and captions.
2. For the purpose of applying this clause, SMIL 1.0 *[SMIL]* user agents should recognize as captions any media object whose reference from SMIL is guarded by the '`system-captions`' test attribute.
3. 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 *[SMIL]* does not include a mechanism analogous to 'system-captions' for auditory descriptions, though *[SMIL20]* is expected to.
4. Another SMIL 1.0 test attribute, '`system-overdub-or-captions`', allows users to choose 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.
5. 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 alternative tracks to provide content for accessibility purposes. The Apple QuickTime player provides this feature through the menu item "Enable Tracks."
6. 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 that play Microsoft Windows Media Object presentations should allow users to turn on and off other conditional content , including auditory description and alternative visual tracks .

References:

1. User agents that implement SMIL 1.0 *[SMIL]* should implement the "Accessibility Features of SMIL" *[SMIL-ACCESS]* .

2.6 Respect synchronization cues. (P1)

1. Respect synchronization cues (e.g., in markup) during rendering.

Note: This checkpoint is an important special case of checkpoint 2.1.

Notes and rationale:

1. The term "synchronization cues" refers to pieces of information that may affect synchronization, such as the size and expected duration of tracks 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).
2. Captions and auditory descriptions may not make sense unless rendered synchronously with related video or audio content. For instance, if someone with a hearing disability is watching a video presentation and reading associated captions, the captions should be synchronized with the audio so that the individual can use any residual hearing. For auditory descriptions, it is crucial that an audio track and an auditory description track be synchronized to avoid having them both play at once, which would reduce the clarity of the presentation.

Who benefits:

1. Users with deafness or who are hard of hearing (e.g., for auditory descriptions and audio tracks), and some users with a cognitive disability.

Example techniques:

1. For synchronization in SMIL 2.0 *[SMIL20]* , refer to section 10, the timing and synchronization module.
2. 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 . Typically, a segment of the captions appears visually near the video for several seconds while the person reads the text. As the visual track continues, a new segment of the captions is presented. However, a problem arises if the captions are 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 rendered caption text that can be presented. The user agent needs to 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.
3. Developers of user agents need to determine how they will handle other synchronization challenges, such as:
    1. Under what circumstances will the presentation automatically pause? Some circumstances where this might occur include:
        - the segment of rendered 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 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 activating a link, then how will related tracks behave? Will they jump as well? Will the user be able to return to a previous location or undo the action?

2.7 Repair missing content. (P2)

1. Allow configuration to generate repair text when the user agent recognizes that the author has failed to provide conditional content that was required by the format specification.
2. The user agent may satisfy this checkpoint by basing the repair text on any of the following available sources of information: URI reference, content type, or element type.

For all content.

Note: Some markup languages (such as HTML 4 *[HTML4]* and SMIL 1.0 *[SMIL]* require the author to provide conditional content for some elements (e.g., the "`alt`" attribute on the `IMG` element). Repair text based on URI reference, content type, or element type is sufficient to satisfy the checkpoint, but may not result in the most effective repair. Information that may be recognized as relevant to repair might not be "near" the missing conditional content in the document object . For instance, instead of generating repair text on a simple URI reference, the user agent might look for helpful information near a different instance of the URI reference in the same document object, or might retrieve useful information (e.g., a title) from the resource designated by the URI reference.

Notes and rationale:

1. Some examples of missing conditional content that is required by format specification:
   - in HTML 4 *[HTML4]* , "`alt`" is required for the `IMG` and `AREA` elements (for validation). In SMIL 1.0 *[SMIL]* , on the other hand, "`alt`" is not required on media objects.
   - whatever the format, text equivalents for non-text content are required by the Web Content Accessibility Guidelines 1.0 *[WCAG10]* .
2. Conditional content may come from markup, inside images (e.g., refer to "Describing and retrieving photos using RDF and HTTP" *[PHOTO-RDF]*), etc.

Who benefits:

1. Users with blindness or low vision.

Example techniques:

1. 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 "Uniform Resource Identifiers (URI): Generic Syntax" (*[RFC2396]* , section 4) for information about URI references, as well as the HTTP/1.1 specification *[RFC2616]* , section 3.2.1.

Related techniques:

1. See content repair techniques , and cell header repair strategies .

Doing more:

1. When configured to generate text, also inform the user (e.g., in the generated text itself) that this content was not provided by the author as a text equivalent.

References:

1. The "Altifier Tool" *[ALTIFIER]* illustrates smart techniques for generating text equivalents (for images, etc.) when the author has not specified any.

2.8 No repair text. (P3)

1. Allow at least two configurations for when the user agent recognizes that conditional content required by the format specification is present but empty :
   - generate no repair text , or
   - generate repair as described in checkpoint 2.7.

For all content.

Note: In some authoring scenarios, empty content (e.g., a string of zero characters) may make an appropriate text equivalent , such as when non-text content has no other function than pure decoration, or when an image is part of a "mosaic" of several images and doesn't make sense out of the mosaic. Please refer to the Web Content Accessibility Guidelines 1.0 *[WCAG10]* for more information about text equivalents.

Notes and rationale:

1. User agents should render nothing in this case because the author may specify an empty text equivalent for content that has no function in the page other than as decoration.

Who benefits:

1. Users with blindness or low vision.

Example techniques:

1. The user agent should not render generic labels such as "[INLINE]" or "[GRAPHIC]" in the face of empty conditional content (unless configured to do so).
2. If no captioning information is available and captioning is turned on, render "no captioning information available" in the captioning region of the viewport (unless configured not to generate repair content).

Doing more:

1. Labels (e.g., "[INLINE]" or "[GRAPHIC]") may be useful in some situations, so the user agent may allow configuration to render "No author text" (or similar) instead of empty conditional content.

2.9 Render conditional content automatically. (P3)

1. Allow configuration to render all conditional content automatically. The user agent is not required to render all conditional content at the same time in a single viewport.
2. Provide access to this content according to format specifications or where unspecified, by applying one of the techniques described in checkpoint 2.3: 1a, 2a, or 1b.

For all content.

Note: For instance, an HTML user agent might allow configuration so that the value of the "`alt`" attribute is rendered in place of all `IMG` elements (while other conditional content might be made available through another mechanism). The user agent may offer multiple configurations (e.g., a first configuration to render one type of conditional content automatically, a second to render another type, etc.).

Who benefits:

1. Any user who may have difficulties with navigation and manual access to content, including some users with a physical disability and users with blindness or low vision.

Example techniques:

1. Provide a "conditional content view", where all content that is not rendered by default is rendered in place of associated content. For example, Amaya *[AMAYA]* offers a "Show alternate" view that accomplishes this. Note, however, cases where an element has more than one piece of associated conditional content (e.g., render them all as a list, or as a list of links, etc.). For long conditional content, instead of rendering in place, link to the content.

2.10 Toggle placeholders. (P3)

1. Once the user has viewed the original author-supplied content associated with a placeholder , allow the user to turn off the rendering of the author-supplied content.

Note: For example, if the user agent substitutes the author-supplied content for the placeholder in context, allow the user to "toggle" between placeholder and the associated content. Or, if the user agent renders the author-supplied content in a separate viewport, allow the user to close that viewport. **Note:** See checkpoint 2.3, provision (1b) for placeholder requirements.

Who benefits:

1. Some users with a cognitive disability may find it difficult to access content once too many images (for example) have been rendered one by one.

Example techniques:

1. Allow the user to designate a placeholder and request to view the associated content in a separate viewport (e.g., through the context menu), leaving the placeholder in context. Per checkpoint 5.3, users are able to close the new viewport.

2.11 Alert unsupported language. (P3)

1. Allow configuration not to render content in unsupported natural languages , when that content would otherwise be rendered. Content "in a natural language" includes pre-recorded spoken language and text in a given script , i.e., writing system.
2. Indicate to the user in context that author-supplied content has not been rendered.
3. This checkpoint does not require the user agent to allow different configurations for different natural languages.

Note: For example, use a text substitute or accessible graphical icon to indicate that content in a particular language has not been rendered.

Notes and rationale:

1. A script is a means of supporting the visual rendering of content in a particular natural language. So, for user agents that render content visually, a user agent might not recognize "the Cyrillic script", which would mean that it would not support the visual rendering of Russian, Ukrainian, and other languages that employ Cyrillic when written.
2. Rendering content in an unsupported language (e.g., as "garbage" characters) may confuse all users. However, this checkpoint is designed primarily to benefit users who access content serially as it allows them to skip portions of content that would be unusable as rendered.
3. There may be cases when a conforming user agent supports a natural language but a speech synthesizer does not, or vice versa.

Who benefits:

1. Users who access content serially, including users with blindness and some users with a physical disability.

Example techniques:

1. For instance, a user agent that doesn't support Korean (e.g., doesn't have the appropriate fonts or voice set) should allow configuration to announce the language change with the message "Unsupported language – unable to render" (e.g., when the language itself is not recognized) or "Korean not supported – unable to render" (e.g., when the language is recognized by the user agent doesn't have resources to render it). The user should also be able to choose no alert 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 alert could be spoken in the default language by a screen reader or voice browser .
2. 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 *[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.
3. When HTTP is used, HTTP headers provide information about content encoding ("Content-Encoding") and content language ("Content-Language"). Refer to the HTTP/1.1 specification *[RFC2616]* , sections 14.11 and 14.12, respectively.

4. CSS2's attribute selector may be used with the HTML "lang" or XML "xml:lang" attributes to control rendering based on recognized natural language information. Refer also to the ':lang' pseudo-class (*[CSS2]* , section 5.11.4).

Related techniques:

1. See techniques for generated content , which may be used to insert text to indicate a language change.
2. See content repair techniques and accessibility and internationalization techniques .
3. See techniques for synthesized speech .

References:

1. For information on language codes, refer to "Codes for the representation of names of languages" *[ISO639]* .
2. 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.

Guideline 3. Allow configuration 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

3.1 Toggle background images. (P1)

1. Allow configuration not to render background image content .
2. In this configuration, the user agent is not required to retrieve background images from the Web.
3. This checkpoint only requires control of background images for "two-layered renderings", i.e., one rendered background image with all other content rendered "above it".

Note: See checkpoint 2.3 for information about how to provide access to unrendered background images. When background images are not rendered, user agents should render a solid background color instead (see checkpoint 4.3).

Notes and rationale:

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).

Who benefits:

1. Some users with a cognitive disability or color deficiencies who may find it difficult or impossible to read superimposed text or understand other superimposed content.

Example techniques:

1. If background image are turned off, make available to the user associated conditional content .
2. In CSS, background images may be turned on/off with the 'background' and 'background-image' properties (*[CSS2]* , section 14.2.1).

Doing more:

1. Allow control of image depth in multi-layer presentations.

3.2 Toggle audio, video, animated images. (P1)

1. Allow configuration not to render audio, video, or animated image content , except on explicit user request . This configuration is required for content rendered without any user interaction (including content rendered on load or as the result of a script), as well as content rendered as the result of user interaction (e.g., when the user activates a link).
2. The user agent may satisfy this checkpoint by making video and animated images invisible and audio silent , but this technique is not recommended.
3. When configured not to render content except on explicit user request, the user agent is not required to retrieve the audio, video, or animated image from the Web until requested by the user.

Note: See checkpoint 2.3 for information about how to provide access to unrendered audio, video, and animated images. See also checkpoint 4.5, checkpoint 4.9, and checkpoint 4.10.

Who benefits:

1. Some users with a cognitive disability, for whom an excess of visual information (and in particular animated information) might it impossible to understand parts of content. Also, audio rendered automatically on load may interfere with speech synthesizers.

Example techniques:

1. For user agents that hand off content to different rendering engines, the configuration should cause the content not to be handed off, and instead a placeholder rendered.
2. The "silent" or "invisible" solution for satisfying this checkpoint (e.g., by implementing the 'visibility' property defined in section 11.2 of CSS 2 *[CSS2]*). is not recommended. This solution means that the content is processed, though not rendered, and processing may cause undesirable side effects such as firing events. Or, processing may interfere with the processing of other content (e.g., silent audio may interfere with other sources of sound such as the output of a speech synthesizer). This technique should be deployed with caution.
3. As a placeholder for an animated image, render a motionless image built from the first frame of the animated image.

3.3 Toggle animated/blinking text. (P1)

1. Allow configuration to render animated or blinking text content . as motionless, unblinking text. Blinking text is text whose visual rendering alternates between visible and invisible, any rate of change.
2. In this configuration, the user must still have access to the same text content, but the user agent may render it in a separate viewport (e.g., for large amounts of streaming text).
3. The user agent also satisfies this checkpoint by always rendering animated or blinking text as motionless, unblinking text.

Note: Animation (a rendering effect) differs from streaming (a delivery mechanism). Streaming content might be rendered as an animation (e.g., an animated stock ticker or vertically scrolling text) or as static text (e.g., movie subtitles, which are rendered for a limited time, but do not give the impression of movement). See also checkpoint 3.5. This checkpoint does not apply for blinking and animation effects that are caused by mechanisms that the user agent cannot recognize.

Notes and rationale:

1. The definition of blinking text is based on the CSS2 definition of the 'blink' value; refer to *[CSS2]* , section 16.3.1.

Who benefits:

1. Flashing content may trigger seizures in people with photosensitive epilepsy, or may make a Web page too distracting to be usable by someone with a cognitive disability. Blinking text can affect screen reader users, since screen readers (in conjunction with speech synthesizers or braille displays) may re-render the text every time it blinks.
2. Configuration is preferred as some users may benefit from blinking effects (e.g.,

users who are deaf or hard of hearing). However, the priority of this checkpoint was assigned on the basis of requirements unrelated to this benefit.

Example techniques:

1. The user agent may render the motionless text in a number of ways. Inline is preferred, but for extremely long text, it may be better to render the text in another viewport, easily reachable from the user's browsing context.
2. 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).
3. Some sources of blinking and 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).
   - In JavaScript, to control the start and speed of scrolling for a `MARQUEE` element:
     - `document.all.myBanner.start();`
     - `document.all.myBanner.scrollDelay = 100`

3.4 Toggle scripts. (P1)

1. Allow configuration not to execute any executable content (e.g., scripts and applets).
2. In this configuration, provide an option to alert the user when executable content is available (but has not been executed).
3. The user agent is only required to alert the user to the presence of more than zero scripts or applets (i.e., per-element alerts are not required).

Note: This checkpoint does not refer to plug-ins and other programs that are not part of content . Scripts and applets may provide very useful functionality, not all of which causes accessibility problems. Developers should not consider that the user's ability to turn off scripts is an effective way to improve content accessibility; turning off scripts means losing the benefits they offer. Instead, developers should provide users with finer control over user agent or content behavior known to raise accessibility barriers. The user should only have to turn off scripts as a last resort.

Notes and rationale:

1. Executable content includes scripts, applets , ActiveX controls, etc. This checkpoint does not apply to plug-ins ; they are not part of content .
2. Executable content includes those that run "on load" (e.g., when a document loads into a viewport) and when other events occur (e.g., user interface events).
3. The alert that scripts are available but not executed is important, for instance,

for helping users understand why some poorly authored pages without script alternatives produce no content when scripts are turned off.
4. Where possible, authors should encode knowledge in declarative formats rather than in scripts. Knowledge and behaviors embedded in scripts is difficult to extract, which means that user agents are less likely to be able to offer control by the user over the script's effect.

Who benefits:

1. Control of executable content is particularly important as it can 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. Peak sensitivity to flickering or flashing occurs at 20 Hertz.

Example techniques:

1. Do not make available the switch to turn scripts off only in the "Security" part of the user interface as people may not think to look there. For instance, include a "Scripts" entry in the index that allows people to find the switch more easily.

Related techniques:

1. See the section on script techniques .

Doing more:

1. While this checkpoint only requires an on/off configuration switch, user agents should allow finer control over executable content. For instance, in addition to the switch, allow users to turn off just input device event handlers, or to turn on and off scripts in a given scripting language only.

3.5 Toggle content refresh. (P1)

1. Allow configuration so that the user agent only refreshes content on explicit user request .
2. In this configuration, alert the user of the refresh rate specified in content, and allow the user to request fresh content manually (e.g., by following a link or confirming a prompt).
3. When the user chooses not to refresh content, the user agent may ignore that content; buffering is not required.
4. This checkpoint only applies when the user agent (not the server) automatically initiates the request for fresh content.

Note: For example, allow configuration to prompt the user to confirm content refresh, at the rate specified by the author.

Notes and rationale:

1. 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.

Who benefits:

1. Automatically changing content can disorient some users with a cognitive disability, users with blindness or low vision, and most users.

Example techniques:

1. Alert the user that suppressing refresh may lead to loss of information (i.e., packet loss).

Doing more:

1. Allow users to specify their own refresh rate.
2. Allow configuration for at least one very slow refresh rate (e.g., every 10 minutes).
3. Retrieve new content without displaying it automatically. Allow the user to view the differences (e.g., by highlighting or filtering) between the currently rendered content and the new content (including no differences).

3.6 Toggle redirects. (P2)

1. Allow configuration so that a "client-side redirect" (i.e., one initiated by the user agent, not the server) only changes content on explicit user request .
2. Allow the user to access the new content on demand (e.g., by following a link or confirming a prompt).
3. The user agent is not required to provide these functionalities for client-side redirects specified to occur instantaneously (i.e., after no delay).

Note: Some HTML user agents support client-side redirects authored using a `META` element with `http-equiv="refresh"`. Authors (and Web masters) should use the redirect mechanisms of HTTP instead.

Notes and rationale:

1. This checkpoint is a Priority 2 checkpoint in part because the author’s redirect implies that users aren’t expected to use the content prior to the redirect.

Who benefits:

1. Automatically changing content can disorient some users with a cognitive disability, users with blindness or low vision, and most users.

Example techniques:

1. Provide a configuration so that when the user navigates "back" through the user agent history to a page with a client-side redirect, the user agent does not re-execute the client-side redirect.

Doing more:

1. Allow configuration to allow access on demand to new content even when the client-side redirect has been specified by the author to be instantaneous.

References:

1. For Web content authors: refer to the HTTP/1.1 specification *[RFC2616]* for information about using server-side redirect mechanisms (instead of client-side redirects).

3.7 Toggle images. (P2)

1. Allow configuration not to render image content .
2. The user agent may satisfy this checkpoint by making images invisible , but this technique is not recommended.

Note: See checkpoint 2.3 for information about how to provide access to unrendered images.

Notes and rationale:

1. This priority of checkpoint 3.2 is higher than the priority of this checkpoint because an excess of moving visual information is likely to be more distracting to some users than an excess of still visual information.

Who benefits:

1. Some users with a cognitive disability, for whom an excess of visual information might it difficult to understand parts of content.

Related techniques:

1. See techniques for checkpoint 3.1.

Guideline 4. Ensure user control of rendering.

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

Checkpoints for visually rendered text

4.1 Configure text size. (P1)

1. Allow global configuration of the reference size of visually rendered text , with an option to override reference sizes specified by the author or user agent defaults.
2. Offer a range of text sizes to the user that includes at least:
   - the range offered by the conventional utility available in the operating environment that allows users to choose the text size (e.g., the font size),
   - or, if no such utility is available, the range of text sizes supported by the conventional APIs of the operating environment for drawing text.

Note: The reference size of rendered text corresponds to the default value of the CSS2 'font-size' property, which is 'medium' (refer to CSS2 *[CSS2]* , section 15.2.4). For example, in HTML, this might be paragraph text. The default reference size of rendered text may vary among user agents. User agents may offer different mechanisms to allow control of the size of rendered text (e.g., font size control, zoom, magnification, etc.). Refer, for example to the Scalable Vector Graphics specification *[SVG]* for information about scalable rendering.

Notes and rationale:

1. For example, allow the user to configure the user agent to apply the same font family across Web resources, so that all text is displayed by default using that font family. Or, allow the user to control the text size dynamically for a given element, e.g., by navigating to the element and zooming in on it.
2. 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 rendered 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 rendered 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.
3. The primary intention of this checkpoint is to allow users with low vision to increase the size of text. Full configurability includes the choice of (very) small text sizes that may be available, though this is not considered by the User Agent Accessibility Guidelines Working Group to be part of the priority 1 requirement. This checkpoint does not include a "lower bound" (above which text sizes would be required) because of how users' needs may vary across writing systems and

hardware.

Who benefits:

1. Users with low vision benefit from the ability to increase the text size. Note that some users may also benefit from the ability to choose small font sizes (e.g., users of screen readers who wish to have more content per screen so they have to scroll less frequently).

Example techniques:

1. Inherit text size information from user preferences specified for the operating environment .
2. Use operating environment magnification features.
3. When scaling text, maintain size relationships among text of different sizes.
4. Implement the 'font-size' property in CSS (*[CSS2]* , section 15.2.4).
5. For example, in Windows, the `ChooseFont` function in the Comdlg32 library will create the conventional utility of that operating system that allows users to choose text (font) size. The `DrawText` API is the lower-level API for drawing text.

Doing more:

1. Allow the user to configure the text size on an element level (i.e., more precisely than globally). User style sheets allow such detailed configurations.
2. Allow the user to configure the text size differently for different scripts (i.e., writing systems).

4.2 Configure font family. (P1)

1. Allow global configuration of the font family of all visually rendered text , with an option to override font families specified by the author or by user agent defaults.
2. Offer a range of font families to the user that includes at least:
   - the range offered by the conventional utility available in the operating environment that allows users to choose the font family,
   - or, if no such utility is available, the range of font families supported by the conventional APIs of the operating environment for drawing text.
3. For text that cannot be rendered properly using the user's preferred font family, the user agent may substitute an alternative font family.

Note: For example, allow the user to specify that all text is to be rendered in a particular sans-serif font family.

Who benefits:

1. Users with low vision or some users with a cognitive disability or reading disorder require the ability to change the font family of text in order to read it.

Example techniques:

1. Inherit font family information from user preferences specified for the operating environment .
2. Implement the 'font-family' property in CSS (*[CSS2]* , section 15.2.2).
3. Allow the user to override author-specified font families with differing levels of detail. For instance, use font A in place of any sans-serif font and font B in place of any serif font.
4. For example, in Windows, the `ChooseFont` function in the Comdlg32 library will create the conventional utility of that operating system that allows users to choose font families. The `DrawText` API is the lower-level API for drawing text.

Doing more:

1. Allow the user to configure font families on an element level (i.e., more precisely than globally). User style sheets allow such detailed configurations.

4.3 Configure text colors. (P1)

1. Allow global configuration of the foreground and background color of all visually rendered text , with an option to override foreground and background colors specified by the author or user agent defaults.
2. Offer a range of colors to the user that includes at least:
   - the range offered by the conventional utility available in the operating environment that allows users to choose colors,
   - or, if no such utility is available, the range of colors supported by the conventional APIs of the operating environment for specifying colors.

Note: User configuration of foreground and background colors may inadvertently lead to the inability to distinguish ordinary text from selected text, focused text, etc. See checkpoint 10.3 for more information about highlight styles.

Who benefits:

1. Users with color deficiencies and some users with a cognitive disability.

Example techniques:

1. Inherit foreground and background color information from user preferences specified for the operating environment .
2. Implement the 'color' and 'border-color' properties in CSS 2 (*[CSS2]* , sections 14.1 and 8.5.2, respectively).

3. Implement the 'background-color' property (and other background properties) in CSS 2 (*[CSS2]* , section 14.2.1).
4. SMIL does not have a global property for "background color", but allows specification of background color by region (refer, for example, to the definition of the '`background-color`' attribute defined in section 3.3.1 of SMIL 1.0 *[SMIL]*). In the case of SMIL, the user agent would satisfy this checkpoint by applying the users preferred background color to all regions (and to all `root-layout` elements as well). SMIL 1.0 does not have a way to specify the foreground color of text, so that portion of the checkpoint would not apply.
5. In SVG 1.0 *[SVG]* , the 'fill' and 'stroke' properties are used to paint foreground colors.
6. For example, in Windows, the `ChooseColor` function in the Comdlg32 library will create the conventional utility of that operating system that allows users to choose colors. The `DrawText` API is the lower-level API for drawing text.

Doing more:

1. Allow the user to specify minimal contrast between foreground and background colors, adjusting colors dynamically to meet those requirements.

Checkpoints for multimedia presentations and other presentations that change continuously over time

4.4 Slow multimedia. (P1)

1. Allow the user to slow the presentation rate of rendered audio and animations (including video and animated images).
2. For a visual track , provide at least one setting between 40% and 60% of the original speed.
3. For a prerecorded audio track including audio-only presentations , provide at least one setting between 75% and 80% of the original speed.
4. When the user agent allows the user to slow the visual track of a synchronized multimedia presentation to between 100% and 80% of its original speed, synchronize the visual and audio tracks. Below 80%, the user agent is not required to render the audio track .
5. The user agent is not required to satisfy this checkpoint for audio and animations whose recognized role is to create a purely stylistic effect.

Note: Purely stylistic effects include background sounds, decorative animated images, and effects caused by style sheets. The style exception of this checkpoint is based on the assumption that authors have satisfied the requirements of the "Web Content Accessibility Guidelines 1.0" *[WCAG10]* not to convey information through style alone (e.g., through color alone or style sheets alone). See checkpoint 2.6 and checkpoint 4.7.

Notes and rationale:

1. Slowing one track (e.g., video) may make it harder for a user to understand another synchronized track (e.g., audio), but if the user can understand content after two passes, this is better than not being able to understand it at all.
2. Some formats (e.g., streaming formats), might not enable the user agent to slow down playback and would thus be subject to applicability.

Who benefits:

1. Some users with a learning or cognitive disability, or some users with newly acquired sensory limitations (such as a person who is newly blind and learning to use a screen reader). Users who have beginning familiarity with a natural language may also benefit.

Example techniques:

1. When changing the rate of audio, avoid pitch distortion.
2. HTML 4 *[HTML4]* , background animations may be specified with the deprecated `background` attribute.
3. The SMIL 2.0 Time Manipulations Module (*[SMIL20]* , chapter 11) defines the `speed` attribute, which can be used to change the playback rate (as well as forward or reverse direction) of any animation.
4. Authors sometimes specify background sounds with the "bgsound" attribute. **Note:** This attribute is **not** part of HTML 4 *[HTML4]* .

Doing more:

1. Allowing the user to speed up audio is also useful. For example, some users who access content serially benefit from the ability to speed up audio.

References:

1. Refer to variable playback speed techniques used for Digital Talking Books *[TALKINGBOOKS]* .

4.5 Start, stop, pause, and navigate multimedia. (P1)

1. Allow the user to stop, pause, and resume rendered audio and animations (including video and animated images) that last three or more seconds at their default playback rate.
2. Allow the user to navigate efficiently within audio and animations (including video and animated images) that last three or more seconds at their default playback rate. The user agent may satisfy this requirement through forward and backward sequential access techniques (e.g., advance three seconds), or direct access techniques (e.g., play starting at the 10-minute mark), or some combination.

3. When serial techniques are used to satisfy the previous requirement, the user agent is not required to play back content during serial advance or rewind (though doing so may help orient the user).
4. The user agent is not required to satisfy this checkpoint for audio and animations whose recognized role is to create a purely stylistic effect.
5. When the user pauses a real-time audio or animation, the user agent may discard packets that continue to arrive during the pause.

Note: See checkpoint 4.4 for more information about the exception for purely stylistic effects. This checkpoint applies to content that is either rendered automatically or on request from the user. Respect synchronization cues per checkpoint 2.6.

Notes and rationale:

1. Some formats (e.g., streaming formats), might not enable the user agent to fast forward or rewind content and would thus be subject to applicability.
2. For some streaming media formats, the user agent might not be able to offer some functionalities (e.g,. fast forward) when the content is being delivered over the Web in real time. However, the user agent is expected to offer these functionalities for content (in the same format) that is fully available, for example on the user's computer.
3. Playback during serial advance or rewind is not required. For example, the user agent is not required to play an animation at double speed during a continuous fast forward. Similarly, when the user fast forwards or rewinds an animation, the user agent is not required to play back a synchronized audio track.

Who benefits:

1. Some users with a cognitive disability. Some users with a physical disability who may not have fine control over advance and rewind functionalities will find useful the ability to advance or rewind the presentation in (configurable) increments.

Example techniques:

1. Sequential access techniques include, for example, rewind in large or small time increments, forward to the next audio track, etc. Direct access techniques include access to video track number 7, to the 10-minute mark, etc. The best choice of serial or direct access techniques will depend on the nature of the content being rendered.
2. 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.
3. The SMIL 2.0 Time Manipulations Module (*[SMIL20]* , chapter 11) defines the `speed` attribute, which can be used to change the playback direction (forward or reverse) of any animation. See also the `accelerate` and `decelerate` attributes.

4. Some content lends itself to different forward and reverse functionalities. For instance, compact disk players often let listeners fast forward and rewind, but also skip to the next or previous song.

Doing more:

1. The user agent should display time codes or represent otherwise position in content to orient the user.
2. Apply techniques for changing audio speed without introducing distortion.
3. Alert the user whenever pausing the user agent may lead to packet loss.

References:

1. Refer to fast forward and rewind techniques used for Digital Talking Books *[TALKINGBOOKS]* .
2. Home Page Reader *[HPR]* lets users insert bookmarks in presentations.

4.6 Position captions. (P1)

1. For graphical viewports, allow the user to position rendered captions with respect to synchronized visual tracks as follows:
   - if the user agent satisfies this checkpoint by using a markup language or style sheet language to provide configuration or control, then the user agent must allow the user to choose from among at least the range of positions enabled by the format
   - otherwise the user agent must allow both non-overlapping and overlapping positions (e.g., by rendering captions in a separate viewport that may be positioned on top of the visual track).
2. In either case, the user agent must allow the user to override the author's specified position.
3. The user agent is not required to change the layout of other content (i.e., reflow) after the user has changed the position of captions.
4. The user agent is not required to make the captions background transparent when those captions are rendered above a related video track.

Notes and rationale:

1. One good reasons to render captions in an independent viewport is to allow users with screen access programs to focus on them.
2. Traditionally, captions have a background, and research shows that some users prefer a black background behind white lettering is preferred.

Who benefits:

1. Some users (e.g., with a cognitive disability) may 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 a screen magnifier) may require pieces of content to be in a particular relation to one another, even if this means that some content will obscure other content.

Example techniques:

1. User agents should implement the positioning features of the employed markup or style sheet language. Even when a markup language does not specify a positioning mechanism, 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).
2. For the purpose of applying this clause, SMIL 1.0 *[SMIL]* user agents should recognize as captions any media object whose reference from SMIL is guarded by the '`system-captions`' test attribute.
3. Implement the CSS 2 'position' property (*[CSS2]* , section 9.3.1).
4. 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.
5. 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.
6. 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.
7. Do not require users to edit the source code of the presentation to achieve the desired effect.

Doing more:

1. The user agent may allow configuration for transparent backgrounds. Refer to checkpoint 4.3 for requirements related to the control of text background colors.
2. 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).
3. Allow the user to resize (graphically) the captions, etc.

4.7 Slow other multimedia. (P2)

1. Allow the user to slow the presentation rate of rendered audio and animations (including video and animated images) not covered by checkpoint 4.4.
2. The same speed percentage requirements of checkpoint 4.4 apply.

Note: User agents automatically satisfy this checkpoint if they satisfy checkpoint 4.4 for all audio and animations.

Who benefits:

1. See the techniques for checkpoint 4.4.

Related techniques:

1. See the techniques for checkpoint 4.4.

4.8 Control other multimedia. (P2)

1. Allow the user to stop, pause, resume, and navigate efficiently rendered audio and animations (including video and animated images) not covered by checkpoint 4.5.

Note: User agents automatically satisfy this checkpoint if they satisfy checkpoint 4.5 for all audio and animations.

Who benefits:

1. See the techniques for checkpoint 4.4.

Related techniques:

1. See the techniques for checkpoint 4.5.

Checkpoints for audio volume control

4.9 Global volume control. (P1)

1. Allow global configuration of the volume of all rendered audio, with an option to override audio volumes specified by the author or user agent defaults.
2. Allow the user to choose zero volume (i.e., silent).

Note: User agents should allow configuration of volume through available operating environment controls.

Example techniques:

1. Use audio control mechanisms provided by the operating environment . Control of volume mix is particularly important, and the user agent should provide easy access to those mechanisms provided by the operating environment.
2. Implement the CSS 2 'volume' property (*[CSS2]* , section 19.2).
3. Implement the 'display', 'play-during', and 'speak' properties in CSS 2 (*[CSS2]* , sections 9.2.5, 19.6, and 19.5, respectively).
4. Authors sometimes specify background sounds with the "bgsound" attribute. **Note:** This attribute is **not** part of HTML 4 *[HTML4]* .

Who benefits:

1. Users who are hard of hearing or who rely on audio and synthesized speech rendering. Users in a noisy environment will also benefit.

References:

1. Refer to guidelines for audio characteristics used for Digital Talking Books *[TALKINGBOOKS]* .

4.10 Independent volume control. (P1)

1. Allow independent control of the volumes of rendered audio sources synchronized to play simultaneously.
2. The user agent is not required to satisfy this checkpoint for audio whose recognized role is to create a purely stylistic effect.
3. The user control required by this checkpoint includes the ability to override author-specified volumes for the relevant sources of audio.

Note: See checkpoint 4.4 for more information about the exception for purely stylistic effects. The user agent should satisfy this checkpoint by allowing the user to control independently the volumes of all audio sources (e.g., by implementing a general audio mixer type of functionality). See also checkpoint 4.13.

Notes and rationale:

1. Sounds that play at different times are distinguishable and therefore independent control of their volumes is not required by this checkpoint (since volume control required by checkpoint 4.9 suffices).
2. There are at least three good reasons for strongly recommending that all sounds be independently configurable, not just those synchronized to play simultaneously.
    1. sounds that are not synchronized may end up playing simultaneously;
    2. if the user cannot anticipate when a sound will play, the user cannot adjust the global volume control at appropriate times to affect this sound;
    3. it is extremely inconvenient to have to adjust the global volume frequently.

3. Sounds specified by the author to play "on document load" are likely to overlap with each other. If they continue to play, they are also likely to overlap with subsequent sounds played manually or automatically.

Who benefits:

1. Users (e.g., with blindness or low vision) who rely on audio and synthesized speech rendering.

Related techniques:

1. For each source of audio , allow the user to control the volume using the same user interface used to satisfy the requirements of checkpoint 4.5.

4.11 Control other volume. (P2)

1. Allow independent control of the volumes of rendered audio sources synchronized to play simultaneously that are not covered by checkpoint 4.10.

Note: User agents automatically satisfy this checkpoint if they satisfy checkpoint 4.10 for all audio.

Who benefits:

1. See the techniques for checkpoint 4.10.

Related techniques:

1. See the techniques for checkpoint 4.10.

Checkpoints for synthesized speech rendering

See also techniques for synthesized speech rendering .

4.12 Configure synthesized speech rate. (P1)

1. Allow configuration of the synthesized speech rate, according to the full range offered by the speech synthesizer.

Note: The range of synthesized speech rates offered by the speech synthesizer may depend on natural language.

Example techniques:

1. For example, many speech synthesizers offer a range for English speech of 120 - 500 words per minute or more. The user should be able to increase or decrease the rendering rate in convenient increments (e.g., in large steps, then in small steps for finer control).

2. User agents may allow different synthesized speech rate configurations for different natural languages. For example, this may be implemented with CSS2 style sheets using the :lang pseudo-class (*[CSS2]* , section 5.11.4).
3. Use synthesized speech mechanisms provided by the operating environment .
4. Implement the CSS 2 'speech-rate' property (*[CSS2]* , section 19.8).

Who benefits:

1. Users (e.g., with blindness or low vision) who rely on audio and synthesized speech rendering.

Doing more:

1. Content may include commands that are interpreted by a speech synthesizer to change the rate (or control other synthesized speech parameters). This checkpoint does not require the user agent to allow the user to override author-specified rate changes (e.g., by transforming or otherwise stripping out these commands before passing on the content to the speech synthesizer). Speech synthesizers themselves may allow user override of author-specified rate changes. For these such synthesizers, the user agent should ensure access to this feature as part of satisfying this checkpoint.

4.13 Configure synthesized speech volume. (P1)

1. Allow control of the synthesized speech volume, independent of other sources of audio .
2. The user control required by this checkpoint includes the ability to override author-specified synthesized speech volume.

Note: See also checkpoint 4.10.

Example techniques:

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

Who benefits:

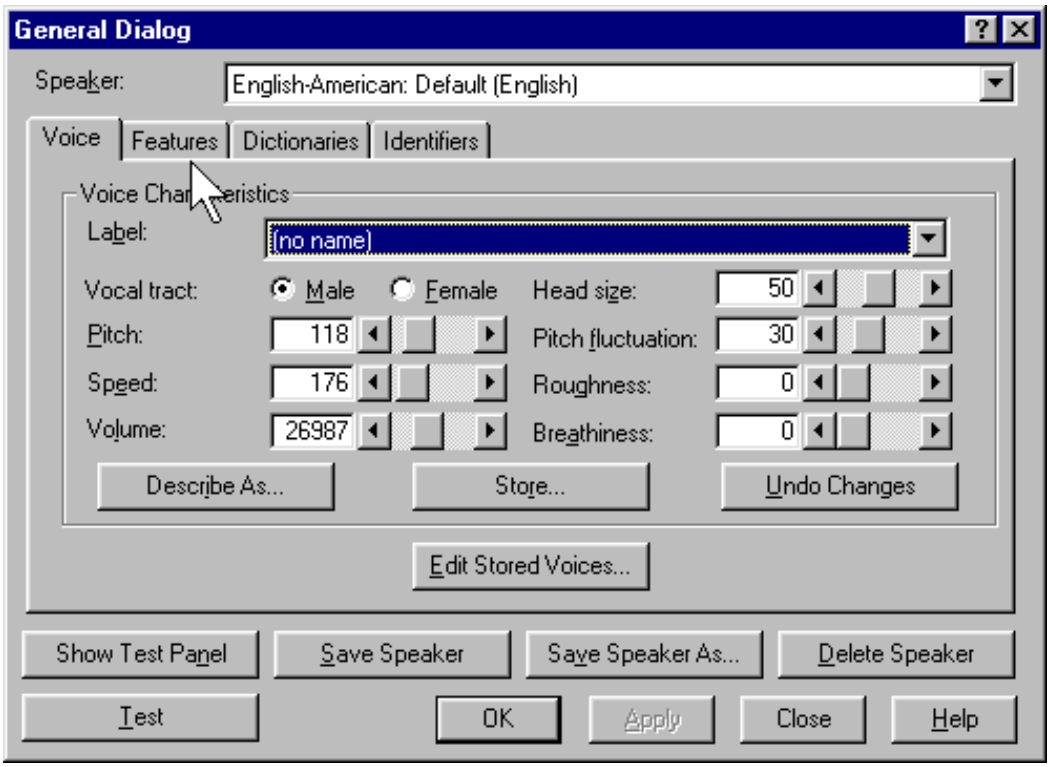
1. Users (e.g., with blindness or low vision) who rely on audio and synthesized speech rendering.

4.14 Configure synthesized speech characteristics. (P1)

1. Allow configuration of synthesized speech characteristics according to the full range of values offered by the speech synthesizer.

Note: Some speech synthesizers allow users to choose values for synthesized speech characteristics at a higher abstraction layer, i.e., by choosing from present options that group several characteristics. Some typical options one might encounter include: "adult male voice", "female child voice", "robot voice", "pitch", "stress", etc. Ranges for values may vary among speech synthesizers.

Example techniques:

1. Use synthesized speech mechanisms provided by the operating environment .
2. One example of a synthesized speech API is Microsoft's Speech Application Programming Interface *[SAPI]* .
3. 

This image shows how ViaVoice *[VIAVOICE]* allows users to configure voice characteristics of the speech synthesizer.

Who benefits:

1. Users (e.g., with blindness or low vision) who rely on audio and synthesized speech rendering.

References:

1. For information about these synthesized speech characteristics, please refer to descriptions in section 19.8 of Cascading Style Sheets Level 2 *[CSS2]* .

4.15 Specific synthesized speech characteristics. (P2)

1. Allow configuration of the following synthesized speech characteristics: pitch, pitch range, stress, richness.
2. Pitch refers to the average frequency of the speaking voice.
3. Pitch range specifies a variation in average frequency.
4. Stress refers to the height of "local peaks" in the intonation contour of the voice.
5. Richness refers to the richness or brightness of the voice.

Note: This checkpoint is more specific than checkpoint 4.14: it requires support for the voice characteristics listed. Definitions for these characteristics are based on descriptions in section 19 of the Cascading Style Sheets Level 2 Recommendation *[CSS2]* ; please refer to that specification for additional informative descriptions. Some speech synthesizers allow users to choose values for synthesized speech characteristics at a higher abstraction layer, i.e., by choosing from present options distinguished by "gender", "age", "accent", etc. Ranges of values may vary among speech synthesizers.

Who benefits:

1. See the techniques for checkpoint 4.14.

Related techniques:

1. See the techniques for checkpoint 4.14.

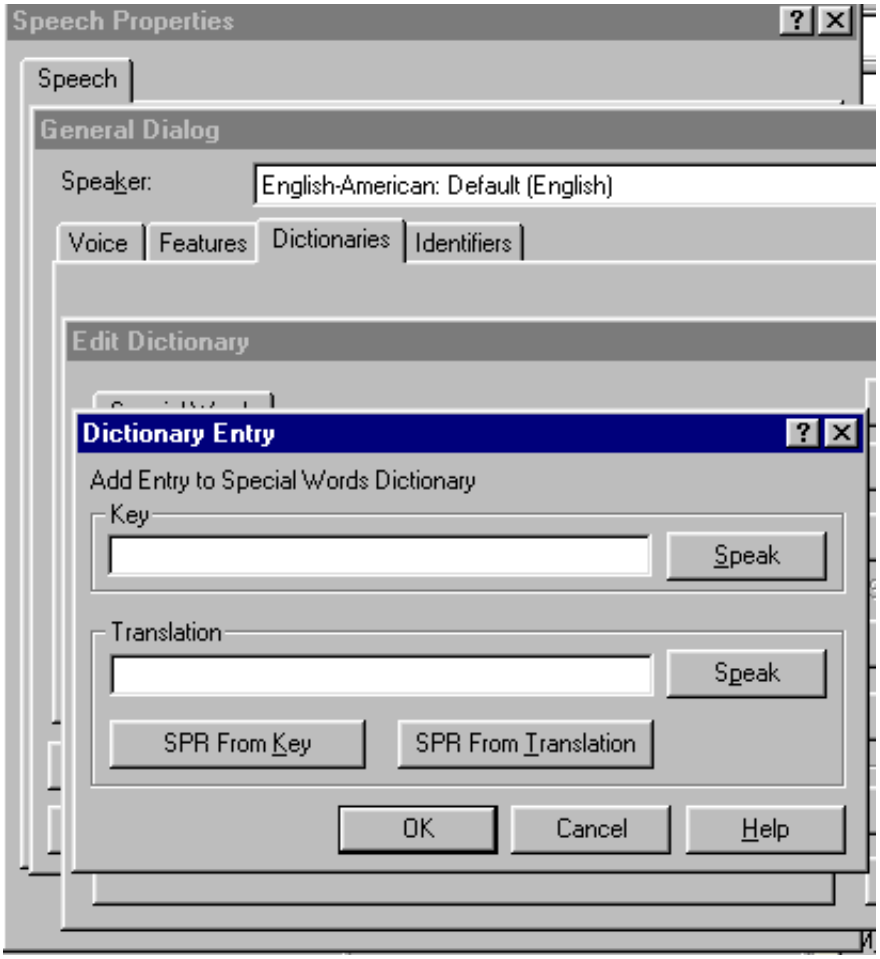
4.16 Configure synthesized speech features. (P2)

1. Provide support for user-defined extensions to the synthesized speech dictionary, as well as the following functionalities:
   - spell-out: spell text one character at a time or according to language-dependent pronunciation rules;
   - speak-numeral: speak a numeral as individual digits or as a full number; and
   - speak-punctuation: speak punctuation literally or render as natural pauses.

Note: Definitions for the functionalities listed are based on descriptions in section 19 of the Cascading Style Sheets Level 2 Recommendation *[CSS2]* ; please refer to that specification for additional informative descriptions.

Example techniques:

1.



This image shows how ViaVoice *[VIAVOICE]* allows users to add entries to the user's personal dictionary.

Who benefits:

1. Users (e.g., with blindness or low vision) who rely on audio and synthesized speech rendering.

References:

1. For information about these functionalities, please refer to descriptions in section 19.8 of Cascading Style Sheets Level 2 *[CSS2]* .

Checkpoints related to style sheets

4.17 Choose style sheets. (P1)

1. For user agents that support style sheets:
   - Allow the user to choose from and apply available author style sheets (in content).
   - Allow the user to choose from and apply available user style sheets .
   - Allow the user to ignore author and user style sheets.

Note: By definition, the user agent's default style sheet is always present, but may be overridden by author or user styles. Developers should not consider that the user's ability to turn off author and user style sheets is an effective way to improve content accessibility; turning off style sheet support means losing the many benefits they offer. Instead, developers should provide users with finer control over user agent or content behavior known to raise accessibility barriers. The user should only have to turn off author and user style sheets as a last resort.

Example techniques:

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

Who benefits:

1. Any user with a disability who needs to override the author's style sheets or user agent default style sheets in order to have control over style and presentation, or who needs to tailor the style of rendered content to meet their own needs.

References:

1. For information about how alternative style sheets are specified in HTML 4 *[HTML4]* , please refer to section 14.3.1.
2. For information about how alternative style sheets are specified in XML 1.0 *[XML]* , please refer to "Associating Style Sheets with XML documents Version 1.0" *[XMLSTYLE]* .

Guideline 5. Ensure user control of user interface behavior.

Checkpoints

5.1 No automatic content focus change. (P2)

1. Allow configuration so that if a viewport opens without explicit user request , its content focus does not automatically become the current focus .
2. Configuration is preferred, but is not required if the content focus can only ever be moved on explicit user request .

Who benefits:

1. Moving the focus automatically (and unexpectedly) to a new viewport may disorient some users with a cognitive disability, blindness, or low vision. These users may find it difficult to restore the previous point of regard.

Example techniques:

1. Allow the user to configure how the 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 alert 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.
2. If a new viewport or prompt appears but focus does not move to it, alert assistive technologies (per checkpoint 6.5) so that they may discreetly inform the user.
3. 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 return with little effort to the point of regard in the duplicate viewport. There are other techniques for accomplishing this (e.g., "registers" in Emacs).
4. In JavaScript, the focus may be changed with `myWindow.focus();`
5. 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"}
```

Doing more:

1. The user agent may also allow configuration about whether the pointing device moves automatically to windows that open without an explicit user request.

5.2 Keep viewport on top. (P2)

1. For graphical user interfaces, allow configuration so that the viewport with the current focus remains "on top" of all other viewports with which it overlaps.

Notes and rationale:

1. The alert is important to ensure that the user realizes a new viewport has opened; the new viewport may be hidden by the viewport configured to remain on top.
2. In most operating environments, the viewport with focus is generally the viewport "on top". In some environments, it's possible to allow a viewport that is not on top to have focus.

Who benefits:

1. Some users with a cognitive disability may find it disorienting if the viewport being viewed unexpectedly changes.

Doing more:

1. The user agent may also allow configuration about whether the viewport designated by the pointing device always remains on top.

5.3 Manual viewport open only. (P2)

1. Allow configuration so that viewports only open on explicit user request .
2. In this configuration, instead of opening a viewport automatically, alert the user and allow the user to open it on demand (e.g., by following a link or confirming a prompt).
3. Allow the user to close viewports.
4. If a viewport (e.g., a frame set) contains other viewports, these requirements only apply to the outermost container viewport.
5. Configuration is preferred, but is not required if viewports can only ever open on explicit user request .
6. User creation of a new viewport (e.g., empty or with a new resource loaded) through the user agent's user interface constitutes an explicit user request.

Note: Generally, viewports open automatically as the result of instructions in content . See also checkpoint 5.1 (for control over changes of focus when a viewport opens) and checkpoint 6.5 (for programmatic alert of changes to the user interface).

Who benefits:

1. Navigation of multiple open viewports may be difficult for some users who navigate viewports serially (e.g., users with visual or physical disabilities) and for some users with a cognitive disability (as it may disorient them).

Example techniques:

1. 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.
2. For SMIL *[SMIL]* , allow the user to control viewports created with the "`new`" value of the "`show`" attribute.
3. In JavaScript, windows may be opened with:
   - `myWindow.open("example.com", "My New Window");`
   - `myWindow.showHelp(URI);`

5.4 Selection and focus in viewport. (P2)

1. Ensure that when a viewport's selection or content focus changes, it is at least partially in the viewport after the change.

Note: For example, if users navigating links move to a portion of the document outside a graphical viewport, the viewport should scroll to include the new location of the focus. Or, for users of audio viewports, allow configuration to render the selection or focus immediately after the change.

Who benefits:

1. Users who may be disoriented by a change in focus or selection that is not reflected in the viewport. This includes some users with blindness or low vision, and some users with a cognitive disability.

Example techniques:

1. There are times when the content focus changes (e.g., link navigation) and the viewport should move 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.
2. If a search causes the selection or focus to change, ensure that the found content is not hidden by the search prompt.
3. When the content focus changes, register the newly focused element in the navigation sequence; sequential navigation should start from there.
4. Unless viewports have been coordinated, changes to selection or focus in one viewport should not affect the selection or focus in another viewport.
5. The persistence of the selection or focus in the viewport will vary according to

the type of viewport. For any viewport with persistent rendering (e.g., a two-dimensional graphical or tactile viewport), the focus or selection should remain in the viewport after the change until the user changes the viewport. For any viewport without persistent rendering (e.g., and audio viewport), once the focus or selection has been rendered, it will no longer be "in" the viewport. In a pure audio environment, the whole persistent context is in the mind of the user. In a graphical viewport, there is a large shared buffer of dialog information in the display. In audio, there is no such sensible patch of interaction that is maintained by the computer and accessed, ad lib, by the user. The audio rendering of content requires the elapse of time, which is a scarce resource. Consequently, the flow of content through the viewport has to be managed more carefully, notably when the content was designed primarily for graphical rendering.

6. If the rendered selection or focus does not fit entirely within the limits of a graphical viewport:
    1. if the region actually displayed prior to the change was within the selection or focus, do not move the viewport.
    2. otherwise, if the region actually displayed prior to the change was not within the newly selected or focused content, move to display at least the initial fragment of such content.

5.5 Confirm form submission. (P2)

1. Allow configuration to prompt the user to confirm (or cancel) any form submission.
2. Configuration is preferred, but it not required if forms can only ever be submitted on explicit user request .

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" or "change" event occurs.

Example techniques:

1. In HTML 4 *[HTML4]* , form submit controls are the INPUT element (section 17.4) with `type="submit"` and `type="image"`, and the BUTTON element (section 17.5) with `type="submit"`.
2. Allow the user to configure script-based submission (e.g., form submission accomplished through 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).
3. 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 elements.

4. Be aware that users may inadvertently pressing the **`Return`** or **`Enter`** key and accidentally submit a form.
5. In JavaScript, a form may be submitted with:
   - `document.form[0].submit();`
   - `document.all.mySubmitButton.click();`
6. Generate a form submit button when the author has not provided one.

Who benefits:

1. Any user who might be disoriented by an automatic form submission (e.g., users with blindness who are are navigating serially through select box options, or some users with a cognitive disability) or who might inadvertently submit a form (e.g., some users with a physical disability).

Doing more:

1. Some users may not want to have to confirm all form submissions, so allow multiple configurations, such as: confirm all form submissions; confirm script-activated form submissions; confirm all form submissions except those done through the graphical user interface (e.g., when the user moves content focus to a submit button and activates it); etc.
2. 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.
3. For forms, allow users to search for controls that need to be changed by the user before submitting the form.

5.6 Confirm fee links. (P2)

1. Allow configuration to prompt the user to confirm (or cancel) any payment that results from activation of a fee link .
2. Configuration is preferred, but is not required if fee links can only ever be activated on explicit user request .

Who benefits:

1. Any user who might inadvertently activate a fee link (e.g., some users with a physical or cognitive disability).

Example techniques:

1. Allow the user to configure the user agent to prompt for payments above a certain amount (including any payment).
2. Warn the user that even in this configuration, the user agent may not be able to recognize some payment mechanisms.

5.7 Manual viewport close only. (P3)

1. Allow configuration to prompt the user to confirm (or cancel) closing any viewport that starts to close without explicit user request .

Who benefits:

1. Some users with a cognitive disability may find it disorienting if a viewport closes automatically. On the other hand, some users with a physical disability may wish these same viewports to close automatically (rather than being required to close them manually).

Example techniques:

1. In JavaScript, windows may be closed with `myWindow.close();`

Guideline 6. Implement interoperable application programming interfaces.

Checkpoints

6.1 DOM read access. (P1)

1. Provide programmatic read access to HTML and XML content by conforming to the following modules of the W3C Document Object Model DOM Level 2 Core Specification *[DOM2CORE]* and exporting the interfaces they define:
   - the Core module for HTML;
   - the Core and XML modules for XML.

Note: Please refer to the "Document Object Model (DOM) Level 2 Core Specification" *[DOM2CORE]* for information about HTML and XML versions covered.

Notes and rationale:

1. The primary reason for requiring user agents to implement the DOM is that this gives assistive technologies access to the original structure of the document. For example, this means that assistive technologies that render content as synthesized speech are not required to construct the speech view by "reverse

engineering" a graphical view. Direct access to the structure allows the assistive technologies to render content in a manner best suited to a particular output device. This does not mean that assistive technologies should be prevented from having access to the rendering of the conforming user agent; simply that they not be required to depend entirely on it. In fact, user agents the render as synthesized speech may wish to synchronize a graphical view with a speech view.
2. 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.

Who benefits:

1. Users with a disability who rely on assistive technologies for input and output.

Example techniques:

1. Refer to a listing of DOM implementations at the Open Directory Project *[ODP-DOM]* .

Related techniques:

1. See the appendix on loading assistive technologies for DOM access .

References:

1. For information about rapid access to Internet Explorer's *[IE-WIN]* DOM through COM, refer to *[BHO]* .
2. Refer to the DirectDOM Java implementation of the DOM *[DIRECTDOM]* .

6.2 DOM write access. (P1)

1. If the user can modify HTML and XML content through the user interface , provide the same functionality programmatically by conforming to the following modules of the W3C Document Object Model DOM Level 2 Core Specification *[DOM2CORE]* and exporting the interfaces they define:
   - the Core module for HTML;
   - the Core and XML modules for XML.

Note: For example, if the user interface allows users to complete HTML forms, this must also be possible through the required DOM APIs . Please refer to the "Document Object Model (DOM) Level 2 Core Specification" *[DOM2CORE]* for information about HTML and XML versions covered.

Notes and rationale:

1. Allowing assistive technologies write access through the DOM allows them to:
   - modify the attribute list of a document and thus add information into the document object that will not be rendered by the user agent.
   - add entire nodes to the document that are specific to the assistive technologies and that may not be rendered by a user agent unaware of their function.
2. The ability to write to the DOM can improve performance for the assistive technology. For example, if an assistive technology has already traversed a portion of the document object and knows that a section (e.g., a style element) could not be rendered, it can mark this section "to be skipped".
3. Another benefit is to add information necessary for audio rendering but that would not be stored directly in the DOM during parsing (e.g., numbers in an ordered list). An assistive technology component can add numeric information to the document object. The assistive technology can also mark a subtree as having been traversed and updated, to eliminate recalculating the information the next time the user visits the subtree.

Who benefits:

1. Users with a disability who rely on assistive technologies for input and output.

Related techniques:

1. See also techniques for checkpoint 6.1.

6.3 Programmatic access to non-HTML/XML content. (P1)

1. For markup languages other than HTML and XML, provide programmatic read access to content .
2. Provide programmatic write access for those parts of content that the user can modify through the user interface. To satisfy these requirements, implement at least one API that is either
   - defined by a W3C Recommendation, or
   - a publicly documented API designed to enable interoperability with assistive technologies.
3. If no such API is available, or if available APIs do not enable the user agent to satisfy the requirements, implement at least one publicly documented API to satisfy the requirements, *and* follow operating environment conventions for the use of input and output APIs .
4. An API is considered available if the specification of the API is published (e.g., as a W3C Recommendation) in time for integration into a user agent's development cycle.

Note: This checkpoint addresses content not covered by checkpoints checkpoint 6.1 and checkpoint 6.2.

Notes and rationale:

1. Some examples of markup languages covered by this checkpoint include SGML applications other than HTML and RTF, and TeX.
2. Some software (e.g., Word and Excel for Windows) offer APIs specific to their formats.

Who benefits:

1. Users with a disability who rely on assistive technologies for input and output.

Related techniques:

1. See techniques for checkpoint 6.4.

References:

1. Some public APIs that enable access 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.

6.4 Programmatic operation. (P1)

1. Provide programmatic read access to user agent user interface controls.
2. Provide programmatic write access for those controls that the user can modify through the user interface. For security reasons, user agents are not required to allow instructions in content to modify user agent user interface controls.
3. To satisfy these requirements, implement at least one API that is either
   - defined by a W3C Recommendation, or
   - a publicly documented API designed to enable interoperability with assistive technologies.
4. If no such API is available, or if available APIs do not enable the user agent to satisfy the requirements, implement at least one publicly documented API that allows programmatic operation of all of the functionalities that are available through the user agent user interface, *and* follow operating environment conventions for the use of input and output APIs .
5. An API is considered available if the specification of the API is published (e.g., as a W3C Recommendation) in time for integration into a user agent's development cycle.

For user agent features.

Note: APIs used to satisfy the requirements of this checkpoint may be platform-independent APIs such as the W3C DOM, conventional APIs for a particular operating environment, conventional APIs for programming languages, plug-ins , virtual machine environments, etc. User agent developers are encouraged to implement APIs that allow assistive technologies to interoperate with multiple types of software in a given operating environment (user agents, word processors, spreadsheet programs, etc.), as this reuse will benefit users and assistive technology developers. User agents should always follow operating environment conventions for the use of input and output APIs.

Notes and rationale:

1. It is important to use APIs that ensure that text content is available to assistive technologies as text and not, for example, as a series of strokes drawn on the screen.

Who benefits:

1. Users with a disability who rely on assistive technologies for input and output.

Example techniques:

1. User agents that implement conventional APIs are generally more compatible with assistive technologies and provide accessibility at no extra cost.
2. Use conventional user interface controls. Third-party assistive technology developers are more likely able to access conventional controls than custom controls . If you 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 conventional controls.
3. Make use of operating environment-level features. See the appendix of accessibility features for some common operating systems.
4. Operating system and application frameworks have conventions 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 conventional mechanisms for receiving keyboard input in the case of desktop operating systems. If you use an application framework such as the Microsoft Foundation Classes, the framework used should support the same conventional input mechanisms.
5. Do not communicate directly with an input device; this may circumvent operating environment 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 conventional input mechanisms for proper application coexistence within the user interface framework.
6. Do not implement your own input device event queue mechanism; this may

circumvent operating environment messaging. Some assistive technologies use conventional 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) are one example of an application that feeds the standard event queues. For an example of a standard event queue mechanism, refer to the "Carbon Event Manager Preliminary API Reference" *[APPLE-HI]* .

7. Operating environments have conventions for communicating with output devices. In the case of common desktop operating systems such as Windows, OS/2, and Mac OS, conventional APIs are provided for writing to the display and the multimedia subsystems.
8. Avoid rendering 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 conventional operating environment 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 encourages assistive technologies to access to content directly through published APIs (such as the DOM) rather than via a particular rendering.
9. Common operating environment two-dimensional 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.
10. Do not communicate directly with an output device.
11. Do not draw directly to the video frame buffer.
12. Do not provide your own mechanism for generating pre-defined operating environment sounds.
13. When writing textual information in a GUI operating environment , use conventional operating environment APIs for drawing text.
14. Use operating environment 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-synthesized speech conversion. Also, in operating environments like Windows, a set of audio sound resources is provided to support conventional sounds such as alerts . These preset sounds are used to trigger 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 current viewport. It is important to use the conventional mechanisms to generate audio feedback so that operating environments or special assistive technologies can add additional functionality for users with hearing disabilities.

15. API designers should promote backwards compatibility so that assistive technologies do not suddenly break when a new version of an API is published and implemented by user agents.

References:

1. Some public accessibility APIs include:
   - Microsoft Active Accessibility (*[MSAA]*). This the conventional accessibility API for the Windows 95/98/NT operating systems.
   - Sun Microsystems Java Accessibility API (*[JAVAAPI]*) in the Java JDK. This is the conventional accessibility API for the Java environment. 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.
2. For information about rapid access to Internet Explorer's *[IE-WIN]* DOM through COM, refer to Browser Helper Objects *[BHO]* .

6.5 Programmatic alert of changes. (P1)

1. Provide programmatic alert of changes to content , user interface controls, selection , content focus , and user interface focus .
2. To satisfy these requirements, implement at least one API that is either
   - defined by a W3C Recommendation, or
   - a publicly documented API designed to enable interoperability with assistive technologies.
3. If no such API is available, or if available APIs do not enable the user agent to satisfy the requirements, implement at least one publicly documented API to satisfy the requirements, *and* follow operating environment conventions for the use of input and output APIs .
4. An API is considered available if the specification of the API is published (e.g., as a W3C Recommendation) in time for integration into a user agent's development cycle.

For both content and user agent.

Note: For instance, when user interaction in one frame causes automatic changes to content in another, provide a programmatic alert. This checkpoint does not require the user agent to alert the user of *rendering changes* caused by content (e.g., an animation effect or an effect caused by a style sheet), just changes to the content itself.

Who benefits:

1. Users with a disability who rely on assistive technologies for output.

Example techniques:

1. Write output to and take input from conventional operating environment APIs rather than directly from hardware controls. This will enable the input/output 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.
2. Alert the user when an action in one frame causes the content of another frame to change. Allow the user to navigate with little effort to the frame(s) that changed.

Related techniques:

1. See techniques for checkpoint 6.4.

References:

1. Refer to "mutation events" in "Document Object Model (DOM) Level 2 Events Specification" *[DOM2EVENTS]* . This DOM Level 2 specification allows assistive technologies to be informed of changes to the document tree.
2. Refer also to information about monitoring HTML events through the document object model in Internet Explorer *[IE-WIN]* .

6.6 Conventional keyboard APIs. (P1)

1. Follow operating environment conventions when implementing APIs for the keyboard .
2. If such APIs for the keyboard do not exist, implement publicly documented APIs for the keyboard.

Note: An operating environment may define more than one conventional API for the keyboard. For instance, for Japanese and Chinese, input may be processed in two stages, with an API for each.

Who benefits:

1. Users with a disability who rely on assistive technologies for input.

Example techniques:

1. Account for author-specified keyboard bindings, such as those specified by "accesskey" attribute in HTML 4 (*[HTML4]* , section 17.11.2).
2. 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.

Related techniques:

1. Apply the techniques for checkpoint 1.1 to the keyboard.

Doing more:

1. Enhance the functionality of conventional operating environment controls to improve accessibility where none is provided by responding to conventional 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 conventional keyboard activation facility for Abstract Window Toolkit components.

6.7 API character encodings. (P1)

1. For an API implemented to satisfy requirements of this document, support the character encodings required for that API.

For both content and user agent.

Note: Support for character encodings is important so that text is not "broken" when communicated to assistive technologies. For example, the DOM Level 2 Core Specification *[DOM2CORE]* , section 1.1.5 requires that the `DOMString` type be encoded using UTF-16. This checkpoint is an important special case of the other API requirements of this document.

Who benefits:

1. Users with disabilities who rely on assistive technologies for input and output.

Example techniques:

1. The list of character encodings that any conforming implementation of Java version 1.3 *[JAVA13]* must support is: US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, and UTF-16.
2. MSAA *[MSAA]* relies on the COM interface, which in turn relies on Unicode *[UNICODE]* , which means that for MSAA a user agent must support UTF-16. From Chapter 3 of the COM documentation, on interfaces, entitled "Interface

Binary Standard":

> Finally, and quite significantly, all strings passed through all COM interfaces (and, at least on Microsoft platforms, all COM APIs) are Unicode strings. There simply is no other reasonable way to get interoperable objects in the face of (i) location transparency, and (ii) a high-efficiency object architecture that doesn't in all cases intervene system-provided code between client and server. Further, this burden is in practice not large."

6.8 DOM CSS access. (P2)

1. For user agents that implement Cascading Style Sheets (CSS), provide programmatic access to those style sheets in content by conforming to the CSS module of the W3C Document Object Model (DOM) Level 2 Style Specification *[DOM2STYLE]* and exporting the interfaces it defines.
2. For the purposes of satisfying this checkpoint, Cascading Style Sheets (CSS) are defined by either CSS Level 1 *[CSS1]* or CSS Level 2 *[CSS2]* .

Note: Please refer to the "Document Object Model (DOM) Level 2 Style Specification" *[DOM2STYLE]* for information about CSS versions covered.

Who benefits:

1. Users with a disability who rely on assistive technologies for input and output.

Related techniques:

1. See techniques for

6.9 Timely access. (P2)

1. Ensure that programmatic exchanges proceed in a timely manner.

For both content and user agent.

Note: For example, the programmatic exchange of information required by other checkpoints in this document should 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. Timely exchange is also important for the proper synchronization of alternative renderings. The techniques for this checkpoint explain how developers can reduce communication delays. This will help ensure that assistive technologies have timely access to the document object model and other information that is important for providing access.

Notes and rationale:

1. This document requires that a conforming user agent provide access to content and user interface information through APIs because assistive technologies must be able to respond incrementally to changes in the user's session. Simply providing a "text dump" of content to an assistive technology, for example, would make it extremely difficult for assistive technologies to provide timely access (as the assistive technology would have to recalculate much more information rather than having information about incremental changes).

Who benefits:

1. Users with a disability who rely on assistive technologies for input and output.

Related techniques:

1. Please see the appendix that explains how to load assistive technologies for DOM access .

Doing more:

1. Alert the user when information may be lost due to communication delays.

Guideline 7. Observe operating environment conventions.

Checkpoints

7.1 Focus and selection conventions. (P1)

1. Follow operating environment conventions that benefit accessibility when implementing the selection , content focus , and user interface focus .

Note: This checkpoint is an important special case of checkpoint 7.3. See also checkpoint 9.1 and checkpoint 9.2.

Who benefits:

1. Many users with many types of disabilities.

Related techniques:

1. See techniques for checkpoint 7.3.

References:

1. Refer to Selection and Partial Selection of DOM Level 2 (*[DOM2RANGE]* , section 2.2.2).
2. For information about focus in the Motif environment (under X Windows), refer to the OSF/Motif Style Guide *[MOTIF]* .

7.2 Respect input configuration conventions. (P1)

1. Ensure that default input configurations of the user agent do not interfere with operating environment accessibility conventions (e.g., for keyboard accessibility).

For user agent features.

Note: See also checkpoint 11.5.

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. The default configuration should not include "**`Alt-F4`**", "**`Control-Alt-Delete`**", or other combinations that have reserved meanings in a given operating environment.
2. Clearly document any default configurations that depart from operating environment conventions.

Related techniques:

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

7.3 Operating environment conventions. (P2)

1. Follow operating environment conventions that benefit accessibility. In particular, follow conventions that benefit accessibility for user interface design, keyboard configuration, product installation, and documentation .
2. For the purposes of this checkpoint, an operating environment convention that benefits accessibility is either
   - one identified as such in operating environment design or accessibility guidelines, or
   - one that allows the author to satisfy any requirement of the "Web Content Accessibility Guidelines 1.0" *[WCAG10]* or of the current document.

For user agent features.

Note:

Notes and rationale:

1. Much of the rationale behind the content requirements of User Agent Accessibility Guidelines 1.0 also makes sense for the user agent user interface (e.g., allow the user to turn off any blinking or moving user interface components).

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. Follow operating environment conventions for loading assistive technologies. See 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.
2. Inherit operating environment settings related to accessibility (e.g., for fonts, colors, natural language preferences, input configurations, etc.).
3. Ensure that any online services (e.g., automated update facilities, download-and-install functionalities, sniff-and-fill forms, etc.) observe relevant operating environment conventions concerning device independence and accessibility (as well as the Web Content Accessibility Guidelines 1.0 *[WCAG10]*).
4. Evaluate the conventional interface controls on the target platform against any built-in operating environment accessibility functions (see 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 environment 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 when implementing new features. However, developers should make changes conservatively to the layout of user interface controls , the behavior of existing functionalities, and the default keyboard configuration.

Related techniques:

1. See techniques for checkpoint 6.4 and checkpoint 7.2.

References:

1. Follow accessibility guidelines for specific platforms:
   - "GNOME Accessibility Project" *[GNOME-ACCESS]*
   - "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]* .
2. Follow general guidelines for producing accessible software:
   - "Accessibility for applications designers" *[MS-ENABLE]* .
   - "Application Software Design Guidelines" *[TRACE-REF]* . Refer also to "EZ ACCESS(tm) for electronic devices V 2.0 implementation guide" *[TRACE-EZ]* from the Trace Research and Development Center.
   - Articles and papers from Sun Microsystems about 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]* .

7.4 Input configuration indications. (P2)

1. Follow operating environment conventions to indicate the input configuration .

For user agent features.

Note: For example, in some operating environments, when a functionality may be triggered through a menu and through the keyboard, the developer may design the menu entry so that the character of the activating key is also shown. This checkpoint is an important special case of checkpoint 7.3. See also checkpoint 11.5.

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. Use operating environment 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 key such as **`Alt`**, etc.) These are conventions used by the Sun Java Foundations Classes *[JAVA-TUT]* and Microsoft Foundations Classes for Windows.
2. 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).
3. If the current configuration changes locally (e.g., a search prompt opens, changing the keyboard mapping for the duration of the prompt), alert the user.
4. 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 meaningful 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.

Related techniques:

1. See input configuration techniques .

Guideline 8. Implement specifications that benefit accessibility.

Checkpoints

8.1 Implement accessibility features. (P1)

1. Implement the accessibility features of specifications (markup languages, style sheet languages, metadata languages, graphics formats, etc.). For the purposes of this checkpoint, an accessibility feature is either
   - one identified as such, or
   - one that allows the author to satisfy any requirement of the "Web Content Accessibility Guidelines 1.0" *[WCAG10]* .

For all content.

Note: This checkpoint applies to both W3C-developed and non-W3C specifications. conformance and implementing specifications for more information.

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. Make obvious to users features that are known to benefit accessibility. Make them easy to find in the user interface and in documentation.
2. Some specifications include optional features (not required for conformance to the specification). If an optional feature is likely to cause accessibility problems, developers should either ensure that the user can turn off the feature or they not implement the feature.
3. Refer to the following list of accessibility features of HTML 4 *[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 interactive elements such as links and form elements.

References:

1. Refer to the "Accessibility Features of CSS" *[CSS-ACCESS]* . Note that CSS 2 includes properties for configuring synthesized speech styles.
2. Refer to the "Accessibility Features of SMIL" *[SMIL-ACCESS]* .
3. Refer to the "Accessibility Features of SVG" *[SVG-ACCESS]* .
4. For information about the Sun Microsystems Java Accessibility API in Java JDK, refer to *[JAVAAPI]* .
5. For information about captioning for the Synchronized Accessible Multimedia Interchange (SAMI), refer to *[SAMI]* .

8.2 Conform to specifications. (P2)

1. Use and conform to either
   - W3C Recommendations when they are available and appropriate for a task, or
   - non-W3C specifications that enable the creation of content that conforms at

level A or better to the Web Content Accessibility Guidelines 1.0 *[WCAG10]* .

2. When a requirement of another specification contradicts a requirement of the current document, the user agent may disregard the requirement of the other specification and still satisfy this checkpoint.
3. A specification is considered available if it is published (e.g., as a W3C Recommendation) in time for integration into a user agent's development cycle.

For all content.

Note: For instance, for markup, the user agent may conform to HTML 4 *[HTML4]* , XHTML 1.0 *[XHTML10]* , or XML 1.0 *[XML]* . For style sheets, the user agent may conform to CSS (*[CSS1]* , *[CSS2]*). For mathematics, the user agent may conform to MathML 2.0 *[MATHML20]* . For synchronized multimedia, the user agent may conform to SMIL 1.0 *[SMIL]* . The user agent is not required to satisfy this checkpoint for all implemented specifications; see the section on conformance and implementing specifications for more information.

Notes and rationale:

1. The right to disregard only applies when the requirement of another specification contradicts the requirements of the current document; no exemption is granted if the other specification is consistent with or silent about a requirement made by the current document.
2. Conformance to W3C Recommendations is not a Priority 1 requirement because user agents can (and should!) provide access for non-W3C specifications as well.
3. The requirement of this checkpoint is to conform to *at least* one W3C Recommendation that is available and appropriate for a particular task, or at least one non-W3C specification that allows the creation of content that conforms to WCAG 1.0 *[WCAG10]* . For example, user agents would satisfy this checkpoint by conforming to 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. Each specification defines what conformance means for that specification.

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. If more than one version or level of a specification is appropriate for a particular task, user agents are encouraged to conform to the latest version. However, developers should consider implementing the version that best supports accessibility, even if this is not the latest version.
2. For reasons of backward compatibility, user agents should continue to implement deprecated features of specifications. Information about deprecated

language features is generally part of the language's specification.

References:

1. The list of current W3C Recommendations and other technical documents is available at `http://www.w3.org/TR/`.
2. W3C make available validation services to promote the proper usage and implementation of specifications. Refer to the:
   - HTML (including XHTML), and XML validator service *[VALIDATOR]* .
   - CSS validator service *[CSSVALIDATOR]* .
3. Information about PDF and accessibility is made available by Adobe *[ADOBE]* .

Guideline 9. Provide navigation mechanisms.

Checkpoints

9.1 Provide content focus. (P1)

1. Provide at least one content focus for each viewport (including frames) where enabled elements are part of the rendered content .
2. Allow the user to make the content focus of each viewport the current focus .

Note: For example, when two frames of a frameset contain enabled elements, allow the user to make the content focus of either frame the current focus. Note that viewports "owned" by plug-ins that are part of a conformance claim are also covered by this checkpoint.

Who benefits:

1. Users who rely on the content focus for interaction (e.g., for interaction with enabled elements through the keyboard, or for assistive technologies that consider the current focus a point of regard). This includes some users with blindness, low vision, or a physical disability.

Example techniques:

1. None.

9.2 Provide user interface focus. (P1)

1. Provide a user interface focus .

Who benefits:

1. Users who rely on the user interface focus for interaction (e.g., for interaction with user interface controls through the keyboard, or for assistive technologies that consider the current focus a point of regard). This includes some users with blindness, low vision, or a physical disability.

Example techniques:

1. Some operating environments provide a means to move the user interface focus 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.

9.3 Move content focus. (P1)

1. Allow the user to move the content focus to any enabled element in the viewport .
2. Allow configuration so that the content focus of a viewport only changes on explicit user request . Configuration is not required if the content focus only ever changes on explicit user request. See also checkpoint 5.1.
3. If the author has not specified a navigation order, allow at least forward sequential navigation to each element, in document order.
4. The user agent may also include disabled elements in the navigation order.

Note: In addition to forward sequential navigation, the user agent should also allow reverse sequential navigation. This checkpoint is an important special case of checkpoint 9.9.

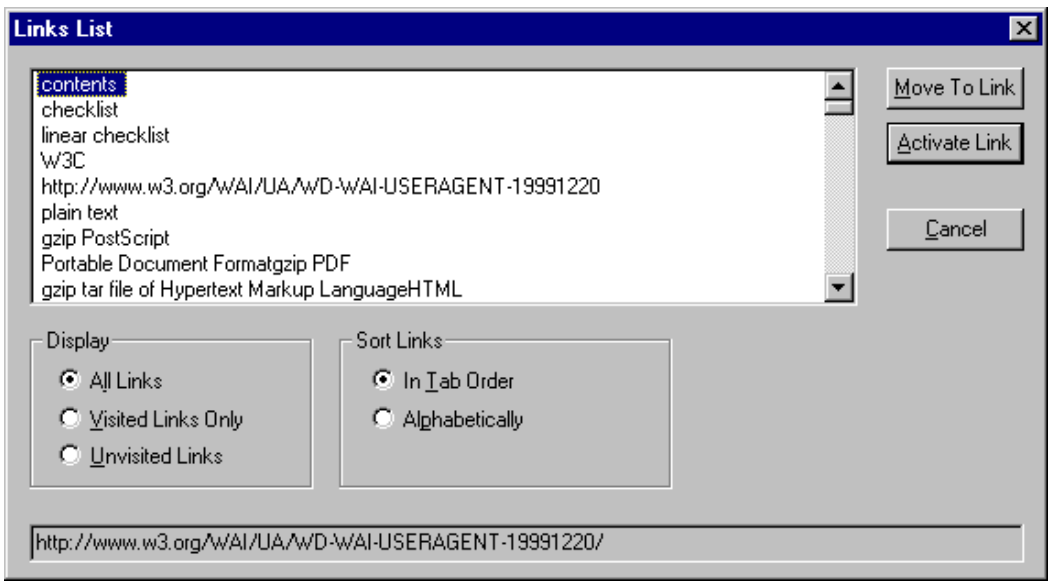
Who benefits:

1. Users who rely on the focus for interaction (e.g., for interaction with enabled elements through the keyboard, or for assistive technologies that consider the focus a point of regard). This includes some users with blindness, low vision, or a physical disability.

Example techniques:

1. Allow the user to move the content focus to each enabled element 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. See also checkpoint 11.3.
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.).

3. 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).
4. 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.
5. 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:" }').
6. 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.
7. 

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 main view.

Doing more:

1. 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, see techniques for checkpoint 5.5.
2. For graphical user agents (or any user agent offering a two-dimensional display), navigation based not on document order but on layout may also benefit the user. For example, allow the user to navigate up, down, left, and right to the nearest rendered enabled link. This type of navigation may be particularly useful when it is clear from the layout where the next navigation step will take the user (e.g., grid layouts where it is clear what the next link to the left or below will be).
3. 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 element text, associated labels, or form element names).

9.4 Restore history. (P1)

1. For user agents that implement a viewport history mechanism, for each state in a viewport's browsing history, maintain information about the point of regard , content focus , and selection .
2. When the user returns to any state in the viewport history, restore the saved values for these three state variables.

Note: For example, when the user uses the "back button", restore the point of regard, content focus, and selection for previous state in the viewport's history.

Notes and rationale:

1. This checkpoint only refers to a per-viewport history mechanism, not a history mechanism that is common to all viewports (e.g., of visited Web resources).

Who benefits:

1. Users who may have difficulty re-orienting themselves during a browsing session. This includes some users with a memory or cognitive disability, some users with a physical disability, and some users who access content serially and for whom repositioning will be time consuming (e.g., users with blindness or low vision).

Example techniques:

1. 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.
2. Allow the user to configure whether leaving a viewport pauses a multimedia presentation.
3. 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.
4. In JavaScript, the following may be used to change the Web resource in the viewport, and navigate the history:
   - `myWindow.home();`
   - `myWindow.forward();`
   - `myWindow.back();`
   - `myWindow.navigate("http://example.com/");`
   - `myWindow.history.back();`
   - `myWindow.history.forward();`
   - `myWindow.history.go( -2 );`
   - `location.href = "http://example.com/"`
   - `location.reload();`
   - `location.replace("http://example.com/");`

Doing more:

1. Restore the four state variables after the user refreshes the same content.

References:

1. Refer to the HTTP/1.1 specification for information about history mechanisms (*[RFC2616]* , section 13.13).

9.5 No events on focus change. (P2)

1. Allow configuration so that moving the content focus to or from an enabled element does not automatically activate any explicitly associated event handlers .

Note: For instance, in this configuration for an HTML document, do not activate any handlers for the '`onfocus`', '`onblur`', or '`onchange`' attributes. In this configuration, user agents should still apply any stylistic changes (e.g., highlighting) that may occur when there is a change in content focus .

Notes and rationale:

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

Who benefits:

1. Users with blindness or some users with a physical disability, and anyone without a pointing device.

Example techniques:

1. Allow the following configurations:
    1. On invocation of the input binding, move focus to the associated enabled element, but do not activate it.
    2. On invocation of the input binding, move focus to the associated enabled 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 enabled element and activate it.

9.6 Show event handlers. (P2)

1. For the element with content focus , make available the list of input device event handlers explicitly associated with the element.

Note: For example, allow the user to query the element with content focus for the list of input device event handlers, or add them directly to the serial navigation order described in checkpoint 9.3. See checkpoint 1.2 for information about activation of event handlers associated with the element with focus.

Who benefits:

1. Users with blindness or some users with a physical disability, and anyone without a pointing device.

Example techniques:

1. For HTML content, the left mouse button is generally the only mouse button that is used to activate event handlers associated with mouse clicks.

References:

1. See checkpoint 1.2 for information about input device event handlers in HTML 4 *[HTML4]* and the Document Object Model (DOM) Level 2 Events Specification *[DOM2EVENTS]* .

9.7 Move content focus optimally. (P2)

1. Allow the user to move the content focus to any enabled element in the viewport .
2. If the author has not specified a navigation order, allow at least forward and reverse sequential navigation to each element, in document order.
3. The user agent must not include disabled elements in the navigation order.

Note: This checkpoint is a special case of checkpoint 9.3.

Who benefits:

1. Users who rely on the focus for interaction (e.g., for interaction with enabled elements through the keyboard, or for assistive technologies that consider the focus a point of regard). This includes some users with blindness, low vision, or a physical disability.

Related techniques:

1. Apply the techniques of checkpoint 9.3 to enabled elements only.

Doing more:

1. Allow configuration so that disabled elements are included in the navigation order. These elements cannot be activated (as they are disabled), but their presence may lend continuity to navigation.

9.8 Text search. (P2)

1. Allow the user to search within rendered text for a sequence of characters from the document character set .
2. Allow the user to start a forward search (in document order) from any selected or focused location in content.
3. When there is a match do both of the following:
   - move the viewport so that the matched text content is within it, and
   - allow the user to search for the next instance of the text from the location of the match.
4. Alert the user when there is no match, when the search reaches the end of content, and prior to any wrapping. A wrapping search is one that restarts automatically at the beginning of content once the end of content has been

reached.
5. Provide a case-insensitive search option for text in scripts (i.e., writing systems) where case is significant.

For all rendered content.

Note: If the user has not indicated a start position for the search, the search should start from the beginning of content. Per checkpoint 7.3, use operating environments conventions for indicating the result of a search (e.g., selection or content focus).

Who benefits:

1. Some users who access content serially (e.g., users with blindness or low vision), some users with a cognitive disability (who may have difficulty locating information among other information), and some users with a physical disability (for whom navigation may be a significant effort).

Example techniques:

1. Use the selection or focus to indicate found text. This will provide assistive technologies with access to the text.
2. Allow users to search all views (e.g., including views of the text source).
3. For extremely small viewports or extremely long matches, the entire matched text content may not fit within the viewport. In this case, developers may move the viewport to encompass the initial part of the matched content.
4. The search string input method should follow operating environment conventions (e.g., for international character input).
5. When the point of regard depends on time (e.g., for audio viewports), the user needs to be able to search through content that will be available through that viewport. This is analogous to content rendered graphically that is reachable by scrolling.
6. For frames, allow users to search for content in all frames, without having to be in a particular frame.
7. 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.
8. Allow users to search the element content of form elements (where applicable) and any label text.
9. When searching a document, the user agent should not search text whose properties prevent it from being visible (such as text that has `visibility="hidden"`), or equivalent text for elements with such properties (such as "`alt`" text for an image that has `visibility="hidden"`).

Doing more:

1. If the number of matches is known, provide this information to orient the user.
2. 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, alert the user of this particular search mode.
3. Allow the following additional search functionalities:
    1. Allow the user to start a search from the beginning of the document rather than from the current selection or focus.
    2. Provide distinct alerts for the situation where the user has searched through all content or where the user has simply reached the end of the document and needs to wrap to the beginning.
    3. Allow reverse search so the user doesn't not have to start he search from the beginning of the document if the search goes too far.
    4. Allow the user to easily start a search from the beginning of the content currently rendered in the viewport.
    5. Provide the option of searching through conditional content that is associated with rendered content, and render the found conditional content (e.g., by showing its relation to the rendered content).

References:

1. For information about when case is significant in a script , please refer to Section 4.1 of Unicode *[UNICODE]* .

9.9 Structured navigation. (P2)

1. Allow the user to navigate efficiently to and among important structural elements in rendered content .
2. Allow forward and backward sequential navigation to these important structural elements.

Note: This specification intentionally does not identify which "important elements" must be navigable as this will vary according to markup language. What constitutes "efficient navigation" may depend on a number of factors as well, including the "shape" of content (e.g., serial navigation of long lists is not efficient) and desired granularity (e.g., among tables, then among the cells of a given table).

Who benefits:

1. Users who access content serially, including users with blindness and some users with a physical disability.

Notes and rationale:

1. 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. (The navigation view may also be an expanding/contracting outline view; see checkpoint 10.5.) Instead, user agents are expected to construct the navigation view based on markup.

Example techniques:

1. In HTML 4 *[HTML4]* , important elements include: `A`, `ADDRESS`, `APPLET`, `BUTTON`, `FIELDSET`, `DD`, `DIV`, `DL`, `DT`, `FORM`, `FRAME`, `H1–H6`, `IFRAME`, `IMG`, `INPUT`, `LI`, `LINK` (if rendered), `MAP`, `OBJECT`, `OL`, `OPTGROUP`, `OPTION`, `P`, `TABLE`, `TEXTAREA`, and `UL`. HTML also allows authors to specify keyboard configurations ("accesskey", "tabindex"), which can serve as hints about what the author considers important.
2. Allow navigation based on commonly understood document models, even if they do not adhere strictly to a Document Type Definition (DTD) or schema. 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.
3. Use the DOM (*[DOM2CORE]*) 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.
4. Follow operating environment conventions for indicating navigation progress (e.g., selection or content focus).
5. 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. See the section on table navigation .
6. 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.). See also checkpoint 1.3.
7. For those languages with known (e.g., by specification, schema, metadata, etc.) conventions for identifying important components, user agents should construct the navigation tree from those components, allowing users to navigate up and down the document tree, and forward and backward among siblings. As the same time, allow users to shrink and expand portions of the document tree. For instance, if a subtree consists of a long series of links, this will pose problems for users with serial access to content. At any level in the document tree (for forward and backward navigation of siblings), limit the number of siblings to between five and ten. Break longer lists down into structured pieces so that users can access content efficiently, decide whether they want to explore it in

detail, or skip it and move on.

8. 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 a 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.
9. Navigation and orientation go together. The user agent should allow the user to navigate to a location in content, explore the context, navigate again, etc. In particular, 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 facilitate orientation at the same time.
10. 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.
11. 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. If the number of elements of a particular type is known, provide this information to orient the user.
    3. 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, the MAP element allows block content, not just AREA elements.
12. Allow depth-first as well as breadth-first navigation.
13. Allow users to navigate synchronized multimedia presentations. See also checkpoint 4.5.

Doing more:

1. Allow the user to navigate characters, words, sentences, paragraphs, screenfuls, etc. according to conventions of the natural language . This benefits users of synthesized speech-based user agents and has been implemented by several screen readers, including Winvision *[WINVISION]* , Window-Eyes *[WINDOWEYES]* , and JAWS for Windows *[JFW]* .

References:

1. The following is a summary of ideas provided by the National Information Standards Organization with respect to Digital Talking Books *[TALKINGBOOKS]* :

   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 efficiently, 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.

9.10 Configure important elements. (P3)

1. Allow configuration of the set of important elements required by checkpoint 9.9 and checkpoint 10.5.
2. Allow the user to include and exclude element types in the set of elements.

Note: For example, allow the user to navigate only paragraphs, or only headings and paragraphs, or to suppress and restore navigation bars, to navigate within and among tables and table cells, etc.

Who benefits:

1. Users who access content serially, including users with blindness and some users with a physical disability.

Example techniques:

1. Allow the user to navigate HTML elements that share the same "class" attribute.
2. 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 .

Doing more:

1. Allow the user to navigate according to similar styles (which may be an approximation for similar element types).

Guideline 10. Orient the user.

Checkpoints

10.1 Table orientation. (P1)

1. Make available to the user the purpose of each rendered table (e.g., as expressed in a summary or table caption) and the relationships among the table cells and headers.

Note: This checkpoint refers only to table purpose and cell/header relationship information that the user agent can recognize. Depending on the table, some techniques may be more efficient than others for conveying data relationships. For many tables, user agents rendering in two dimensions may satisfy this checkpoint by rendering a table as a 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. This checkpoint is an important special case of checkpoint 2.1.

Notes and rationale:

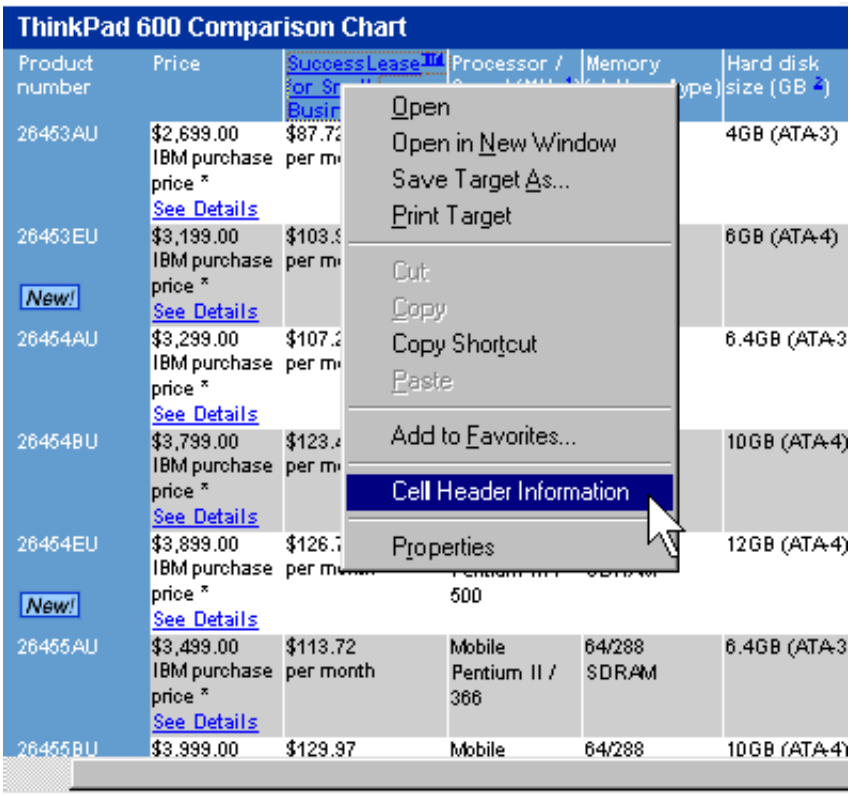
1. The more complex the table, the more clues to table structure are needed. Make available information summarizing table structure, including any table head and foot rows, and possible row grouping into multiple table bodies, column groups, header cells and how they relate to data cells, the grouping and spanning of rows and columns that apply to qualify any cell value, cell position information, table dimensions, etc.

Who benefits:

1. Users for whom summaries are important (e.g., some users with a cognitive or memory disability), and for whom two-dimensional relationships may be difficult to process (e.g., users with blindness who have serial access to the content, or some users with a cognitive disability). Renderings that provide easy access to cell header information will also help some users with low vision or a physical disability, for whom it may be time-consuming to scroll in order to locate relevant headers.

Example techniques:

1. 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".
2. 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, see the section on table techniques .
3. 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).
4.



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

10.2 Highlight selection and content focus. (P1)

1. Provide a mechanism for highlighting the selection and content focus of each viewport.
2. The highlight mechanism must not rely on color alone.
3. Allow global configuration of selection and focus highlight styles.
4. For graphical viewports, if the highlight mechanism involves colors or text decorations , offer a range of colors or text decorations to the user that includes at least:
   - the range offered by the conventional utility available in the operating environment that allows users to choose colors or text decorations,
   - or, if no such utility is available, the range of colors or text decorations supported by the conventional APIs of the operating environment for specifying colors or drawing text.

Note: Examples of highlight mechanisms include foreground and background color variations, underlining, distinctive synthesized speech prosody, border styling, etc. Because the selection and focus change frequently, user agents should not highlight them using mechanisms (e.g., font size variations) that cause content to reflow as this may disorient the user. See also checkpoint 7.1.

Who benefits:

1. Users with color deficiencies or blindness, for whom color will not be useful. Also, some devices may not render colors (e.g., speech synthesizers, black and white screens).

Example techniques:

1. Inherit selection and focus information from user's settings for the operating environment .
2. A highlighted selection or focus may span text with different background colors, text foreground colors, font families, etc.
3. For selection:
   - As a 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).
4. For focus, implement the ':hover', ':active', and ':focus' pseudo-classes of CSS 2 (*[CSS2]* , section 5.11.3). and dynamic outlines and focus of CSS 2 (*[CSS2]* , sections 5.11.3 and 18.4.1, respectively).

   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 }
   ```

Doing more:

1. Test the user agent to ensure that individuals who have low vision and use screen magnification software are able to follow highlighted item(s).

Related techniques:

1. For Windows, see information about `ChooseFont` and `ChooseColor` in techniques for checkpoint 4.1, checkpoint 4.2, and checkpoint 4.3. `ChooseFont` is also used to choose some text decorations in Windows.

10.3 Distinct default highlight styles. (P1)

1. Ensure that all of the default highlight styles for the selection and content focus , as well as for enabled elements , recently visited links, and fee links in rendered content :
   - do not rely on color alone, and
   - differ from each other, and not by color alone.
2. This checkpoint does not apply to those highlight styles inherited from the operating environment as default values, as long as the user can change the styles in the operating environment.

Note: For instance, by default a graphical user agent may present the selection using color and a dotted outline, the focus using a solid outline, enabled elements as underlined in blue, recently visited links as dotted underlined in purple, and fee links using a special icon or flag to draw the user's attention.

Who benefits:

1. For this checkpoint, and for others in this document, "color" includes black, white, and greys.
2. Users with color deficiencies or blindness, for whom color will not be useful. Also, some devices may not render colors (e.g., speech synthesizers, black and white screens).

Example techniques:

1. If the user overrides the default styling for any one of these mechanisms, the new styling may interfere with the others. Therefore, the user agent should allow the user to configure them all at once or should alert the user to potential conflicts when change are made. For instance, if the user configures both the selection and focus highlighting to use colors, there may be a conflict (especially if the colors are the same or similar).
2. If default highlight styles are inherited from the operating environment, document how to change them, or explain where to find this information in the documentation for the operating environment.

10.4 Highlight special elements. (P2)

1. Provide a mechanism for highlighting all enabled elements , recently visited links, and fee links in rendered content .
2. Allow the user to configure the highlight styles. The highlight mechanism must not rely on color alone.
3. For graphical viewports, if the highlight mechanism involves text size, font family, colors, or text decorations , offer the corresponding range of values required by checkpoint 4.1, checkpoint 4.2, checkpoint 4.3, or checkpoint 10.2.
4. For a graphically rendered enabled elements, highlight the most specific

rendered element that:

- encompasses the enabled element, and
- is rendered as a coherent unit according to specification.

For example, an HTML user agent rendering a PNG image as part of an image map is only required to highlight the image as a whole, not each enabled region. On the other hand, an SVG user agent rendering an SVG image with embedded graphical links is required to highlight each graphical link that may be rendered independently according to the SVG specification.

Note: Examples of highlight mechanisms include foreground and background color variations, font variations, underlining, distinctive synthesized speech prosody, border styling, etc.

Notes and rationale:

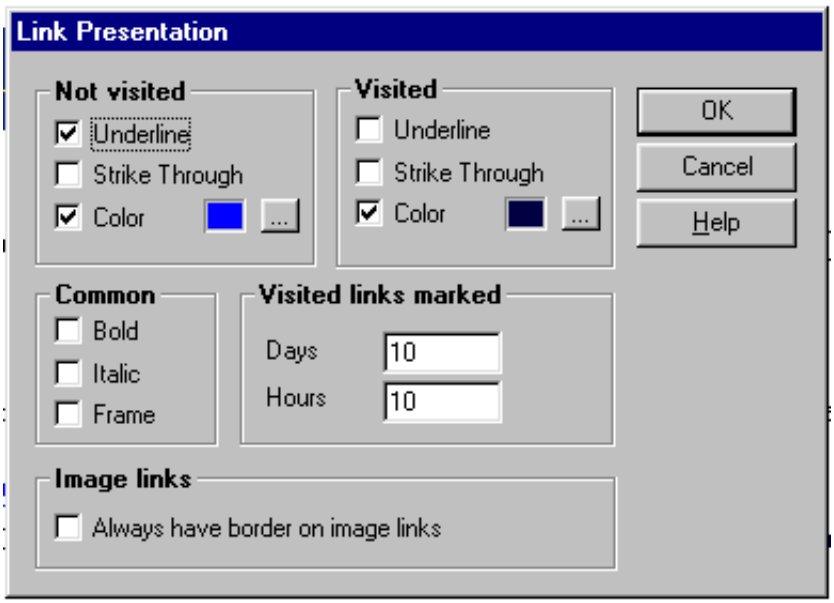
1. For example, most graphical user agents highlight all the links on a page so that users know at a glance where to interact.

Who benefits:

1. Users with color deficiencies or blindness, for whom color will not be useful. Also, some devices may not render colors (e.g., speech synthesizers, black and white screens). If different text styles are used, some users with low vision may need to configure them.

Example techniques:

1. 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.).
2. Use CSS2 *[CSS2]* to add style to these different classes of elements. In particular, consider the 'text-decoration' property (*[CSS2]* , section 16.3.1), aural cascading style sheets, font properties, and color properties.
3. For enabled elements, implement CSS2 attribute selectors to match elements with associated scripts (*[CSS2]* , section 5.8).
4. For fee links:
   - The W3C specification "Common Markup for micropayment per-fee-links" *[MICROPAYMENT]* describes how authors may mark up micropayment information in an interoperable manner.
   - Use conventional, 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).
5.



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

Related techniques:

1. For links, see the section on link techniques , the visited links example in the section on generated content techniques , and techniques for checkpoint 9.3.

Doing more:

1. Test the user agent to ensure that individuals who have low vision and use screen magnification software are able to follow highlighted item(s).

10.5 Outline view. (P2)

1. Make available to the user an "outline" view of content , composed of labels for important structural elements (e.g., heading text, table titles, form titles, etc.).
2. What constitutes a label is defined by each markup language specification. A label is not required to be text only.

Note: This checkpoint is meant to provide the user with a simplified view of content (e.g, a table of contents). For example, in HTML, a heading (`H1`-`H6`) is a label for the section that follows it, a `CAPTION` is a label for a table, the "`title`" attribute is a label for its element, etc. For important elements that do not have associated labels, user agents may generate labels for the outline view. For information about what constitutes the set of important structural elements, please see the Note following checkpoint 9.9. By making the outline view navigable, it is possible to satisfy this checkpoint and checkpoint 9.9 together: allow users to navigate among the important elements of the outline view, and to navigate from a

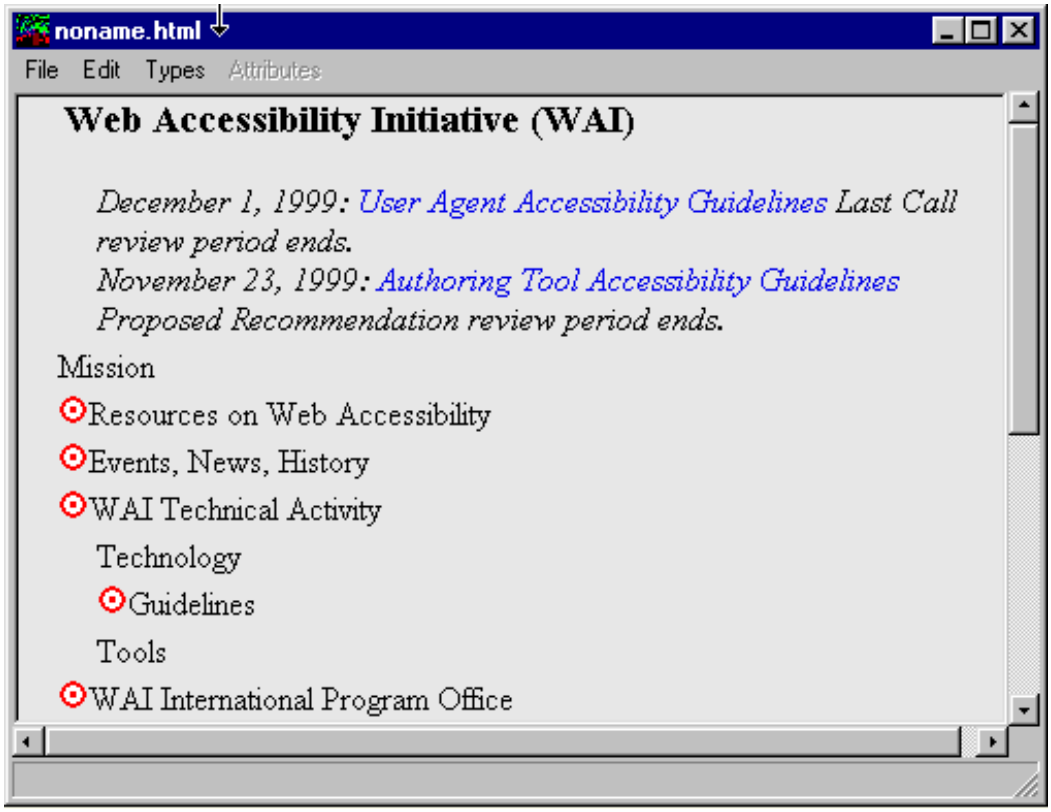
position in the outline view to the corresponding position in a full view of content. See also checkpoint 9.10.

Who benefits:

1. The outline view is a type of summary view, and will benefit some users with a memory or cognitive disability, as well as users for whom serial access is time consuming (e.g., some users with blindness or a physical disability, or some users with low vision). A navigable outline view will add further benefits for these users.

Example techniques:

1. For instance, in HTML, labels include the following:
   - The `CAPTION` element is a label for `TABLE`
   - The "`title`" attribute is a label for many elements.
   - The `H1-H6` elements are labels for sections that follow
   - The `LABEL` element is a label for form element
   - The `LEGEND` element is a label for a set of form elements
   - The `TH` element is a label for a row/column of table cells.
   - The `TITLE` element is a label for the document.
2. Allow the user to expand or shrink portions of the outline view (configure detail level) for faster access to important parts of content.
3. Hide portions of content by using the CSS 'display' and 'visibility' properties (*[CSS2]* , sections 9.2.5 and 11.2, respectively).
4. Provide a structured view of form controls (e.g., those grouped by `LEGEND` or `OPTGROUP` in HTML) along with their labels.
5.



This image shows the table of contents view provided by Amaya *[AMAYA]* . This view is coordinated with the main 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.

Related techniques:

1. See structured navigation techniques for checkpoint 9.9.

Doing more:

1. 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.

10.6 Provide link information. (P3)

1. To help the user decide whether to traverse a link, make available the following information about it:
   - link element content,
   - link title,
   - whether the link is internal to the resource (e.g., the link is to a target in the

same Web page),
- whether the user has traversed the link recently,
- whether traversing it may involve a fee, and
- information about the type, size, and natural language of linked Web resources.

2. The user agent is not required to compute or make available information that requires retrieval of linked Web resources .

Who benefits:

1. Users for whom following a link may lead to loss of context upon return, including some users with blindness and low vision, a memory or cognitive disability, or a physical disability.

Example techniques:

1. Some markup languages allow authors to provide hints about the nature of linked content (e.g., in HTML 4 *[HTML4]* , the "hreflang" and "type" attributes on the A element). Specifications should indicate when this type of information is a hint from the author and when these hints may be overridden by another mechanism (e.g., by HTTP headers in the case of HTML). User agent developers should make the author's hints available to the user (prior to retrieving a resource), but should provide definitive information once available.
2. Links may be simple (e.g., HTML links) or more complex, such as those defined by the XML Linking Language (XLink) *[XLINK]* .
3. The scope of "recently followed link" depends on the user agent. The user agent may allow the user to configure this parameter, and should allow the user to reset all links as "not followed recently".
4. User agents should cache information determined as the result of retrieving a Web resource and should make it available to the user. Refer to HTTP/1.1 caching mechanisms described in RFC 2616 *[RFC2616]* , section 13.
5. 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).
6. Do not mark all local links (to anchors in the same page) as visited when the page has been visited.

Related techniques:

1. See the section on link techniques .

Doing more:

1. User agents may provide information about any input bindings associated with a link. See checkpoint 11.2.

References:

1. User agents may use HTTP HEAD rather than GET for information about size, language, etc. Refer to RFC 2616 *[RFC2616]* , section 9.3
2. For information about content size in HTTP/1.1, refer to RFC 2616 *[RFC2616]* , section 14.13. User agents are not expected to compute content size recursively (i.e., by adding the sizes of resources referenced by URIs within another resource).
3. For information about content language in HTTP/1.1, refer to RFC 2616 *[RFC2616]* , section 14.12.
4. For information about content type in HTTP/1.1, refer to RFC 2616 *[RFC2616]* , section 14.17.

Checkpoints for the user interface

10.7 Highlight current viewport. (P1)

1. Provide a mechanism for highlighting the viewport with the current focus (including any frame that takes current focus).
2. For graphical viewports, the default highlight mechanism must not rely on color alone.
3. This default color requirement does not apply if the highlight mechanism is inherited from the operating environment as the default and the user can change it in the operating environment.

Note: This checkpoint is an important special case of checkpoint 1.1. See also to checkpoint checkpoint 7.1.

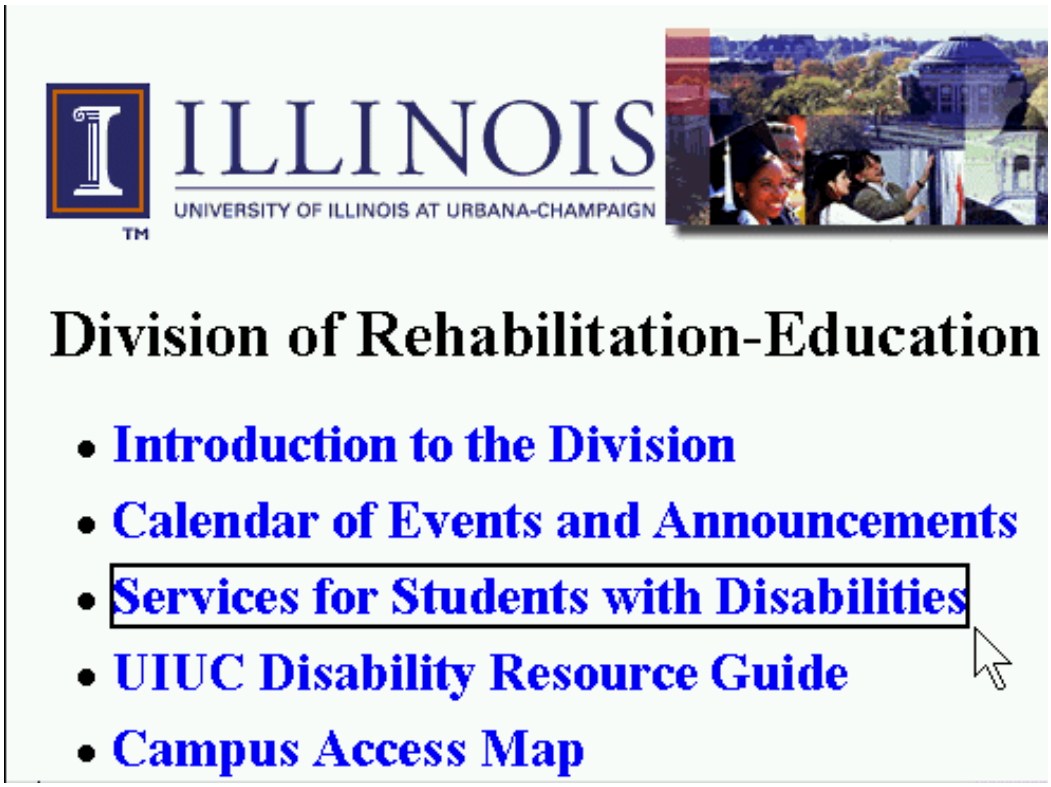
Who benefits:

1. Users with color deficiencies or blindness, for whom color will not be useful. Also, some devices may not render colors (e.g., speech synthesizers, black and white screens).

Example techniques:

1. Provide a setting that causes a window that is the viewport with the current focus 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 viewport so that users have to scroll horizontally or vertically as little as possible.
2. If the viewport with the current focus 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 viewport with the current focus.
3. If the default highlight mechanism is inherited from the operating environment, document how to change it, or explain where to find this information in the

documentation for the operating environment.

4. For synthesized speech or braille output, use the frame or window title to identify the viewport with the current focus.
5. Use operating environment conventions, for specifying selection and content focus (e.g., schemes in Windows).
6. 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 .
7. 

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 operating environment highlight colors to indicate content focus.

Related techniques:

1. See the section on frame techniques .

10.8 Indicate rendering progress. (P3)

1. Indicate the viewport's position relative to rendered content (e.g., the proportion of an audio or video clip that has been played, the proportion of a Web page that has been viewed, etc.).
2. The user agent may calculate the relative position according to content focus position, selection position, or viewport position, depending on how the user has been browsing.
3. For two-dimensional renderings, relative position includes both vertical and horizontal positions.
4. The user agent may indicate the proportion of content viewed in a number of ways, including as a percentage, as a relative size in bytes, etc.

Notes and rationale:

1. This checkpoint does not specify how to calculate the proportion in all cases, and implementations may vary. For instance, suppose a user agent is to render fifty audio clips one after the other. It may be costly to calculate the proportion based on the total time required by all fifty clips (as this may require the user agent to fetch all fifty in advance). Instead, the user agent may represent the proportion as something like "2:43 remaining in the tenth audio clip (of fifty)."

Who benefits:

1. This type of context information benefits everyone, but is particularly valuable to some users with serial access to content (e.g., users with blindness) and to some users with a cognitive disability.

Example techniques:

1. The proportion should be indicated using a relative value (where applicable), otherwise as an absolute offset from some recognized landmark.
2. Provide a scrollbar for the viewport. Some specifications address scrolling requirements or suggestions, such as for the `THEAD` and `TBODY` elements of HTML 4 (*[HTML4]* , section 11.2.3) and the 'overflow' property of CSS 2 (*[CSS2]* , section 11.1.1).
3. 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.
4. Indicate the number of screens of information, based on the current dimensions of the viewport (e.g., "screen 4 of 10").
5. Use a variable pitch audio signal to indicate the viewport's different positions.
6. Provide markers for specific percentages through the document.
7. Provide markers for positions relative to some position – a user selected point, the bottom, the `H1`, etc.
8. 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).
9. For images that render gradually (coarsely to finely), it is not necessary to show percentages for each rendering pass.

Doing more:

1. 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 element group (e.g., `FIELDSET` in HTML).

Guideline 11. Allow configuration and customization.

Checkpoints

11.1 Current user bindings. (P1)

1. Provide information to the user about current user preferences for input configurations .
2. To satisfy this checkpoint, the user agent may make available binding information in a centralized fashion (e.g., a list of bindings) or a distributed fashion (e.g., by listing keyboard shortcuts in user interface menus).

For user agent features.

Who benefits:

1. Many users benefit from direct access to important user agent functionalities (e.g., via a single key stroke or short voice command): users with blindness (for whom the pointing device is not useful), users with poor physical control (who might mistakenly repeat a key stroke), users who fatigue easily (for whom the composition of key sequences is a significant effort), users who cannot remember key combinations, and any user who wants to operate the user agent efficiently.

Related techniques:

1. See the input configuration techniques .

11.2 Current author bindings. (P2)

1. Provide a centralized view of the current author-specified input configuration bindings.
2. The user agent may satisfy this checkpoint by providing different views for different input modalities (keyboard, pointing device, voice, etc.).

For all content.

Note: For example, for HTML documents, provide a view of keyboard bindings specified by the author through the "`accesskey`" attribute. The intent of this checkpoint is to centralize information about author-specified bindings so that the user does not have to read the entire content first to find out what bindings are available.

Who benefits:

1. Refer to checkpoint 11.2: some users with blindness, a physical disability, or a memory or cognitive disability.

Example techniques:

1. If the user agent offers a special view that lists author-specified bindings, allow the user to navigate easily back and forth between the viewport with the current focus and the list of bindings.

Related techniques:

1. See input configuration techniques .

Doing more:

1. In addition to providing a centralized view of bindings, allow users to find out about bindings in content. For example, highlight enabled elements that have associated event handlers (e.g., by indicating bindings near the element).

11.3 Override bindings. (P2)

1. Allow the user to override any binding that is part of the user agent default input configuration .
2. The user agent is not required to allow the user to override conventional bindings for the operating environment (e.g., for access to help).
3. The override requirement only applies to bindings for the same input modality (e.g., the user must be able to override a keyboard binding with another keyboard binding).

For user agent features.

Note: See also checkpoint 11.5, checkpoint 11.7, and checkpoint 12.3.

Who benefits:

1. Refer to checkpoint 11.2: some users with blindness, a physical disability, or a memory or cognitive disability.

Example techniques:

1. Allow the user to override bindings at the level of the operating environment.

Related techniques:

1. See input configuration techniques .

Doing more:

1. Allow users to choose 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.
2. Allow users to restore easily the default input configuration.
3. Allow users to create macros and bind them to key strokes or other input methods.
4. Test the default keyboard configuration for usability. Ask users with different disabilities and combinations of disabilities to test configurations.

11.4 Single key access. (P2)

1. Allow the user to override any binding in the user agent default keyboard configuration with a binding to either a key plus modifier keys or to a single-key. In this checkpoint, "key" refers to a physical key of the keyboard (rather than, say, a character of the document character set).
2. For each functionality in the set required by checkpoint 11.5, allow the user to configure a single-key binding (i.e., one key press performs the task, with zero modifier keys).
3. If the number of physical keys on the keyboard is less than the number of functionalities required by checkpoint 11.5, allow single-key bindings for as many of those functionalities as possible.
4. The single-key binding requirements may be satisfied with a "single-key mode" (i.e., a mode where the current bindings are replaced by a set of single-key bindings).
5. The user agent is not required to allow the user to override conventional bindings for the operating environment (e.g., for access to help).
6. This checkpoint does not require single physical key bindings for character input, only for the activation of user agent functionalities.

For user agent features.

Note: Because single-key access is so important to some users with physical disabilities, user agents should ensure that (1) most keys of the physical keyboard may be configured for single-key bindings, and (2) most functionalities of the user agent may be configured for single-key bindings. For information about access to user agent functionality through a keyboard API, see checkpoint 6.6.

Notes and rationale:

1. When using a physical keyboard, some users require single-key access, others require that keys activated in combination be physically close together, while others require that they be spaced physically far apart.
2. In some modes of interaction (e.g., when the user is entering text), the number of available single keys will be significantly reduced.
3. A "single-key mode" allows user agents to "save" keys for other bindings by default and still satisfy this checkpoint. However, even when a single-key mode is offered, user agents should include as many required single-key bindings as possible in the default keyboard configuration. The user should be able to enter into a single-key mode by using a single-key.

Who benefits:

1. Single-key access is particularly important to some users with a physical disability, or a memory or cognitive disability (for simplicity's sake).

Example techniques:

1. Offer a single-key mode where, once the user has entered into that mode (e.g., by pressing a single key), most of the keys of the keyboard are configurable for single-key operation of the user agent. Allow the user to exit that mode by pressing a single key as well. For example, Opera *[OPERA]* includes a mode in which users can access important user agent functionalities with single strokes from the numeric keypad.
2. 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 activate 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).
3. Mouse Keys (available in some operating environments) 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.

Doing more:

1. 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.
2. 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).

11.5 Default binding requirements. (P2)

1. Ensure that the user agent default input configuration includes bindings for the following functionalities required by other checkpoints in this document:
   - move focus to next enabled element , and move focus to previous enabled element;
   - activate focused link;
   - search for text;
   - search again for same text;
   - increase size of rendered text , and decrease size of rendered text;
   - increase global volume, and decrease global volume;
   - stop, pause, resume, and navigate efficiently selected audio and animations (including video and animated images).
2. If the user agent supports the following functionalities, the default input configuration must also include bindings for them:
   - next history state (forward), and previous history state (back);
   - enter URI for new resource;
   - add to favorites (i.e., bookmarked resources);
   - view favorites;
   - stop loading resource;
   - reload resource;
   - refresh rendering;
   - forward one viewport, and back one viewport;
   - next line, and previous line.

For user agent features.

Note: This checkpoint does not make any requirements about the ease of use of default input configurations, though clearly the default configuration should include single-key bindings and allow easy operation. Ease of use is ensured by the configuration requirements of checkpoint 11.3.

Who benefits:

1. Refer to checkpoint 11.2: some users with blindness, a physical disability, or a memory or cognitive disability.

Example techniques:

1. 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.

Related techniques:

1. See the techniques for checkpoint 7.4

Doing more:

1. Provide different input configuration profiles (e.g., one keyboard profile with key combinations close together and another with key combinations far apart).
2. Offer a mode that makes the input configuration compatible with other versions of the software (or with other software).
3. Provide convenient bindings for controlling the user interface, such as showing, hiding, moving, and resizing graphical viewports .
4. Allow the user to configure how much the viewport should move when scrolling the viewport backward or forward through content (e.g., for a graphical viewport, "page down" causes the viewport to move half the height of the viewport, or the full height, or twice the height, etc.).

11.6 User profiles. (P2)

1. For the configuration requirements of this document, allow the user to save user preferences in at least one user profile .
2. Allow the user to choose from among available default profiles, profiles created by the same user, and no profile (i.e., the user agent default settings).

For user agent features.

Notes and rationale:

1. The user agent is only expected to allow the user to choose from profiles created by the same user, not profiles created by other users.

Who benefits:

1. Refer to checkpoint 11.2: some users with blindness, a physical disability, or a memory or cognitive disability.

Example techniques:

1. Follow applicable operating environment conventions for input configuration profiles .
2. Allow users to choose a different profile, to switch rapidly between profiles, and to return to the default input configuration.
3. If the user can edit the profile by hand, the user agent documentation should explain the profile format.

11.7 Configure tool bars. (P3)

1. For graphical user interfaces, allow the user to configure the position of controls on tool bars of the user agent user interface , to add or remove controls for the user interface from a predefined set, and to restore the default user interface.

For user agent features.

Note: This checkpoint is a special case of checkpoint 11.3.

Who benefits:

1. Users for whom serial navigation may be difficult (e.g,. some users with blindness or a physical disability). Some users with a memory or cognitive disability (who may have difficulty remembering where and how to access user agent functionalities).

Example techniques:

1. Use conventional operating environment controls for allowing configuration of font sizes, synthesized speech rates, and other style parameters.
2. Allow the user to show and hide controls. This benefits users with cognitive disabilities and users who navigate user interface controls sequentially.
3. Allow the user to choose icons and/or text.
4. Allow the user to change the grouping of icons and the order of menu entries (e.g., for faster access to frequently used controls).
5. Allow multiple icon sizes (big, small, other sizes). Ensure that these values are applied consistently across the user interface.
6. 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 12. Provide accessible user agent documentation and help.

Checkpoints

12.1 Accessible documentation. (P1)

1. Ensure that at least one version of the user agent documentation conforms to at least Level Double-A of the Web Content Accessibility Guidelines 1.0 *[WCAG10]* .

For user agent features.

Notes and rationale:

1. User agents may provide documentation in many formats, but at least one must conform to at least Level Double-A of the Web Content Accessibility Guidelines 1.0 *[WCAG10]* .
2. Remember to keep documentation accessible as the user agent evolves (e.g., when bug fixes are published, etc.).

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. Distribute accessible documentation over the Web, on CD-ROM, or by telephone. Alternative hardcopy formats may also benefit some users.
2. For example, for conformance to the Web Content Accessibility Guidelines 1.0 *[WCAG10]* :
    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 synthesized 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. See also checkpoint 12.3.
3. Describe the user interface with device-independent terms. For example, use "select" instead of "click on".
4. 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.
5. 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 environment conventions for input configurations related to run-time help.
6. Ensure that user agent identification codes are accessible to users so they may install their software. Codes printed on software packaging may not be accessible to people with visual disabilities.

Doing more:

1. 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.
2. 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.

12.2 Document accessibility features. (P1)

1. Document all user agent features that benefit accessibility.
2. For the purposes of this checkpoint, a user agent feature that benefits accessibility is one implemented to satisfy the requirements of this document (including the requirements of checkpoints 8.1 and 7.3).
3. The user agent may satisfy this checkpoint either by
   - providing a centralized view of the accessibility features, or
   - integrating accessibility features into the rest of the documentation.

For user agent features.

Note: The help system should include discussion of user agent features that benefit accessibility. The user agent should satisfy this checkpoint by providing both centralized and integrated views of accessibility features in the documentation.

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. Document any features that affect accessibility and that depart from system conventions.
2. 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 support this feature by turning off scripts, but users should not have to guess (or know) that turning off scripts will turn off blinking text.
3. Document configurable features in addition to defaults for those features.
4. Document the features implemented to conform with these guidelines.
5. Include references to accessibility features in both the table of contents and index of the documentation.
6. If configuration files are used to satisfy the requirements of this document, the documentation should explain the configuration file formats.

7. In developer documentation, document the APIs that are required by this document. Please see the requirements of guideline 6.

12.3 Document default bindings. (P1)

1. Document the default user agent input configuration (e.g., the default keyboard bindings).

For user agent features.

Note: If the default input configuration is inconsistent with conventions of the operating environment, the documentation should alert the user.

Notes and rationale:

1. Documentation of keyboard accessibility is particularly important to users with visual disabilities and some types of physical disabilities. Without this documentation, a user with a disability (or multiple disabilities) may not think that a particular task can be performed. Or the user may try to use a much less efficient technique to perform a task, such as using a mouse, or using an assistive technology's mouse emulation through key strokes.

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. If the user agent inherits default values (e.g,. for the input configuration, for highlight styles, etc.) from the operating environment, document how to modify them in the operating environment, or explain where to find this information in the documentation for the operating environment.

References:

1. As an example of online documentation of keyboard support, refer to the Mozilla Keyboard Planning FAQ and Cross Reference for the Mozilla browser *[MOZILLA]* .

12.4 Document changes. (P2)

1. Document changes from the previous version of the user agent to accessibility features, including accessibility features of the user interface.
2. Accessibility features are those defined in checkpoint 12.2.

For user agent features.

Who benefits:

1. Many users with many types of disabilities.

Notes and rationale:

1. In particular, document changes to the user interface.

Example techniques:

1. Either describe the changes that affect accessibility in the section of the documentation dedicated to accessibility features (see checkpoint 12.5) or link to the changes from the dedicated section.
2. Provide a text description of changes (e.g., in a README file).

12.5 Dedicated section on accessibility. (P2)

1. Provide a centralized view of all features of the user agent that benefit accessibility in a dedicated section of the documentation .
2. The features that benefit accessibility are those defined in checkpoint 12.2.

For user agent features.

Note: The user agent satisfies this checkpoint automatically by providing a centralized view of accessibility features to satisfy checkpoint 12.2. However, developers are encouraged to integrate descriptions of accessibility features into the documentation alongside other features, in addition to providing a centralized view.

Who benefits:

1. Many users with many types of disabilities.

Example techniques:

1. 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.
2. Ensure that the section on accessibility features is easy to find.

---

2 Accessibility topics

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

2.1 Access to content

User agents need to 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 (what am I examining?);
- Using and generating metadata to provide context (where am I?).

These topics are addressed below.

2.1.1 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., synthesized speech and braille) also need to make those relationships apparent, otherwise users may not understand the purpose of the table and the relationships among its cells (see the section on table techniques). User agents need to 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 may be defined by a Document Type Definition (DTD) or a schema, 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.

2.1.2 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 efficiently 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 has to 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 and opens an attribute menu that shows which attributes are available for the element and which have been assigned values. The user may read or write values to attributes (since Amaya is an editor as well as a browser). 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 rendered text that may cross "element boundaries".

2.1.3 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. See also the section on table techniques , which explains how user 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 (see 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.

2.2 User control of rendering and style

To ensure accessibility, users need to 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 need to allow users to increase the size of rendered 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.

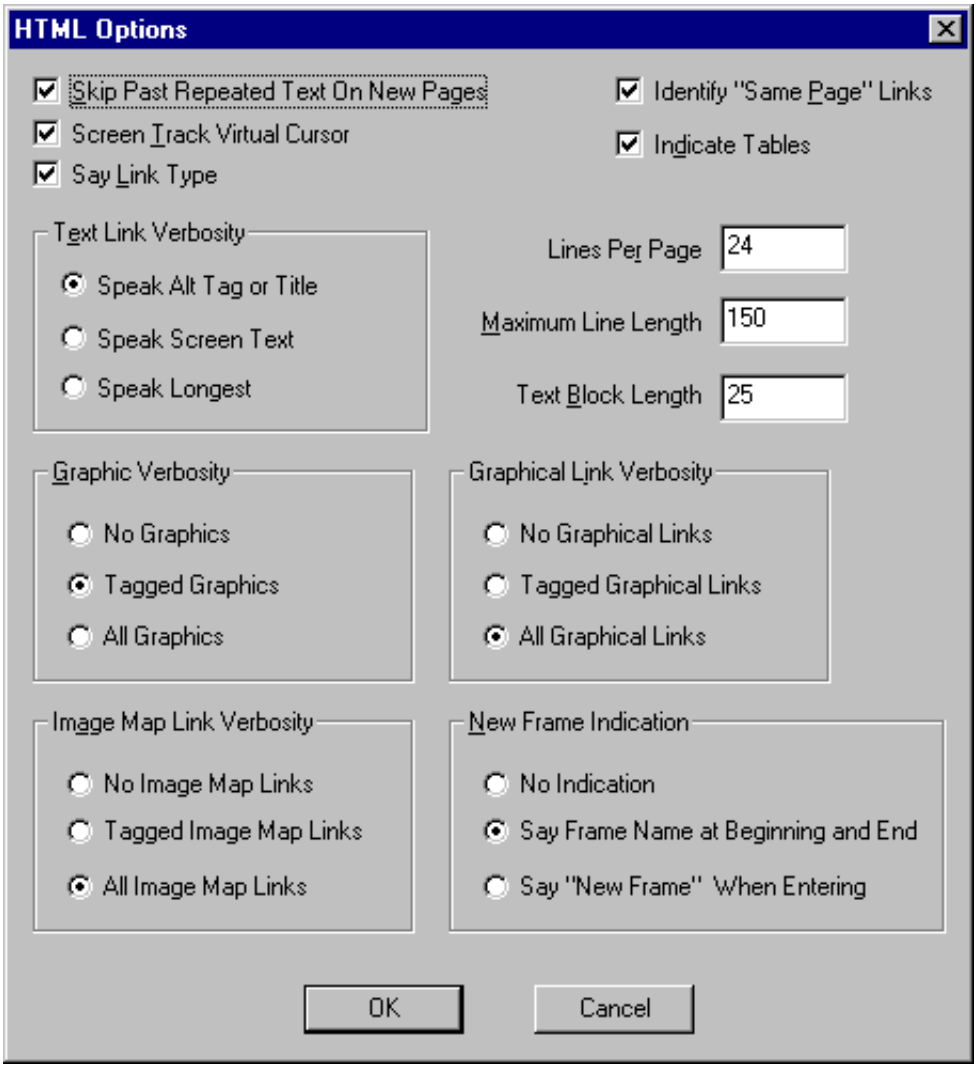
- User agents should implement the cascade order of CSS 2 (*[CSS2]* , section 6.4.1) not CSS 1. In CSS 2, user style sheets with "!important" declarations (section 6.4.2) take precedence over author styles . Refer also to Web Content Accessibility Guidelines 1.0 checkpoint 3.3 *[WCAG10]* .
- CSS-enabled 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.
- In JavaScript, the following may be used to change style information:
  ```
  document.all.myElement style.color = "red";
  ```

2.3 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:

- See 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 example, here is an algorithm for ensuring that an HTML link that has image content has associated text.
    1. If the author has specified conditional content (that is not empty) for the image (e.g., "alt" in HTML), use that as the link text;
    2. Otherwise, use the link title if available;
    3. [Repair] Otherwise, use title information of the designated Web resource (e.g., the TITLE element of HTML for links to HTML documents).
    4. [Repair] Otherwise, render part of the filename or URI of the designated Web resource .
    5. [Repair] Otherwise, insert a generic text placeholder (e.g., [LINK]) in place of the image (if configured to do so).
- For an image in link content, ensure that the user has access to the link and any long description associated with the image.



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.

2.4 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 synthesized 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.

2.5 Table techniques

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

- Users who access a table serially (e.g., as synthesized speech or braille) may have difficulty grasping the relationships among cells, especially for large and complex tables.
- Users 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 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 content through an API (see checkpoint 6.3).

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

2.5.1 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.

2.5.2 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. A linear rendering of a table may be useful as an equivalent for a multi-dimensional table.

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 specifies abbreviated headers for synthesized speech and other rendering (*[HTML4]* , section 11.2.6). See also 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]* .

2.5.3 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 (*[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.

2.5.4 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 *[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 element's 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.

2.5.5 Cell header repair strategies

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:

- Consider `TH` cells on both the left and right of the table.
- For `TH` cells with "rowspan" set: consider the content of those `TH` cells for each of the N-1 rows below the one containing that `TH` content.
- An internal `TH` surrounded by `TD`s 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` element with "`colspan`" set needs to be included in the list of `TH` s for the N-1 columns to its right.

2.5.6 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 within specific rows or columns, row groups or column groups, or limited by associated headers.
- Alert 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]* .

2.6 Image map techniques

One way to make an image map accessible to some users (e.g., users with blindness) is to render the links it contains as text links. This allows assistive technologies to render the links as synthesized 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 (not empty) conditional text content is available (the "alt" attribute), use it as the content of a generated link.
- When the author has specified empty conditional text content ("`alt=''`"), do not render the link.
- When the author has specified no text equivalent (no "alt"), render "Map area" (or similar) 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 conditional text content 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.

2.7 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:

- Equivalents 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 need to be able to navigate from frame to frame in a device independent manner.
- Orientation. Users need to know what frame they are in (so, for example, authors should provide a title for each frame), 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 conditional content related to frames (e.g., provided by the HTML 4 NOFRAMES element (*[HTML4]* , section 16.4.1).
- Here is a technique for the case of a frameset that does not contain a `NOFRAMES` element 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.
- Alert 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 frameset with the current focus (e.g., by using a thick border, by displaying the name of the frameset in the status bar, etc.)
- Allow the user to query the frame with the current focus 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 probably indicates a title, more words may indicate an index, many words may indicate a paragraph.
- Allow navigation between frames (forward and backward through the nested structure, return to a top-level list of links to frames). **Note:** Recall that the user needs to be able to navigate frames through all supported input devices.
- Alert the user when an action in one frame causes the content of another frame to change. Allow the user to navigate with little effort to the frame(s) that changed.
- Authors can suppress scrolling of HTML frames with `scrolling="no"`. In this case, the user agent needs to make available content that is not in the viewport.
- The user agent may ignore some attributes of the FRAME element of HTML 4 (*[HTML4]* , section 16.2.2): "`noresize`", "`scrolling`", and "`frameborder`".

Consider renderings of the following document:

Example.

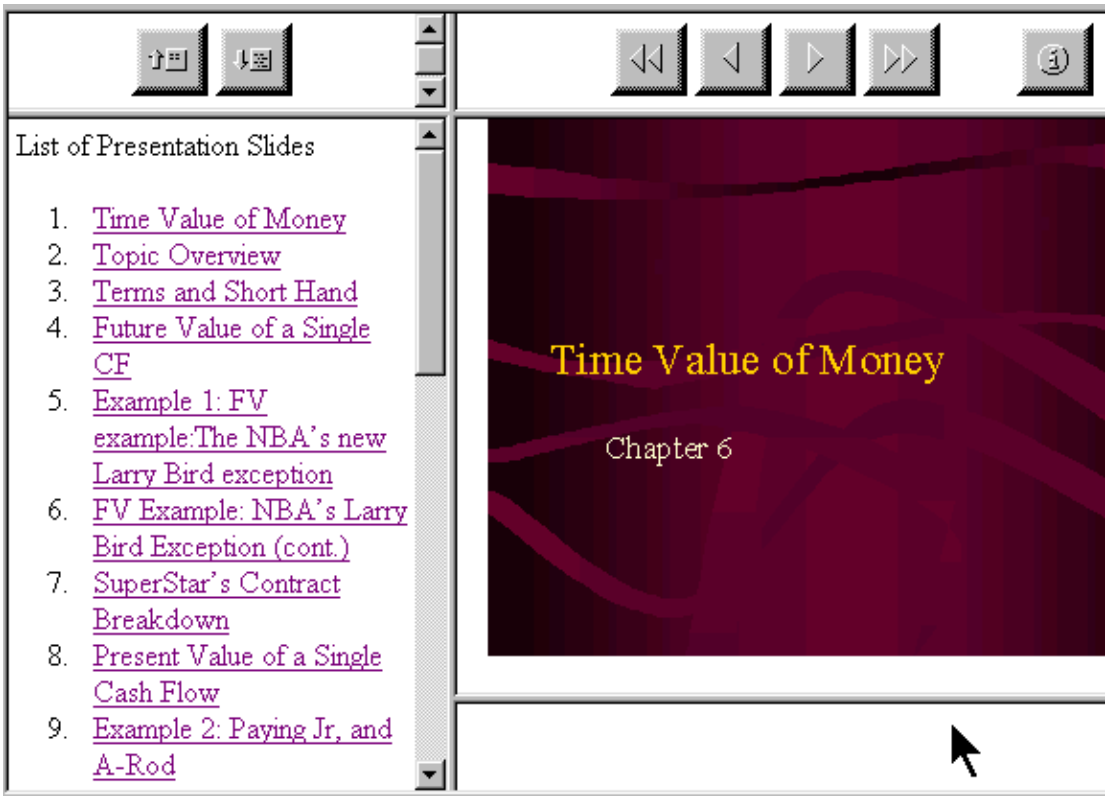
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML lang="en">
<HEAD>
  <META http-equiv="Content-Type"
           content="text/html; charset=iso-8859-1">
  <TITLE>Time Value of Money</TITLE>
</HEAD>

<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]* .

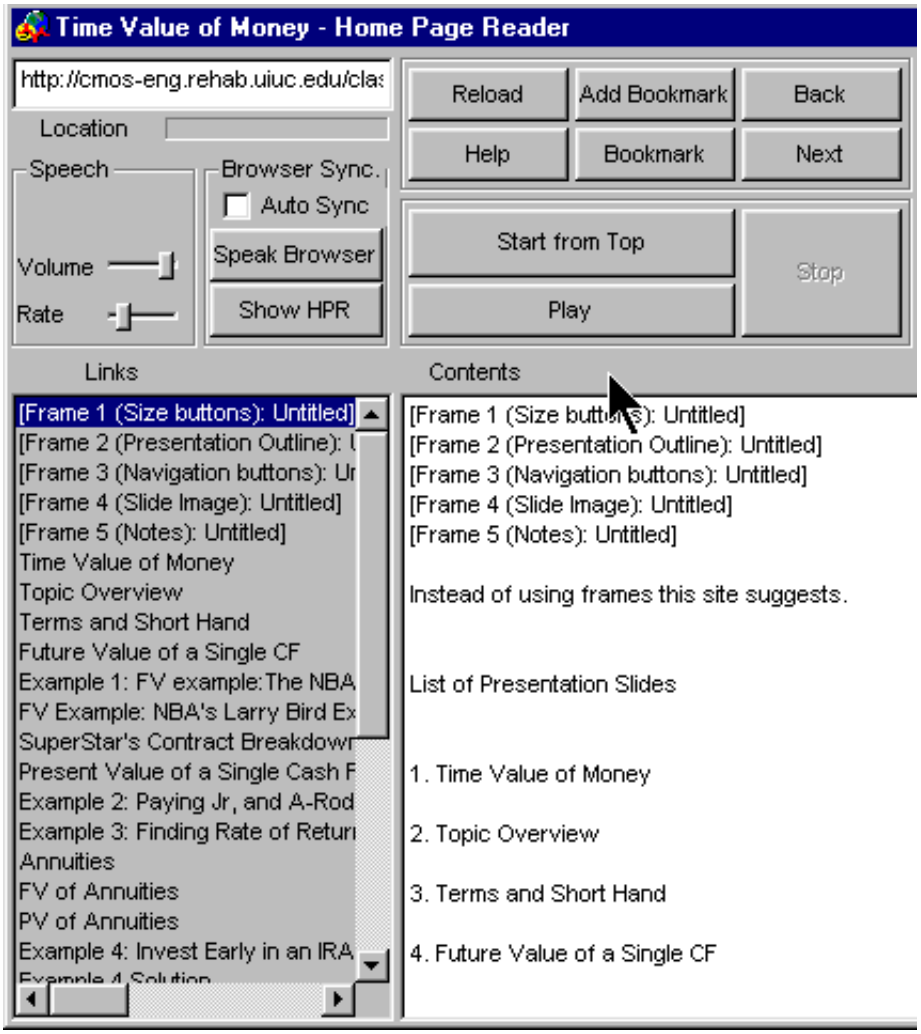
Rendering by Lynx *[LYNX]* :

Example.

Time Value of Money

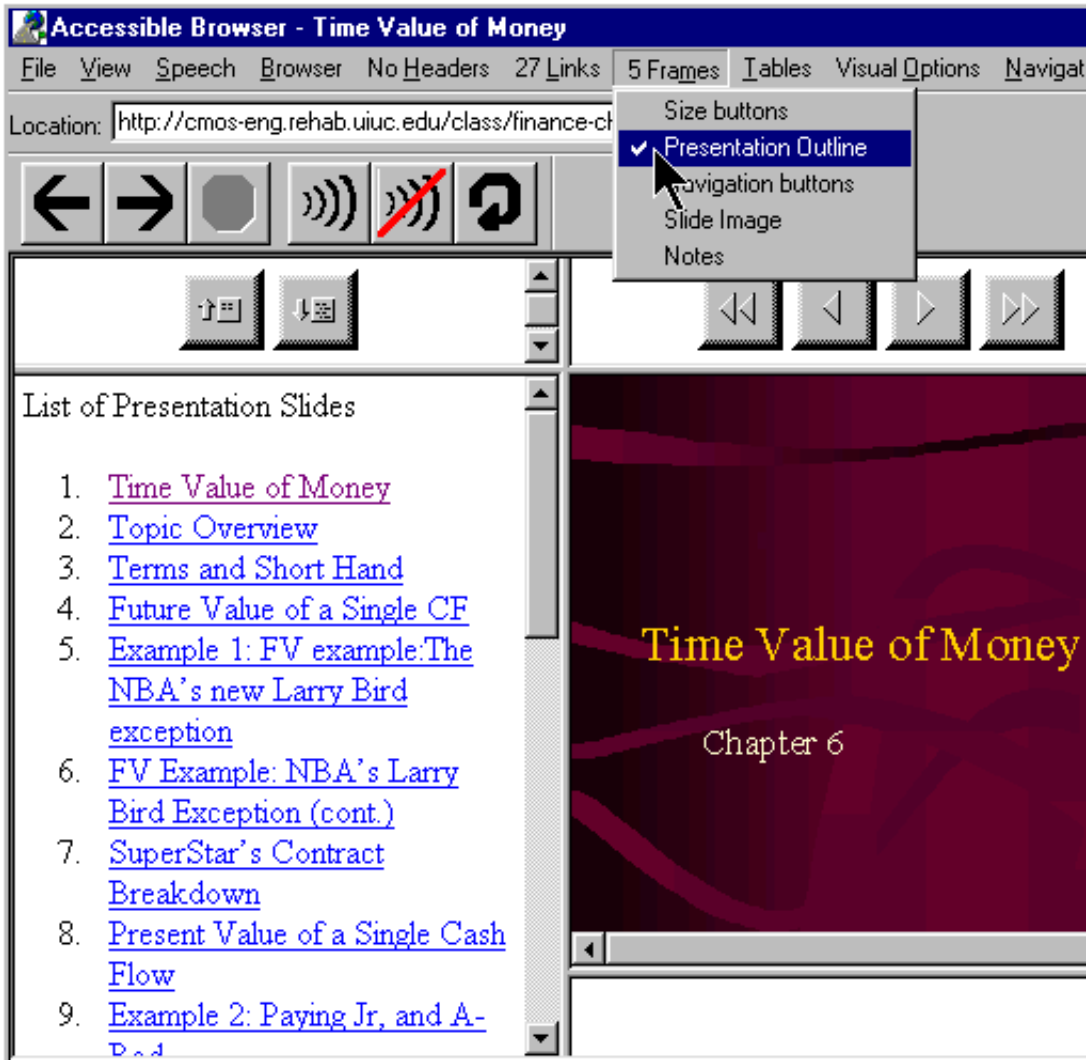
FRAME: Size buttons
FRAME: Presentation Outline
FRAME: Navigation buttons
FRAME: Slide Image
FRAME: Notes

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 has the current focus 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 frame with the current focus.

2.8 Form techniques

To make a form accessible, the user agent needs to ensure that:

- the user can navigate to all of the form elements;
- information about the form and its elements is available on demand;
- the user can interact with all form elements through the keyboard alone (or voice alone or pointing device alone).

2.8.1 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 element (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 elements (e.g., those grouped by `LEGEND` or `OPTGROUP` in HTML) along with their labels.
- Allow the user to navigate away from a menu without selecting any option (e.g., by pressing the **`Escape`** key).

2.8.2 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.

2.8.3 Form element orientation techniques

In conjunction with navigation:

- As the user navigates to a form element, provide information about whether the control has to be activated before form submission.
- For labels associated with form elements in markup (e.g., the "`for`" attribute on `LABEL` in HTML), make available label information when the user navigates among the form elements.
- As the user navigates to a form element, provide information (e.g., through context-sensitive help) about how the user can activate the element. Provide information about what is required for each form element. Lynx *[LYNX]* conveys this information by providing information about the currently selected form element 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`).

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

- Indicate the number of elements in the form.
- Indicate the number of elements that have not yet been completed.
- Provide a list of elements that have to be activated before form submission.
- Provide information about the order of form elements (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 element has focus (e.g., "element 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 elements know whether they have completed the form.
- Allow the user to query a form element 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 specifies the `FIELDSET` element (*[HTML4]* , section 17.10), which allows authors to group thematically related elements 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` element. 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 elements. 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.

2.9 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)

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.

Example.

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 the content is invisible .

The following XSLT style sheet (excerpted from the XSLT Recommendation *[XSLT]* , Section 7.7) shows how 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.

2.10 Content repair techniques

When generating repair content , user agent developers should consider the following issues:

- Not all repair content appears in the document object .
- Users may want to distinguish content (in the document object) provided by the author from content generated by the user agent. For example, the user may trust author-supplied content more than generated content.
- Repair content inserted in the document object should conform to the Web Content Accessibility Guidelines 1.0 *[WCAG10]* . For example, if the user agent inserts a graphical icon in the document object model , that icon should have a text equivalent : since the icon is known to the user agent developer, the developer can provide a sensible text equivalent to accompany it (for the benefit of users of assistive technologies).
- Notification of user agent-initiated changes to the document object model may be made through "DOM events" (refer to the "Document Object Model (DOM) Level 2 Events Specification" *[DOM2EVENTS]*).

See also the section on table cell header repair strategies . Refer also to the W3C document "Techniques for Authoring Tool Accessibility Guidelines 1.0" *[ATAG10-TECHS]* .

2.11 Script and applet techniques

User agents need to 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 do user agents make available equivalents to dynamic content, they have to allow users to turn off scripts, to stop animations , adjust timing parameters, etc. Some user agents also allow users to turn off scripts for security reasons.

2.11.1 Script techniques

Certain elements of a markup language may have associated event handlers that are activated when certain events occur. User agents need to be able to identify those elements with event handlers statically associated (i.e., associated in the content , not in a script). In HTML 4 (*[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 event handlers are activated by (and activate) focus/blur events. Similarly, allow the user to use a key command, such as "**`enter`**" and "**`Shift-enter`**" to fire "`onclick`" and "`ondblclick`" events.
- Implement "Document Object Model (DOM) Level 2 Events Specification" *[DOM2EVENTS]* events with a single activation event and provide a method for firing that event with 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 elements, which can be activated 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)
- For authors: Document the effects of known important scripts to give users an idea in advance of what they do.

2.11.2 Applet techniques

When a user agent loads an applet , it should support the Java system conventions for loading an assistive technology (see the appendix on loading assistive technologies for DOM access). If the user is accessing the applet through an assistive technology, the assistive technology should alert 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 need to make conditional content available to the assistive technology. Applets generally include an application frame that provides title information.

2.12 Input configuration techniques

User agents that allow users to modify default input configurations need to account for configuration information from several sources: user agent defaults, user preferences, author-specified configurations, and operating environment 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 environment 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 12.3 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 viewport with the current focus 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 needs to be accessible: do not rely on color alone to convey information, use conventional 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 taken(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).

2.12.1 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 options for resolving conflicts:

- 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 native configuration may conflict with operating environment conventions. For example, Internet Explorer *[IE-WIN]* in Windows uses the **`Alt`** key as the compose key for author-specified bindings. If the author has specified a

configuration with the characters "**h**" or "**f**", this will interfere with the operating environment 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 compose key (either globally or on a per-document basis when conflicts are detected).

2.13 Synthesized speech techniques

The following techniques apply to user agents that render content as synthesized speech.

- Since user agents that render content as synthesized 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 synthesized speech rate, volume, and pitch.
- User agents that provide accessible solutions for images should, by default, provide *no* information about images where the author has provided empty conditional content associated with the image, 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.
- 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 to be alerted.
- 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."

References:

- For more information about voice browser technology developed at W3C, refer to "Voice Browsers: An introduction and glossary for the requirements drafts" *[VOICEBROWSER]* .
- For information about voice recognition and accessibility, refer to "Speak to Write" *[SPEAK2WRITE]* .

2.14 Techniques for reducing dependency on spatial interactions

Some interactions with content may require spatial information, often provided by users without disabilities through a pointing device such as a mouse.

- User agents should not require users to "move through space" to interact with content (or "time", for that matter; see checkpoint 2.4). Thus, for users without a pointing device, the user agent's first approximation of access, say through the keyboard, would be to simulate the mouse with keystrokes. However, such a technique usually requires a significant amount of visual feedback as well as physical dexterity, both of which may not be possible for users with some disabilities.
- A better alternative is to allow users to enter coordinates where a click should occur. While this is "direct access" to the coordinate, this requires extensive knowledge of the geometry in question.
- A still better alternative is to allow the user to interact with "objects" in content at a more abstract level than geometry. For example, most HTML authors can use "client-side" image maps rather than "server-side" since what is important is not the actual coordinates but rather that the user has selected one region instead

of another. The user agent should convey information about the regions, using descriptions provided by the author. Instead of having users choose a state of the United States by its precise longitude and latitude, it is possible to allow them to choose state by name.

2.15 Accessibility and internationalization techniques

The following techniques may be considered when integrating accessibility features and internationalization.

- Implement content negotiation so that users may specify language preferences. Or allow the user to choose manually from among related resources available in different languages.
- Consider operating environment level natural language preferences as the user's default language preference. However, take caution about sending HTTP Accept-Language request headers (*[RFC2616]* , section 14.4) based on the operating environment 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 environment may define only one language, while the Accept-Language request header may include many languages in different priorities. Setting Accept-Language to be the operating environment 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.
- Render characters with the appropriate directionality. Refer to the "dir" attribute and the BDO element in HTML 4 (*[HTML4]* , sections 8.2 and 8.2.4 respectively). Refer also to the Unicode specification *[UNICODE]* .

2.16 Appendix: Impact matrix

This matrix summarizes which checkpoints are expected to benefit users with certain types of disabilities. For more information about types of disabilities, assistive technologies, access strategies, and more, refer to "How People with Disabilities Use the Web" *[PWD-USE-WEB]* .

seizure disorder
3.3 , 3.4
all
2.1 , 2.3 , 4.17 , 6.1 , 6.2 , 6.3 , 6.4 , 6.5 , 6.6 , 6.7 , 6.8 , 6.9 , 7.1 , 7.2 , 7.3 , 7.4 , 8.1 , 8.2 , 12.1 , 12.2 , 12.3 , 12.4 , 12.5
learning
4.4 , 4.7 , 4.8 , 4.15
cognitive
2.6 , 2.10 , 3.1 , 3.2 , 3.3 , 3.5 , 3.6 , 3.7 , 4.2 , 4.3 , 4.4 , 4.5 , 4.6 , 4.7 , 4.8 , 4.15 , 5.1 , 5.2 , 5.3 , 5.4 , 5.5 , 5.6 , 5.7 , 9.4 , 9.8 , 9.10 , 10.5 , 10.6 , 10.8 , 11.1 , 11.2 , 11.3 , 11.4 , 11.5 , 11.6 , 11.7

deafness
1.3 , 2.2 , 2.5 , 2.6
memory
9.4 , 9.10 , 10.5 , 10.6 , 11.1 , 11.2 , 11.3 , 11.4 , 11.5 , 11.6 , 11.7
low vision
2.2 , 2.5 , 2.7 , 2.8 , 2.9 , 3.2 , 3.3 , 3.5 , 3.6 , 4.1 , 4.2 , 4.4 , 4.6 , 4.7 , 4.8 , 4.9 , 4.10 , 4.11 , 4.12 , 4.13 , 4.14 , 4.15 , 4.16 , 5.1 , 5.4 , 9.1 , 9.2 , 9.3 , 9.4 , 9.7 , 9.8 , 9.10 , 10.4 , 10.5 , 10.6 , 10.7
blindness
1.1 , 1.2 , 1.3 , 2.2 , 2.4 , 2.5 , 2.7 , 2.8 , 2.9 , 2.11 , 3.2 , 3.3 , 3.5 , 3.6 , 4.4 , 4.7 , 4.8 , 4.9 , 4.10 , 4.11 , 4.12 , 4.13 , 4.14 , 4.15 , 4.16 , 5.1 , 5.3 , 5.4 , 5.5 , 9.1 , 9.2 , 9.3 , 9.4 , 9.5 , 9.6 , 9.7 , 9.8 , 9.9 , 9.10 , 10.4 , 10.5 , 10.6 , 10.7 , 10.8 , 11.1 , 11.2 , 11.3 , 11.5 , 11.6 , 11.7
color deficiency
3.1 , 4.3 , 10.2 , 10.4 , 10.7
physical
1.1 , 1.2 , 2.4 , 2.9 , 2.11 , 5.3 , 5.5 , 5.6 , 9.1 , 9.2 , 9.3 , 9.4 , 9.5 , 9.6 , 9.7 , 9.8 , 9.9 , 9.10 , 10.5 , 10.6 , 11.1 , 11.2 , 11.3 , 11.4 , 11.5 , 11.6 , 11.7
hard of hearing
1.3 , 2.2 , 2.5 , 2.6 , 4.9

2.17 Appendix: Accessibility features of some operating systems

Several operating systems 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 viewport that has the current focus 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 alternative 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 conventional input devices/methods.

Microsoft Windows 95, Windows 98, and Windows NT 4.0

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.

References:

- 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]* .

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

2.18 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 (see checkpoint 6.9). 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.

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 alerted 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:

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 alerted that an object has received 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.

Example.

```
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 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-process 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.

Example.

```
/* 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]* .

3 Glossary

This glossary is normative . Some terms (or parts of explanations of terms) may not have an impact on conformance.

Note: In this document, glossary terms generally link to the corresponding entries in this section. These terms are also highlighted through style sheets and identified as glossary terms through markup.

Activate

In this document, the verb "to activate" means (depending on context) either:

- To execute or carry out one or more behaviors associated with an enabled element .
- To execute or carry out one or more behaviors associated with a component of the user agent user interface .

The effect of activation depends on the type of enabled element or user interface control. For instance, when a link is activated, the user agent generally retrieves the linked Web resource . When a form element is activated, it may change state (e.g., check boxes) or may take user input (e.g., a text entry field).

Alert

In this document, "to alert" means to make the user aware of some event, without requiring acknowledgement. For example, the user agent may alert the user that new content is available on the server by displaying a text message in the user agent's status bar. See checkpoint 1.3 for requirements about alerts.

Animation

In this document, an "animation" refers to content that, when rendered, creates a visual movement effect automatically (i.e., without manual user interaction). This definition of animation includes video and animated images. Animation techniques include:

- graphically displaying a sequence of snapshots within the same region (e.g., as is done for video and animated images). The series of snapshots may be provided by a single resource (e.g., an animated GIF image) or from distinct resources (e.g., a series of images downloaded continuously by the user agent).
- scrolling text (e.g., achieved through markup or style sheets).
- displacing graphical objects around the viewport (e.g., a picture of a ball that is moved around the viewport giving the impression that it is bouncing off of the viewport edges). For instance, the SMIL 2.0 *[SMIL20]* animation modules explain how to create such animation effects in a declarative manner (i.e., not by composition of successive snapshots).

Applet

An applet is a program (generally written in the Java programming language) that is part of content , and that the user agent executes.

Application Programming Interface (API), ***conventional input/output/device API***

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

Implementing APIs that are independent of a particular operating environment (as are the W3C DOM Level 2 specifications) may reduce implementation costs for multi-platform user agents and promote the development of multi-platform assistive technologies. Implementing conventional APIs for a particular operating environment may reduce implementation costs for assistive technology developers who wish to interoperate with more than one piece of software running on that operating environment.

A "device API" defines how communication may take place with an input or output device such as a keyboard, mouse, video card, etc.

In this document, an "input/output API" defines how applications or devices communicate with a user agent. As used in this document, input and output APIs include, but are not limited to, device APIs. Input and output APIs also include more abstract communication interfaces than those specified by device APIs. A "conventional input/output API" is one that is expected to be implemented by software running on a particular operating environment. For example, on desktop computers today, the conventional input APIs are for the mouse and keyboard. For touch screen devices or mobile devices, conventional input APIs may include stylus, buttons, voice, etc. The graphical display and sound card are considered conventional ouput devices for a graphical desktop computer environment, and each has an associated API.

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 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 rendered 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.
- voice 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. For example, the following very general definition of "assistive technology device" comes from the (U.S.) Assistive Technology Act of 1998 *[AT1998]* :

> Any item, piece of equipment, or product system, whether acquired commercially, modified, or customized, that is used to increase, maintain, or improve functional capabilities of individuals with disabilities.

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

In this document, the term "audio" refers to content that encodes pre-recorded sound.

Audio-only presentation

An audio-only presentation is content 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 narration.

Audio track

An audio object is content rendered as sound through an audio viewport . An audio track is an audio object that is intended as a whole or partial presentation. An audio track may, but is not required to, correspond to a single audio channel (left or right audio channel).

Auditory description

An auditory description (sometimes, "audio description") is either a prerecorded human voice or a synthesized voice (recorded or generated dynamically) describing the key visual elements of a movie or other animation. The auditory description is synchronized with (and possibly included as part of) 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 content (e.g., style sheets within a document, that are associated with a document, or that are generated by a server).

Captions

Captions (sometimes, "closed captions") are text transcripts that are synchronized with other audio tracks 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.

Character encoding

A "character encoding" is a mapping from a character set definition to the actual code units used to represent the data. Please refer to the Unicode specification *[UNICODE]* for more information about character encodings. Refer to "Character Model for the World Wide Web" *[CHARMOD]* for additional information about characters and character encodings.

Collated text transcript

A collated text transcript is a text equivalent of a movie or other 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). See also the definitions of text transcript and auditory description . Collated text transcripts are essential for individuals who are deaf-blind.

Conditional content

Conditional content is content that, by format specification, should be made available to users through the user interface, generally under certain conditions (e.g., based on user preferences or operating environment limitations). Some examples of conditional content mechanisms include:

- The "`alt`" attribute of the `IMG` element in HTML 4. According to section 13.2 of the HTML 4 specification (*[HTML4]*): "User agents must render alternate text when they cannot support images, they cannot support a certain image type or when they are configured not to display images.
- `OBJECT` elements in HTML 4. Section 13.3.1 of the HTML 4 specification (*[HTML4]*) explains the conditional rendering rules of (nested) `OBJECT` elements. The rules select among ordered alternatives according to user preferences or error conditions.
- The `switch` element and test attributes in SMIL 1.0. Sections 4.3 and 4.4, respectively, of SMIL 1.0 *[SMIL]* explain the conditional rendering rules of these features.
- SVG 1.0 *[SVG]* also includes a `switch` element and several attributes for conditional processing.
- The `NOSCRIPT` and `NOFRAMES` elements in HTML 4 *[HTML4]* allow the author to provide content under conditions when the user agent does not

support scripts or frames, or the user has turned off support for scripts or frames.

Specifications vary in how completely they define how and when to render conditional content. For instance, the HTML 4 specification includes the rendering conditions for the "`alt`" attribute, but not for the "`title`" attribute. The HTML 4 specification does indicate that the "`title`" attribute should be available to users through the user interface ("Values of the title attribute may be rendered by user agents in a variety of ways...").

Note: The Web Content Accessibility Guidelines 1.0 requires that authors provide text equivalents for non-text content. This is generally done by using the conditional content mechanisms of a markup language. Since conditional content may not be rendered by default, the current document requires the user agent to provide access to unrendered conditional content (checkpoint 2.3 and checkpoint 2.9) as it may have been provided to promote accessibility.

Configure, ***control***

In the context of this document, the verbs "to control" and "to 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). In order to be able to configure and control the user agent, the user needs to be able to "read" as well as "write" values for these parameters. 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 limitations of the operating environment or hardware.

Both configuration and control may apply at different "levels": across Web resources (i.e., at the user agent level, or inherited from the operating environment), to the entirety of a Web resource, or to components of a Web resource (e.g., on a per-element basis).

A ***global configuration*** is one that applies across elements of the same Web resource, as well as across Web resources. A global configuration may be implemented by more than one setting (e.g., per component of the user agent). For instance, when a user agent consists of a browser that renders HTML and a plug-in that renders SVG, to satisfy the global configuration requirements of this document, the browser may provide one setting and the plug-in another.

User agents may allow users to choose configurations based on various parameters, such as hardware capabilities, natural language, etc.

Note: In this document, the noun "control" refers to a component of the user agent user interface .

Content

In this specification, the noun "content" is used in three ways:

1. It is used to mean the document object as a whole or in parts.
2. It is used to mean 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.
3. It is used in the context of the phrases non-text content and text content .

Empty content is either a null value or a string consisting of zero characters. For instance, in HTML, "`alt=''`" sets the value of the "`alt`" attribute to the empty string. In some markup languages, an element may have empty content (e.g., the `HR` element in HTML).

Device-independence

Device-independence refers to the ability to make use of software with any appropriate supported input or output device.

Document object, ***Document Object Model*** (DOM)

In general usage, the term "document object" refers to 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.), produced as a result of preferences set within the user agent, added as the result of a repair performed automatically by the user agent, etc. 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 by the user agent without user awareness, such as DTD- or schema-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 kinds of data through the user interface. Most of the requirements of this document apply to the document object after its construction. However, a few checkpoints (e.g., checkpoint 2.7 and checkpoint 2.11) may affect the construction of the document object.

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 may vary in implementation and sometimes in scope. This specification requires that user agents implement the APIs defined in Document Object Model (DOM) Level 2 Specifications (*[DOM2CORE]* and *[DOM2STYLE]*) for access to HTML, XML, and CSS content. These DOM APIs allow authors to access and modify the content via a scripting language (e.g., JavaScript) in a consistent manner across different scripting languages. As a standard interface, the DOM APIs make 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.

Document character set

A document character set (a concept taken from SGML) is a sequence of abstract characters that may appear in Web content represented in a particular format (such as HTML, XML, etc.). A document character set consists of:

- a "repertoire", A set of abstract characters, such as the Latin letter "A", the Cyrillic letter "I", the Chinese character meaning "water", etc.
- Code positions: A set of integer references to characters in the repertoire.

For instance, the character set required by the HTML 4 specification *[HTML4]* is defined in the Unicode specification *[UNICODE]* . Refer to "Character Model for the World Wide Web" *[CHARMOD]* for more information about document character sets.

Document source, ***text source***

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 (e.g., as the result of an HTTP/1.1 *[RFC2616]* "GET", or as the result of viewing a resource on the local file system). The document source generally refers to the "payload" of the user agent's request, and doesn't generally include information exchanged as part of the transfer protocol. The document source is data that is prior to any repair by the user agent (e.g., prior to repairing invalid markup). "Text source" refers to document source that is composed of text .

Documentation

Documentation refers to information that supports the use of a user agent. This information may be found in manuals, installation instructions, the help system, tutorials, etc. Documentation may be distributed (e.g., some parts may be delivered on CD-ROM, others on the Web). Refer to guideline 12 for information about documentation requirements.

Element, ***element type***,

This document uses the terms "element" and "element type" in the sense employed by the XML 1.0 specification (*[XML]* , section 3): an element type is a syntactic construct of a Document Type Definition (DTD) for its application. This sense is also relevant to structures defined by XML schemas. The document also uses the term "element" more generally to mean a type of content (such as video or sound) or a logical construct (such as a header or list).

Enabled element, ***disabled element***

An enabled element is a piece of content with associated behaviors that may be activated through the user interface or through an API . The set of elements that a user agent enables is generally derived from, but is not limited to, the set of interactive elements defined by implemented markup languages.

Some elements may only be enabled elements for part of a user session. For instance, an element may be disabled by a script as the result of user interaction. Or, an element may only be enabled during a given time period (e.g., during part of a SMIL 1.0 *[SMIL]* presentation). Or, the user may be viewing content in "read-only" mode, which may disable some elements.

A disabled element is a piece of content that is potentially an enabled element, but is not in the current session. Generally, disabled elements will be interactive elements that are not enabled in the current session. This document distinguishes disabled elements (not currently enabled) from non-interactive elements (never enabled).

For the requirements of this document, user selection does not constitute user interaction with enabled elements. See the definition of content focus .

Note: Enabled and disabled elements come from content; they are not part of the user agent user interface .

Note: The term "active element" is not used in this document since it may suggest several different concepts, including: interactive element, enabled element, an element "in the process of being activated" (which is the meaning of ':active' in CSS2 *[CSS2]* , for example).

Equivalent (for content)

The term "equivalent" is used in this document as it is used in the Web Content Accessibility Guidelines 1.0 *[WCAG10]* :

> Content is "equivalent" to other content when both fulfill essentially the same function or purpose upon presentation to the user. 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.

Equivalents include text equivalents (e.g., text equivalents for images; text transcripts for audio tracks; collated text transcripts for multimedia presentations and animations) and non-text equivalents (e.g., a prerecorded auditory description of a visual track of a movie, or a sign language video rendition of a written text, etc.).

Each markup language defines its own mechanisms for specifying conditional content , and these mechanisms may be used by authors to provide text equivalents. For instance, in HTML 4 *[HTML4]* or SMIL 1.0 *[SMIL]* , authors may use the "`alt`" attribute to specify a text equivalent for some elements. In HTML 4, authors may provide equivalents (or portions of equivalents) 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 equivalents, and `NOSCRIPT` for script equivalents), and in prose. Please consult the Web Content Accessibility Guidelines 1.0 *[WCAG10]* and its associated Techniques document *[WCAG10-TECHS]* for more information about equivalents.

Events and scripting, event handler

User agents often perform a task when an event occurs that is due to user interaction (e.g., document loading, mouse motion or a key press, a request from the operating environment , etc.). Some markup languages allow authors

to specify that a script, called an ***event handler***, be executed when the event occurs. An event handler is "***explicitly associated with an element***" when the event handler is associated with that element through markup or the DOM. The term "***event bubbling***" describes a programming style where a single event handler dispatches events to more than one element. In this case, the event handlers are not explicitly associated with the elements receiving the events (except for the single element that dispatches the 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.

Explicit user request

In this document, the term "explicit user request" refers to any user interaction with a control provided by the user agent user interface (**not** those in content), the focus , or selection . Control behavior should be documented .

Some examples of explicit user requests include when the user selects "New viewport", responds "Yes" to a prompt in the user agent's user interface, configures the user agent to behave in a certain way, or changes the selection or focus with the keyboard or pointing device.

Note: Users make mistakes. For example, a user may inadvertently respond "yes" to a prompt when they meant "no." In this document, this type of mistake is still considered an explicit user request.

Fee link

For the purpose of this document, the term "fee link" refers to a link that when activated, debits the user's electronic "wallet" (generally, a "micropayment"). The link's role as a fee link is identified through markup (in a manner that the user agent can recognize). This definition of fee link excludes payment mechanisms (e.g., some form-based credit card transactions) that cannot be recognized by the user agent as causing payments. For more information about fee links, refer to "Common Markup for micropayment per-fee-links" *[MICROPAYMENT]* .

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

In this document, the term "content focus" refers to a user agent mechanism that satisfies all of the following properties:

1. It designates zero or one element in content that is either enabled or disabled . (In general, the focus should only designate enabled elements, but it may also designate disabled elements.)
2. The user may "set" content focus (programmatically or through the user interface) on an enabled element without triggering the associated behaviors.
3. It has state. The user may prefer to always move the content focus manually from one element to another.
4. It may be used (programmatically or through the user interface) to trigger the behaviors associated with an enabled element. This is generally

implemented by making the focus respond to input device events (often just keyboard events).

User interface mechanisms may resemble content focus, but do not satisfy all of the properties. For example, text editors often implement a "caret" that indicates the current location of text input or editing. The caret may have state and may respond to input device events, but it does not enable users to activate the behaviors associated with enabled elements.

The user interface focus shares the properties of the content focus except the first: the user interface focus designates zero or one control of the user agent user interface that has associated behaviors (e.g., radio button, text box, menu, etc.).

On the screen, the content focus may be highlighted using colors, fonts, graphics, magnification, etc. The content focus may also be highlighted when rendered as synthesized speech, for example through changes in speech prosody. The dimensions of the rendered content focus may exceed those of the viewport.

In this document, each viewport is expected to have at most one content focus and at most one user interface focus. This document includes requirements for content focus only, for user interface focus only, and for both. When a requirement refers to both, the term "focus" is used.

When several viewports coexist, at most one viewport's content focus **or** user interface focus responds to input events; this is called the current focus.

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

Highlight
In this document, "to highlight" means to emphasize through the user interface. For example, user agents highlight which content is selected or focused. Graphical highlight mechanisms include dotted boxes, underlining, and reverse video. Synthesized speech highlight mechanisms include alterations of voice pitch and volume ("speech prosody").

Image
In this document, an "image" refers to content that encodes static (i.e., unmoving) visual information. See also the definition of animation .

Input configuration
An input configuration is the mapping of user agent functionalities to some user interface input 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 documented (per checkpoint 12.3]). Input configurations may be affected by author-specified bindings (e.g., through the "accesskey" attribute of HTML 4 *[HTML4]*).

Interactive element, ***non-interactive element***,
An interactive element is piece of content that, by specification, may have associated behaviors to be executed or carried out as a result of user or

programmatic interaction. For instance, the interactive elements of HTML 4 *[HTML4]* include: links, image maps, form elements, elements with a value for the "longdesc" attribute, and elements with event handlers explicitly associated with them (e.g., through the various "on" attributes). The role of an element as an interactive element is subject to applicability . A non-interactive element is an element that, by format specification, does not have associated behaviors. The expectation of this document is that interactive elements become enabled elements in some sessions, and non-interactive elements never become enabled elements.

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 (*[HTML4]* section 8.1), the "xml:lang" attribute in XML 1.0 (*[XML]* , section 2.12), the HTML 4 "hreflang" attribute for links in HTML 4 (*[HTML4]* , section 12.1.5), the HTTP Content-Language header (*[RFC2616]* , section 14.12) and the Accept-Language request header (*[RFC2616]* , section 14.4). See also the definition of script .

Normative, ***informative***

As used in this document, the term "normative" refers to "that on which the requirements of this document depend for their most precise statement." What is normative is required for conformance (though the conformance scheme of this document allows claimants to exempt certain normative provisions as long as the claim discloses the exemption). What is identified as "informative" (sometimes, "non-normative") is never required for conformance.

Operating environment

The term "operating environment" refers to the environment that governs the user agent's operation, whether it is an operating system or a programming language environment such as Java.

Override

In this document, the term "override" means that one configuration or behavior preference prevails over another. Generally, the requirements of this document involve user preferences prevailing over author preferences and user agent default settings and behaviors. Preferences may be multi-valued in general (e.g., the user prefers blue over red or yellow), and include the special case of two values (e.g., turn on or off blinking text content).

Placeholder

A placeholder is content generated by the user agent to replace author-supplied content. A placeholder may be generated as the result of a user preference (e.g., to not render images) or as repair content (e.g., when an image cannot be found). Placeholders can be any type of content, including text, images, and audio cues.

This document includes requirements that the user be able to view the original author-supplied content associated with a placeholder. To satisfy these requirements, the user agent might render the content in place of the placeholder or in a separate viewport (leaving the placeholder as is). A request

to view the original content associated with a placeholder is considered an explicit user request to render that content.

This document does not require user agents to include placeholders in the document object . A placeholder that is inserted in the document object should conform to the Web Content Accessibility Guidelines 1.0 *[WCAG10]* . If a placeholder is not part of the document object, it is part of the user interface only (and subject, for example, to checkpoint 1.3).

Plug-in

A plug-in is a program that runs as part of the user agent and that is *not* part of content . Users generally choose to include or exclude plug-ins from their user agent.

Point of regard

The point of regard is a position in rendered content that the user is presumed to be viewing. The dimensions of the point of regard may vary. For example, it may be a point (e.g., a moment in an audio rendering or a cursor in a graphical rendering), or a range of text (e.g., focused text), or a two-dimensional area (e.g., content rendered through a two-dimensional graphical viewport). The point of regard is almost always within the viewport, but it may exceed the spatial or temporal dimensions of the viewport (see the definition of rendered content for more information about viewport dimensions). 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 determine the point of regard in a number of ways, including based on viewport position in content, content focus , selection , etc. A user agent should not change the point of regard unexpectedly as this may disorient the user.

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, natural language preferences, etc. In operating environments with distinct user accounts, profiles enable users to reconfigure software quickly when they log on, and profiles may be shared by several users.
Platform-independent profiles are useful for those who use the same user agent on different platforms.

Prompt

In this document, "to prompt" means to require input from the user. The user agent should allow users to configure how they wish to be prompted. For instance, for a user agent functionality X, configurations might include: always prompt me before doing X, always do X without prompting me, never do X but tell me when you could have, never do X and never tell me that you could have, etc.

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 loudspeakers, on a braille display, on a mobile device, etc. Style information (e.g., fonts, colors, synthesized speech prosody, 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

Authors encode information in markup languages, style sheet languages, scripting languages, protocols, etc. When the information is encoded in a manner that allows the user agent to process it with certainty, the user agent can "recognize" the information. For instance, HTML allows authors to specify a heading with the H1 element, so a user agent that implements HTML can recognize that content as a heading. If the author creates headings using a visual effect alone (e.g., by increasing the font size), then the author has encoded the heading in a manner that does not allow the user agent to recognize it as a heading.

Some requirements of this document depend on content roles, content relationships, timing relationships, and other information supplied by the author. These requirements only apply when the author has encoded that information in a manner that the user agent can recognize. See the section on conformance in User Agent Accessibility Guidelines 1.0 *[UAAG10]* for more information about applicability.

In practice, user agents will rely heavily on information that the author has encoded in a markup language or style sheet language. On the other hand, behaviors, style, meaning encoded in a script , and markup in an unfamiliar XML namespace may not be recognized by the user agent as easily or at all.

Rendered content, ***rendered text***

Rendered content is the part of content that the user agent makes available to the user's senses of sight and hearing (and only those senses for the purposes of this document). Any content that causes an effect that may be perceived through these senses constitutes rendered content. This includes text characters, images, style sheets, scripts, and anything else in content that, once processed, may be perceived through sight and hearing.

The term "rendered text" refers to text content that is rendered in a way that communicates information about the characters themselves, whether visually or as synthesized speech.

In the context of this document, "***invisible content***" is content that influences graphical rendering of other content but is not rendered itself. Similarly, "***silent content***" is content that influences audio rendering of other content but is not rendered itself. Neither invisible nor silent content is considered rendered content.

Repair content*, *repair text

In this document, the term "repair content" refers to content generated by the user agent in order to correct an error condition. "Repair text" means repair content consisting only of text . Some error conditions that may lead to the generation of repair content include:

- Erroneous or incomplete content (e.g., ill-formed markup, invalid markup, missing conditional content that is required by format specification, etc.);
- Missing resources for handling or rendering content (e.g., the user agent lacks a font family to display some characters, the user agent doesn't implement a particular scripting language, etc.);

This document does not require user agents to include repair content in the document object . Repair content inserted in the document object should conform to the Web Content Accessibility Guidelines 1.0 *[WCAG10]* . For more information about repair techniques for Web content and software, refer to "Techniques for Authoring Tool Accessibility Guidelines 1.0" *[ATAG10-TECHS]* .

Script

In this document, the term "script" almost always refers to a scripting (programming) language used to create dynamic Web content. However, in checkpoints referring to the written (natural) language of content, the term "script" is used as in Unicode *[UNICODE]* to mean "A collection of symbols used to represent textual information in one or more writing systems."

Information encoded in scripts may be difficult for a user agent to recognize For instance, a user agent is not expected to recognize that, when executed, a script will calculate a factorial. The user agent will be able to recognize some information in a script by virtue of implementing the scripting language or a known program library (e.g., the user agent is expected to recognize when a script will open a viewport or retrieve a resource from the Web).

Selection*, *current selection

In this document, the term "selection" refers to a user agent mechanism for identifying a range of content (e.g., text, images, etc.). Generally, user agents limit selection to text content (e.g., one or more fragments of text). The selection may be structured (based on the document tree) or unstructured (e.g., text-based). The range may be empty.

On the screen, the selection may be highlighted using colors, fonts, graphics, magnification, etc. The selection may also be highlighted when rendered as synthesized speech, for example through changes in speech prosody. The dimensions of the rendered selection may exceed those of the viewport.

The selection may be used for a variety of purposes: for cut and paste operations, to designate a specific element in a document for the purposes of a query, as an indication of point of regard , etc.

The selection has state. It may be set programmatically or through the user interface.

In this document, each viewport is expected to have at most one selection. When several viewports coexist, at most one viewport's selection responds to input events; this is called the current selection.

See the section on the selection label for information about implementing a selection and conformance.

Note: Some user agents may also implement a selection for designating a range of information in controls of the user agent user interface . The current document only includes requirements for a content selection mechanism.

Support, ***implement***, ***conform***

In this document, the terms "support", "implement", and "conform" all refer to what a developer has designed a user agent to do, but they represent different degrees of specificity. A user agent "supports" general classes of objects, such as "images" or "Japanese". A user agent "implements" a specification (e.g., the PNG and SVG image format specifications, a particular scripting language, etc.) or an API (e.g., the DOM API) when it has been programmed to follow all or part of a specification. A user agent "conforms to" a specification when it implements the specification *and* satisfies its conformance criteria. This document includes some conformance requirements to other specifications (e.g., to a particular level of the "Web Content Accessibility Guidelines 1.0" *[WCAG10]*).

Synchronize

In this document, "to synchronize" refers to 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 developers 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.

Text

In this document, the term "text" used by itself refers to a sequence of characters from a markup language's document character set . Refer to the "Character Model for the World Wide Web " *[CHARMOD]* for more information about text and characters. **Note:** This document makes use of other terms that include the word "text" that have highly specialized meanings: collated text transcript , non-text content , text content , non-text element , text element , text equivalent , and text transcript .

Text content, ***non-text content***, ***text element***, ***non-text element***, ***text equivalent*** ***non-text equivalent***

As used in this document a "text element" adds text characters to either content or the user interface . Both in the Web Content Accessibility Guidelines 1.0

[WCAG10] and in this document, text elements are presumed to produce text that can be understood when rendered visually, as synthesized speech, or as Braille. Such text elements benefit at least these three groups of users:

1. visually-displayed text benefits users who are deaf and adept in reading visually-displayed text;
2. synthesized speech benefits users who are blind and adept in use of synthesized speech;
3. braille benefits users who are blind, and possibly deaf-blind, and adept at reading braille.

A text element may consist of both text and non-text data. For instance, a text element may contain markup for style (e.g., font size or color), structure (e.g., heading levels), and other semantics. The essential function of the text element should be retained even if style information happens to be lost in rendering.

A user agent may have to process a text element in order to have access to the text characters. For instance, a text element may consist of markup, it may be encrypted or compressed, or it may include embedded text in a binary format (e.g., JPEG).

"Text content" is content that is composed of one or more text elements. A "text equivalent" (whether in content or the user interface) is an equivalent composed of one or more text elements. Authors generally provide text equivalents for content by using the conditional content mechanisms of a specification.

A "non-text element" is an element (in content or the user interface) that does not have the qualities of a text element. "Non-text content" is composed of one or more non-text elements. A "non-text equivalent" (whether in content or the user interface) is an equivalent composed of one or more non-text elements.

Note that the terms "text element" and "non-text element" are defined by the characteristics of their output (e.g., rendering) rather than those of their input (e.g., information sources) or their internals (e.g., format). Both text elements and non-text elements should be understood as "pre-rendering" content in contrast to the "post-rendering" content that they produce.

Text decoration

In this document, a "text decoration" is any stylistic effect that the user agent may apply to visually rendered text that does not affect the layout of the document (i.e., does not require reformatting when applied or removed). Text decoration mechanisms include underline, overline, and strike-through.

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 other 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 voice-to-text converters). See also the definitions of captions and collated text transcripts .

User agent

In this document, the term "user agent" is used in two ways:

1. Any software that retrieves and renders Web content for users. This may include Web browsers, media players, plug-ins , and other programs – including assistive technologies -- that help in retrieving and rendering Web content.
2. The subject of a conformance claim to User Agent Accessibility Guidelines 1.0 *[UAAG10]* . This is the most common use of the term in this document and is the usage in the checkpoints.

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 (e.g., menus, buttons, prompts, etc.) and mechanisms (e.g., selection and focus) provided by the user agent ("out of the box") that are not created by content .
2. the "content user interface", i.e., the enabled elements that are part of content, such as form elements, links, applets , etc.

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.

Visual-only presentation

A visual-only presentation is content consisting exclusively of one or more visual tracks presented concurrently or in series. A silent movie is an example of a visual-only presentation.

Visual track

A visual object is content rendered through a graphical viewport . Visual objects include graphics, text, and visual portions of movies and other animations. 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 object or software object. A visual track may be text-based or graphic. A visual track may be static or involve animation .

Views, ***viewports***

The user agent renders content through one or more viewports. Viewports include windows, frames, pieces of paper, loudspeakers, virtual magnifying glasses, etc. A viewport may contain another viewport (e.g., nested frames). User interface controls such as prompts, menus, alerts, etc. are not viewports.

When the dimensions (spatial or temporal) of rendered content exceed the dimensions of the viewport (e.g., when the user can only view a portion of a large document through a small graphical viewport, when audio content has already been played, etc.), the user agent provides mechanisms such as scroll bars and advance and rewind controls so that the user can access the rendered content "outside" the viewport.

When several viewports coexist, only one has the current focus at a given moment. This viewport is highlighted to make it stand out.

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.

Voice browser

From "Introduction and Overview of W3C Speech Interface Framework" *[VOICEBROWSER]* : "A voice browser is a device (hardware and software) that interprets voice markup languages to generate voice output, interpret voice input, and possibly accept and produce other modalities of input and output."

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 can be identified by a Uniform Resource Identifier (URI); refer to RFC 2396 *[RFC2396]* .

4 References

For the **latest version** of any W3C specification please consult the list of W3C Technical Reports at http://www.w3.org/TR/. Some documents listed below may have been superseded since the publication of this document.

Note: In this document, bracketed labels such as "[HTML4]" link to the corresponding entries in this section. These labels are also identified as references through markup.

4.1 How to refer to this document

There are two recommended ways to refer to the "Techniques for User Agent Accessibility Guidelines 1.0" (and to W3C documents in general):

1. References to a specific version of "Techniques for User Agent Accessibility Guidelines 1.0". For example, use the "this version" URI to refer to the current document: http://www.w3.org/TR/2001/WD-UAAG10-TECHS-20010912/.
2. References to the latest version of "Techniques for User Agent Accessibility Guidelines 1.0". Use the "latest version" URI to refer to the most recently published document in the series: http://www.w3.org/TR/UAAG10-TECHS/.

In almost all cases, references (either by name or by link) should be to a specific version of the document. W3C will make every effort to make this document indefinitely available at its original address in its original form. The top of this document includes the relevant catalog metadata for specific references (including title, publication date, "this version" URI, editors' names, and copyright information).

An XHTML 1.0 *[XHTML10]* paragraph including a reference to this specific document might be written:

```
<p>
<cite><a href="http://www.w3.org/TR/2001/WD-UAAG10-TECHS-20010912/">
"Techniques for User Agent Accessibility Guidelines 1.0"</a></cite>,
I. Jacobs, J. Gunderson, E. Hansen, eds.,
W3C Working Draft, 12 September 2001.
The <a href="http://www.w3.org/TR/UAAG10-TECHS/">latest
version</a> of this document is available at
http://www.w3.org/TR/UAAG10-TECHS/.</p>
```

For very general references to this document (where stability of content, anchors, etc., is not required), it may be appropriate to refer to the latest version of this document. In this case, please use the "latest version" URI at the top of this document.

4.2 Normative references

[CSS1]
"CSS, level 1 Recommendation", B. Bos, H. Wium Lie, eds., 17 December 1996, revised 11 January 1999. This W3C 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 W3C Recommendation is http://www.w3.org/TR/1998/REC-CSS2-19980512/.

[DOM2CORE]
"Document Object Model (DOM) Level 2 Core Specification", A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, eds., 13 November 2000. This W3C Recommendation is http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.

[DOM2STYLE]
"Document Object Model (DOM) Level 2 Style Specification", V. Apparao, P. Le Hégaret, C. Wilson, eds., 13 November 2000. This W3C Recommendation is http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113/.

[RFC2046]
"Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", N. Freed, N. Borenstein, November 1996.

[UAAG10]
"User Agent Accessibility Guidelines 1.0", I. Jacobs, J. Gunderson, E. Hansen, eds. The latest draft of the guidelines is available at http://www.w3.org/TR/UAAG10/.

[WCAG10]
"Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, and I. Jacobs, eds., 5 May 1999. This W3C Recommendation is http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/.

4.3 Informative references

Some of the references in this section become normative if they are used to satisfy the requirements of guideline 6 and guideline 8.

[AT1998]
The *Assistive Technology Act of 1998, 13 November 1998, United States P.L. 105-394*.

[ATAG10]
"Authoring Tool Accessibility Guidelines 1.0", J. Treviranus, C. McCathieNevile, I. Jacobs, and J. Richards, eds., 3 February 2000. This W3C 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., 4 May 2000. This W3C Note is http://www.w3.org/TR/2000/NOTE-ATAG10-TECHS-20000504/.

[CHARMOD]
"Character Model for the World Wide Web", M. Dürst and F. Yergeau, eds., 29 November 1999. This W3C Working Draft is http://www.w3.org/TR/1999/WD-charmod-19991129/

[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.

[DOM2EVENTS]
Document Object Model (DOM) Level 2 Events Specification, V. Pixley, ed., 13 November 2000. This W3C Recommendation is http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/.

[DOM2RANGE]
Document Object Model (DOM) Level 2 Traversal and Range Specification, J. Kesselman, J. Robie, M. Champion, P. Sharpe, V. Apparao, and L. Wood, eds., 13 November 2000. This W3C Recommendation is http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113/.

[HTML4]
"HTML 4.01 Recommendation", D. Raggett, A. Le Hors, and I. Jacobs, eds., 24 December 1999. This W3C Recommendation is http://www.w3.org/TR/1999/REC-html401-19991224/.

[MATHML20]
"Mathematical Markup Language (MathML) Version 2.0", D. Carlisle, P. Ion, R. Miner, N. Poppelier, et al., 21 February 2001. This W3C Recommendation is http://www.w3.org/TR/2001/REC-MathML2-20010221/.

[MICROPAYMENT]
"Common Markup for micropayment per-fee-links", T. Michel, ed., 25 August 1999. This W3C Working Draft is http://www.w3.org/TR/1999/WD-Micropayment-Markup-19990825/.

[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.

[PWD-USE-WEB]
The *How People with Disabilities Use the Web*, J. Brewer. This document provides an introduction to use of the Web by people with disabilities. It is not yet a formal W3C Working Draft.

[RFC2396]
"Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998.

[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 W3C 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/.

[SMIL20]
Synchronized Multimedia Integration Language (SMIL 2.0) Specification, J. Ayars, et al., eds., 7 August 2001. This W3C Recommendation is http://www.w3.org/TR/2001/REC-smil20-20010807/.

[SVG]
"Scalable Vector Graphics (SVG) 1.0 Specification", J. Ferraiolo, ed., 2 August 2000. This W3C Candidate Recommendation is http://www.w3.org/TR/2000/CR-SVG-20000802/.

[SVG-ACCESS]
"Accessibility Features of SVG", C. McCathieNevile and M.-R. Koivunen, 7 August 2000. This W3C Note is http://www.w3.org/TR/2000/NOTE-SVG-access-20000807/.

[UNICODE]
"The Unicode Standard, Version 3.1". This technical report of the Unicode Consortium is available at http://www.unicode.org/unicode/reports/tr27/. This is a revision of "The Unicode Standard, Version 3.0", The Unicode Consortium, Addison-Wesley Developers Press, 2000. ISBN 0-201-61633-5. Refer also to http://www.unicode.org/unicode/standard/versions/. For information about character encodings , refer to Unicode Technical Report #17 "Character Encoding Model".

[VOICEBROWSER]
"Voice Browsers: An introduction and glossary for the requirements drafts", M. Robin, J. Larson, 23 December 1999. This document is http://www.w3.org/TR/1999/WD-voice-intro-19991223/. This document includes references to additional W3C specifications about voice browser technology.

[WCAG10-TECHS]
"Techniques for Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, and I. Jacobs, eds. This W3C Note is http://www.w3.org/TR/1999/WAI-WEBCONTENT-TECHS-19990505/.

[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. This W3C Working Draft is http://www.w3.org/1999/05/WCA-terms/01.

[XHTML10]
"XHTML[tm] 1.0: The Extensible HyperText Markup Language", S. Pemberton, et al., 26 January 2000. This W3C Recommendation is http://www.w3.org/TR/2000/REC-xhtml1-20000126/.

[XLINK]
"XML Linking Language (XLink) Version 1.0", S. DeRose, E. Maler, D. Orchard, B. Trafford, eds., 3 July 2000. This XML 1.0 Candidate Recommendation is http://www.w3.org/TR/2000/CR-xlink-20000703/.

[XML]
"Extensible Markup Language (XML) 1.0", T. Bray, J. Paoli, C.M. Sperberg-McQueen, eds., 10 February 1998. This W3C Recommendation is http://www.w3.org/TR/1998/REC-xml-19980210.

[XMLSTYLE]
"Associating Style Sheets with XML documents Version 1.0", J. Clark, ed., 29 June 1999. This W3C Recommendation is http://www.w3.org/1999/06/REC-xml-stylesheet-19990629/

[XSLT]
"XSL Transformations (XSLT) Version 1.0", J. Clark, 16 November 1999. This W3C Recommendation is http://www.w3.org/TR/1999/REC-xslt-19991116.

5 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 by W3C.*

5.1 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 Technology Highlighting 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.
- Carbon Event Manager Preliminary API Reference. This reference defines the standard event queue API on the MAC OS X.

[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.

[GNOME-ACCESS]
The *GNOME Accessibility Project*.

[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.

[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. Refer also to information on the Java Foundation Classes.

[JAVA13]
Refer to information about character encodings required by Java version 1.3.

[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.

[NOTES-ACCESS]
"Lotus Notes Accessibility" 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]
Articles, Talks, and Papers from Sun Microsystems about accessibility.

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

[TALKINGBOOKS]
National Information Standards Organization. One activity pursued by this organization concerns Digital Talking Books. Refer to the *"Digital Talking Book Features List"* and *"Digital Talking Book Standards Committee Document Navigation Features List"* drafts for more information.

[TRACE-EZ]
"EZ ACCESS(tm) for electronic devices V 2.0 implementation guide", C. M. Law, G. C. Vanderheiden, 23 February 2000. This guide, developed by the Trace Research and Development Center, describes a simple set of interface enhancements that can be applied to electronic devices so that they can be used by people with disabilities, or anyone who experiences difficulty using a device in the standard method of operation.

[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, available at the IBM Accessibility Center, 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.

5.2 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.

[ADOBE]
access.adobe.com. Tools and information about Adobe PDF and accessibility.
[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.
[DIRECTDOM]
DirectDom technology, available from alphaWorks, allows a Java developer to manipulate the live Document Object Model of a browser or Scalable Vector Graphics plug-in to build rich graphical user interfaces.
[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.
[JFW]
JAWS for Windows.
[LYNX]
The Lynx Browser.
[MOZILLA]
The Mozilla browser.
[NAVIGATOR]
Netscape Navigator.
[ODP-DOM]
Open Directory Project information on the W3C DOM.
[OPERA]
The Opera Browser.
[QUICKTIME]
The QuickTime player.
[TABLENAV]
A table navigation script from the Trace Research Center.

[VALIDATOR]
W3C's HTML/XML Validator service.
[VIAVOICE]
ViaVoice voice recognition software.
[WINDOWEYES]
Window-Eyes.
[WINVISION]
Winvision.

5.3 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 voice recognition to promote accessibility.

5.4 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.

6 Acknowledgments

The active participants of the User Agent Accessibility Guidelines Working Group who authored this document were: James Allan, Denis Anson (College Misericordia), Harvey Bingham, Al Gilman, Jon Gunderson (Chair of the Working Group, University of Illinois, Urbana-Champaign), Eric Hansen (Educational Testing Service), Ian Jacobs (Team Contact, W3C), Tim Lacy (Microsoft), Charles McCathieNevile (W3C), David Poehlman, Mickey Quenzer, Gregory Rosmaita (Visually Impaired Computer Users Group of New York City), and Rich Schwerdtfeger (IBM).

Many thanks to the following people who have contributed through review and past participation in the Working Group: Paul Adelson, Jonny Axelsson, Kitch Barnicle, Olivier Borius, Judy Brewer, Dick Brown, Bryan Campbell, Kevin Carey, Tantek Çelik, 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, Marja-Riitta Koivunen, Peter Korn, Josh Krieger, Catherine Laws, Aaron Leventhal, Greg Lowney, Susan Lesch, Scott Luebking, William Loughborough, Napoleon Maou, Peter Meijer, Karen Moses, Masafumi Nakane, Mark Novak, Charles Oppermann, Mike Paciello, David Pawson, Michael Pederson, Helen Petrie, Michael Pieper, Richard Premack, Jan Richards, Hans Riesebos, Joe Roeder, Lakespur L. Roca, Madeleine Rothberg, 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.