Techniques for Web Content Accessibility Guidelines 1.0

W3C NOTE 5-May-1999

This version:
http://www.w3.org/TR/1999/WAI-WEBCONTENT-TECHS-19990505
(plain text, postscript, pdf, gzip tar file of HTML, zip archive of HTML)
Latest version:
http://www.w3.org/TR/WAI-WEBCONTENT-TECHS
Previous version:
http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990324/wai-pageauth-tech
Latest version of "Web Content Accessibility Guidelines 1.0"
http://www.w3.org/TR/WAI-WEBCONTENT
Editors:
Wendy Chisholm, Trace R & D Center, University of Wisconsin -- Madison
Gregg Vanderheiden, Trace R & D Center, University of Wisconsin -- Madison
Ian Jacobs, W3C

Abstract

This document is a list of techniques that implement the checkpoints defined in "Web Content Accessibility Guidelines 1.0".

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

Status of this document

This document is a NOTE made available by the W3C for discussion only. Publication of this Note by W3C indicates no endorsement by W3C or the W3C Team, or any W3C Members.

While Web Content 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 pages.

This document has been produced as part of the W3C Web Accessibility Initiative. The goal of the Web Content Guidelines Working Group is discussed in the Working Group charter.

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

Please send detailed comments on this document to wai-wcag-editor@w3.org.

Table of Contents

---

1 Priorities

Each checkpoint has a priority level assigned by the Working Group based on the checkpoint's impact on accessibility.

[Priority 1]
: A Web content developer **must** satisfy this checkpoint. Otherwise, one or more groups will find it impossible to access information in the document. Satisfying this checkpoint is a basic requirement for some groups to be able to use Web documents.

[Priority 2]
: A Web content developer **should** satisfy this checkpoint. Otherwise, one or more groups will find it difficult to access information in the document. Satisfying this checkpoint will remove significant barriers to accessing Web documents.

[Priority 3]
: A Web content developer **may** address this checkpoint. Otherwise, one or more groups will find it somewhat difficult to access information in the document. Satisfying this checkpoint will improve access to Web documents.

Some checkpoints specify a priority level that may change under certain (indicated) conditions.

The checkpoints in this document are numbered to match their numbering in Web Content Accessibility Guidelines 1.0.

---

2 How the Techniques are Organized

This document is a list of techniques that implement the checkpoints defined in "Web Content Accessibility Guidelines 1.0". This document is organized as follows:

Accessibility Themes
: This section introduces some general techniques to promote accessibility that are independent of any specific markup language.

HTML Techniques
: This section explains how to implement applicable checkpoints in HTML (refer to [HTML40], [HTML32]) and includes numerous practical examples. To complement this section, an index of HTML elements and attributes provides information about all elements of HTML 4.0 and all attributes that affect accessibility directly. For each element, the index includes links to techniques that refer to it.

CSS Techniques
: This section explains how to implement applicable checkpoints in CSS1 and CSS2 (refer to [CSS1], [CSS2]).

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

2.1 Examples and Deprecated Examples

This document contains a number of examples that illustrate accessible solutions in HTML, CSS, etc. but also deprecated examples that illustrate what content developers should not do. The deprecated examples are highlighted and readers should approach them with caution -- they are meant for illustrative purposes only.

3 Accessibility Themes

The following sections discuss some accessibility themes that Web content developers should keep in mind as they design documents and sites.

3.1 Structure vs. Presentation

When designing a document or series of documents, content developers should strive first to identify the desired structure for their documents before thinking about how the documents will be presented to the user. Distinguishing the structure of a document from how the content is presented offers a number of advantages, including improved accessibility, manageability, and portability.

Identifying what is structure and what is presentation may be challenging at times. For instance, many content developers consider that a horizontal rule (the `HR` element) communicates a structural division. This may be true for sighted users, but to unsighted users or users without graphical browsers, a horizontal rule has next to no meaning (One might "guess" that an HR element implies a structural division, but without other information, there is no guarantee.) In HTML, content developers should use the HTML 4.0 header elements (`H1`-H6) to identify new sections. These may be *complemented* by visual or other cues such as horizontal rules, but should not be replaced by them.

The inverse holds as well: content developers should not use structural elements to achieve presentation effects. For instance in HTML, even though the `BLOCKQUOTE` element may cause indented text in some browsers, it is designed to identify a quotation, not create a presentation side-effect. BLOCKQUOTE elements used for indentation confuse users and search robots alike, who expect the element to be used to mark up block quotations.

The separation of presentation from structure in XML documents is inherent. As Norman Walsh states in "A Guide to XML" [WALSH],

> HTML browsers are largely hardcoded. A first level heading appears the way it does because the browser recognizes the `H1` tag. Again, since XML documents have no fixed tag set, this approach will not work. The presentation of an XML document is dependent on a stylesheet.

Quicktest! To determine if content is structural or presentational, create an outline of your document. Each point in the hierarchy denotes a structural change. Use structural markup to mark these changes and presentational markup to make them more apparent visually and aurally. Notice that horizontal rules will not appear in this outline and therefore are not structural, but presentational. **Note.** This quicktest addresses chapter, section, and paragraph structure. To determine structure within phrases, look for abbreviations, changes in natural language, definitions, and list items.

3.2 Text equivalents

Checkpoints in this section: 1.1, 1.2, 1.5, 13.10, 3.3, and 12.1, and 12.2.

Text is considered accessible to almost all users since it may be handled by screen readers, non-visual browsers, and braille readers. It may be displayed visually, magnified, synchronized with a video to create a caption, etc. As you design a document containing non-textual information (images, applets, sounds, multimedia presentations, etc.), think about supplementing that information with textual equivalents wherever possible.

A text equivalent describes the function or purpose of content. For complex content (charts, graphs, etc.), the text equivalent may be longer and include descriptive information.

Text equivalents should be provided for logos, photos, submit buttons, applets, bullets in lists, ascii art, and all of the links within an image map as well as invisible images used to lay out a page.

Quicktest! A good test to determine if a text equivalent is useful is to imagine reading the document aloud over the telephone. What would you say upon encountering this image to make the page comprehensible to the listener?

3.2.1 Overview of technologies

How one specifies a text equivalent depends on the document language.

For example, depending on the element, HTML allows content developers to specify text equivalents through attributes ("`alt`" or "`longdesc`") or in element content (the `OBJECT` element).

Video formats, such as Quicktime, will allow developers to include a variety of alternative audio and video tracks. SMIL ([SMIL]), allows developers to synchronize alternative audio and video clips, and text files with each other.

In creating XML DTDs, ensure that elements that might need a description have some way of associating themselves with the description.

Some image formats allow internal text in the data file along with the image information. If an image format supports such text (e.g., Portable Network Graphics, see [PNG]) content developers may also supply information there as well.

3.2.2 Backward Compatibility

Content developers must consider backward compatibility when designing Web pages or sites since:

- Some user agents do not support some HTML features,
- People may use older browsers or video players,
- Compatibility problems may arise between software

Therefore, when designing for older technologies, consider these techniques:

- Provide inline text equivalents. For example, include a description of the image immediately after the image.
- Provide links to long text equivalents either in a different file or on the same page. These are called description links or "d-links". The link text should explain that the link designates a description. Where possible, it should also explain the nature of the description. However, content developers concerned about how the description link will affect the visual appearance of the page may use more discrete link text such as "[D]", which is recommended by NCAM (refer to [NCAM]). In this case, they should also provide more information about the link target so that users can distinguish links that share "[D]" as content (e.g., with the "`title`" attribute in HTML).

3.3 Alternative pages

Checkpoints in this section: 11.4, and 6.5.

Although it is possible to make most content accessible, it may happen that all or part of a page remains inaccessible. Additional techniques for creating accessible alternatives include:

1. Allow users to navigate to a separate page that is accessible and maintained with the same frequency as the inaccessible original page.
2. Instead of static alternative pages, set up server-side scripts that generate accessible versions of a page on demand.
3. Refer to the examples for Frames and Scripts.
4. Provide a phone number, fax number, e-mail, or postal address where information is available and accessible, preferably 24 hours a day

Here are two techniques for linking to an accessible alternative page:

1. Provide links at the top of both the main and alternative pages to allow a user to move back and forth between them. For example, at the top of a graphical page include a link to the text-only page, and at the top of a text-only page include a link to the associated graphical page. Ensure that these links are one of the first that users will tab to by placing them at the top of the page, before other links.
2. Use meta information to designate alternative documents. Browsers should load the alternative page automatically based on the user’s browser type and preferences. For example, in HTML, use

the LINK element as follows:

Example.

User agents that support LINK will load the alternative page for those users whose browsers may be identified as supporting "aural","braille", or "tty" rendering.

```
<HEAD>
<TITLE>Welcome to the Virtual Mall!</TITLE>
<LINK title="Text-only version"
      rel="alternate"
      href="text_only"
      media="aural, braille, tty">
</HEAD>
<BODY><P>...</BODY>
```

End example.

3.4 Keyboard access

Checkpoints in this section: 9.2.

Not every user has a graphic environment with a mouse or other pointing device. Some users rely on keyboard, alternative keyboard or voice input to navigate links, activate form controls, etc. Content developers should always ensure that users may interact with a page with devices other than a pointing device. A page designed for keyboard access (in addition to mouse access) will generally be accessible to users with other input devices. What's more, designing a page for keyboard access will usually improve its overall design as well.

Keyboard access to links and form controls may be specified in a few ways:

Image map links
: Provide text equivalents for client-side image map areas, or provide redundant text links for server-side image maps. Refer to the image map section for examples.

Keyboard shortcuts
: Provide keyboard shortcuts so that users may combine keystrokes to navigate links or form controls on a page. **Note.** Keyboard shortcuts -- notably the key used to activate the shortcut -- may be handled differently by different operating systems. On Windows machines, the "alt" and "ctrl" key are most commonly used while on a Macintosh, it is the apple or "clover leaf" key. Refer to the Keyboard access for links and Keyboard Access to Forms sections for examples.

Tabbing order
: Tabbing order describes a (logical) order for navigating from link to link or form control to form control (usually by pressing the "tab" key, hence the name). Refer to the Keyboard Access to Forms section for examples.

3.4.1 Device-independent control for embedded interfaces

Some elements import objects (e.g., applets or multimedia players) whose interfaces cannot be controlled through the markup language. In such cases, content developers should provide alternative

equivalents with accessible interfaces if the imported objects themselves do not provide accessible interfaces.

3.5 Navigation

Checkpoints in this section: 14.3, 13.4, 13.5, 13.3, 13.7, and 13.2.

A consistent style of presentation on each page allows users to locate navigation mechanisms more easily but also to skip navigation mechanisms more easily to find important content. This helps people with learning and reading disabilities but also makes navigation easier for all users. Predictability will increase the likelihood that people will find information at your site, or avoid it when they so desire.

Examples of structures that may appear at the same place between pages:

1. navigation bars
2. the primary content of a page
3. advertising

A navigation mechanism creates a set of paths a user may take through your site. Providing navigation bars, site maps, and search features all increase the likelihood that a user will reach the information they seek at your site. If your site is highly visual in nature, the structure might be harder to navigate if the user can’t form a mental map of where they are going or where they have been. To help them, content developers should describe any navigation mechanisms. It is crucial that the descriptions and site guides be accessible since people who are lost at your site will rely heavily on them.

When providing search functionality, content developers should offer search mechanisms that satisfy varying skill levels and preferences. Most search facilities require the user to enter keywords for search terms. Users with spelling disabilities and users unfamiliar with the language of your site will have a difficult time finding what they need if the search requires perfect spelling. Search engines might include a spell checker, offer "best guess" alternatives, query-by-example searches, similarity searches, etc.

3.6 Comprehension

Checkpoints in this section: 14.1, 13.8, and 14.2.

The following sections discuss techniques for helping comprehension of a page or site.

3.6.1 Writing style

The following writing style suggestions should help make the content of your site easier to read for everyone, especially people with reading and/or cognitive disabilities. Several guides (including [HACKER]) discuss these and other writing style issues in more detail.

1. Strive for clear and accurate headings and link descriptions. This includes using link phrases that are terse and that make sense when read out of context or as part of a series of links (Some users browse by jumping from link to link and listening only to link text.) Use informative headers so that users can scan a page quickly for information rather than reading it in detail.

2. State the topic of the sentence or paragraph at the beginning of the sentence or paragraph (this is called "front-loading"). This will help both people who are skimming visually, but also people who use speech synthesizers. "Skimming" with speech currently means that the user jumps from heading to heading, or paragraph to paragraph and listens to just enough words to determine whether the current chunk of information (heading, paragraph, link, etc.) interests them. If the main idea of the paragraph is in the middle or at the end, speech users may have to listen to most of the document before finding what they want. Depending on what the user is looking for and how much they know about the topic, search features may also help users locate content more quickly.
3. Limit each paragraph to one main idea.
4. Avoid slang, jargon, and specialized meanings of familiar words, unless defined within your document.
5. Favor words that are commonly used. For example, use "begin" rather than "commence" or use "try" rather than "endeavor."
6. Use active rather than passive verbs.
7. Avoid complex sentence structures.

To help determine whether your document is easy to read, consider using the Gunning-Fog reading measure (described in [SPOOL] with examples and the algorithm online at [TECHHEAD]). This algorithm generally produces a lower score when content is easier to read. As example results, the Bible, Shakespeare, Mark Twain, and TV Guide all have Fog indexes of about 6. Time, Newsweek, and the Wall St. Journal an average Fog index of about 11.

3.6.2 Multimedia equivalents

For people who do not read well or not at all, multimedia (non-text) equivalents may help facilitate comprehension. Beware that multimedia presentations do not **always** make text easier to understand. Sometimes, multimedia presentations may make it more confusing.

Examples of multimedia that supplement text:

1. A chart of complex data, such as sales figures of a business for the past fiscal year.
2. A translation of the text into a Sign Language movie clip. Sign Language is a very different language than spoken languages. For example, some people who may communicate via American Sign Language may not be able to read American English.
3. Pre-recorded audio of music, spoken language, or sound effects may also help non-readers who can perceive audio presentations. Although text may be generated as speech through speech synthesis, changes in a recorded speaker’s voice can convey information that is lost through synthesis.

3.7 Content negotiation

Checkpoints in this section: 11.3.

1. Instead of including links such as "Here is the French version of this document", use content negotiation so that the French version is served to clients requesting French versions of documents.
2. If not possible to use content negotiation, indicate content type or language through markup (e.g.,

in HTML use "`type`" and "`hreflang`").

3.8 Automatic page refresh

Checkpoints in this section: 7.4, and 7.5.

Content developers sometimes create pages that refresh or change without the user requesting the refresh. This automatic refresh can be very disorienting to some users. Instead, in order of preference, authors should:

1. Configure the server to use the appropriate HTTP status code (301). Using HTTP headers is preferable because it reduces Internet traffic and download times, it may by applied to non-HTML documents, and it may be used by agents who requested only a HEAD request (e.g., link checkers). Also, status codes of the 30x type provide information such as "moved permanently" or "moved temporarily" that cannot be given with META refresh.
2. Replace the page that would be redirected with a static page containing a normal link to the new page.

Note. Both checkpoint 7.4 and checkpoint 7.5 address problems posed by legacy user agents. Newer user agents should disable refresh and substitute a link to new information at the top of the page.

The following are **deprecated** HTML examples. The first changes the user's page at page at regular intervals. Content developers should **not** use this technique to simulate "push" technology. Developers cannot predict how much time a user will require to read a page; premature refresh can disorient users. Content developers should avoid periodic refresh and allow users to choose when they want the latest information.

Deprecated example.

```
<META http-equiv="refresh" content="60">
<BODY>
<P>...Information...
</BODY>
```

The following HTML example (using the `META` element) forwards the user from one page to another after a timeout. However, users should **not** redirect users with this markup since is non-standard, it disorients users, and it can disrupt a browser's history of visited pages.

Deprecated example.

```
<HEAD>
<TITLE>Don't use this!</TITLE>
<META http-equiv="refresh" content="5;
         http://www.acme.com/newpage">
</HEAD>
<BODY>
<P>If your browser supports Refresh,
you'll be transported to our
<A href="http://www.acme.com/newpage">new site</A>
in 5 seconds, otherwise, select the link manually.
</BODY>
```

3.9 Screen flicker

Checkpoints in this section: 7.1.

A flickering or flashing screen may cause seizures in users with photosensitive epilepsy and content developers should thus avoid causing the screen to flicker. Seizures can be triggered by flickering or flashing in the 4 to 59 flashes per second (Hertz) range with a peak sensitivity at 20 flashes per second as well as quick changes from dark to light (like strobe lights).

3.10 Bundled documents

Checkpoints in this section: 13.9.

Bundled documents can facilitate reading offline. To create a coherent package:

- Use metadata to describe the relationships between components of the package (refer to link metadata for HTML).
- Use archiving tools such as zip, tar and gzip, and stuffit to create the package.

3.11 Validation

This section discusses strategies and techniques for testing Web documents to determine accessibility issues that have been resolved and those that haven't. These tests should highlight major access issues, are valuable in reducing a number of accessibility barriers. However, some of these testing scenarios only replicate conditions caused by a disability; they do not simulate the full experience a user with a disability might have. In real-life settings, your pages may be less usable than you expected. Thus, one of the strategies recommends that content developers observe people with different disabilities as they attempt to use a page or site.

If, after completing the following tests and adjusting your design accordingly, you find that a page is still not accessible, it is likely that you should create an alternative page that is accessible.

Note. Passing these tests does not guarantee conformance to the "Web Content Accessibility Guidelines 1.0".

3.11.1 Automatic validators

A validator can verify the syntax of your pages (e.g., HTML, CSS, XML). Correct syntax will help eliminate a number of accessibility problems since software can process well-formed documents more easily. Also, some validators can warn you of some accessibility problems based on syntax alone (e.g., a document is missing an attribute or property that is important to accessibility). Note, however, that correct syntax does not guarantee that a document will be accessible. For instance, you may provide a text equivalent for an image according to the language's specification, but the text may be inaccurate or insufficient. Some validators will therefore ask you questions and step you through more subjective parts of the analysis. Some examples of automatic validators include:

1. An automated accessibility validation tool such as Bobby (refer to [BOBBY]).

2. An HTML validation service such as the W3C HTML Validation Service (refer to [HTMLVAL]).
3. A style sheets validation service such as the W3C CSS Validation Service (refer to [CSSVAL]).

3.11.2 Repair tools

Validators usually report what issues to solve and often give examples of how to solve them. They do not usually help an author walk through each problem and help the author modify the document interactively. The WAI Evaluation and Repair Working Group ([WAI-ER]) is working to develop a suite of tools that will help authors not only identify issues but solve them interactively.

3.11.3 User scenarios

Keep in mind that most user agents (browsers) and operating systems allow users to configure settings that change the way software looks, sounds, and behaves. With the variety of user agents, different users will have very different experiences with the Web. Therefore:

1. Test your pages with a text-only browser such as Lynx ([LYNX]) or a Lynx emulator such as Lynx Viewer ([LYNXVIEW]) or Lynx-me ([LYNXME]).
2. Use multiple graphic browsers, with:
    - sounds and images loaded,
    - images not loaded,
    - sounds not loaded,
    - no mouse,
    - frames, scripts, style sheets, and applets not loaded.
3. Use several browsers, old and new. **Note.** Some operating systems or browsers do not allow multiple installations of the browser on the same machine. It may also be difficult to locate older browser software.
4. Use other tools such as a self-voicing browser (e.g., [PWWEBSPEAK] and [HOMEPAGEREADER]), a screen reader (e.g., [JAWS] and [WINVISION]), magnification software, a small display, an onscreen keyboard, an alternative keyboard, etc.

3.11.4 Spell and grammar checks

A person reading a page with a speech synthesizer may not be able to decipher the synthesizer's best guess for a word with a spelling error. Grammar checkers will help to ensure that the textual content of your page is correct. This will help readers for whom your document is not written in their native tongue, or people who are just learning the language of the document. Thus, you will help increase the comprehension of your page.

3.12 Browser Support

Checkpoints in this section: 11.1.

Note. *At the time of this writing, not all user agents support some of the (new) HTML 4.0 attributes and elements that may significantly increase accessibility of Web pages.*

Please refer to the W3C Web site ([WAI-UA-SUPPORT]) for information about browser and other user agent support of accessibility features.

In general, please note that HTML user agents ignore attributes they don’t support and they render the content of unsupported elements.

---

4 HTML Techniques

The following sections list some techniques for designing accessible HTML documents. The sections are organized by topic (and mirror the organization of the HTML 4.0 specification, [HTML40]).

4.1 Document structure and metadata

Checkpoints in this section: 3.2

Content developers should use structural markup and use it according to specification. Structural elements and attribute (refer to the index of HTML elements and attributes to identify them) promote consistency in documents and supply information to other tools (e.g., indexing tools, search engines, programs that extract tables to databases, navigation tools that use header elements, and automatic translation software that translates text from one language into another.

4.1.1 Metadata

Checkpoints in this section: 13.2.

Some structural elements provide information about the document itself. This is called "metadata" about the document -- Metadata is information about data. Well-crafted metadata can provide important orientation information to users. HTML elements that provide useful information about a document include:

- `TITLE`: The document title. Note that the (mandatory) TITLE element, which only appears once in a document, is different from the "`title`" attribute, which applies to almost every HTML 4.0 element. Content developers should use the "title" attribute in accordance with the HTML 4.0 specification. For example, "title" should be used with links to provide information about the target of the link.
- `ADDRESS`: Can be used to provide information about the creator of the page.
- `LINK`: Can be used to indicate alternative documents (different structure, different language, different target device, etc.).
- The `META` element can specify arbitrary metadata for a document. Please refer to the section on automatic page refresh for information on why META should **not** be used to redirect pages.

4.1.2 Section headers

Checkpoints in this section: 3.5.

Sections should be introduced with the HTML header elements (`H1`-H6). Other markup may complement these elements to improve presentation (e.g., the `HR` element to create a horizontal dividing line), but visual presentation is not sufficient to identify document sections.

Since some users skim through a document by navigating its headings, it is important to use them appropriately to convey document structure. Users should order heading elements properly. For example, in HTML, H2 elements should follow H1 elements, H3 elements should follow H2 elements, etc. Content developers should not "skip" levels (e.g., H1 directly to H3). Do not use headings to create font effects; use style sheets to change font styles for example.

Note that in HTML, heading elements (H1 - H6) only start sections, they don't contain them as element content. The following HTML markup shows how style sheets may be used to control the appearance of a header and the content that follows:

Example.

```
<HEAD>
<TITLE>Cooking techniques</TITLE>
<STYLE type="text/css">
   /* Indent header and following content */
   DIV.section2 { margin-left: 5% }
</STYLE>
</HEAD>
<BODY>
<H1>Cooking techniques</H1>
... some text here ...
<DIV class="section2">
<H2>Cooking with oil</H2>
... text of the section ...
</DIV>

<DIV class="section2">
<H2>Cooking with butter</H2>
... text of the section ...
</DIV>
```

End example.

4.1.3 Link metadata and navigation tools

Checkpoints in this section: 13.2.

Content developers should use the `LINK` element and link types (refer to [HTML40], section 6.12) to describe document navigation mechanisms and organization. Some user agents may synthesize navigation tools or allow ordered printing of a set of documents based on such markup.

Example.

The following `LINK` elements might be included in the head of chapter 2 of a book:

```
<LINK rel="Next" href="chapter3">
<LINK rel="Previous" href="chapter1">
<LINK rel="Start" href="cover">
<LINK rel="Glossary" href="glossary">
```

End example.

See also the section on links.

4.1.4 Structural grouping

Checkpoints in this section: 12.3.

The following HTML 4.0 mechanisms group content and make it easier to understand.:

- Use `FIELDSET` to group form controls into semantic units and describe the group with the `LEGEND` element.
- Use `OPTGROUP` to organize long lists of menu options into smaller groups..
- Use tables for tabular data and describe the table with `CAPTION`.
- Group table rows and columns with `THEAD`, `TBODY`, `TFOOT`, and `COLGROUP`.
- Nest lists with `UL`, `OL`, and `DL`.
- Use section headers (`H1` - H6) to create structured documents and break up long stretches of text.
- Break up lines of text into paragraphs (with the `P` element).

All of these grouping mechanisms should be used when appropriate and natural, i.e., when the information lends itself to logical groups. Content developers should not create groups randomly, as this will confuse all users.

4.2 Language information

Checkpoints in this section: 4.1, and 4.3.

If you use a number of different languages on a page, make sure that any changes in language are clearly identified by using the "`lang`" attribute:

Example.

```
<P>And with a certain <SPAN lang="fr">je ne sais quoi</SPAN>,
she entered both the room, and his life, forever. <Q>My name
is Natasha,</Q> she said. <Q lang="it">Piacere,</Q>
he replied in impeccable Italian, locking the door.
```

End example.

Identifying changes in language are important for a number of reasons:

1. Users who are reading the document in braille will be able to substitute the appropriate control codes (markup) where language changes occur to ensure that the braille translation software will generate the correct characters (accented characters, for instance). These control codes also prevent braille contractions from being generated, which could further confuse the user. Braille contractions combine commonly used groups of characters that usually appear in multiple cells into a single cell. For example, "ing" which usually takes up three cells (one for each character) can be contracted into a single cell.
2. Similarly, speech synthesizers that "speak" multiple languages will be able to generate the text in the appropriate accent with proper pronunciation. If changes are not marked, the synthesizer will try its best to speak the words in the primary language it works in. Thus, the French word for car,

"voiture" would be pronounced "voter."
3. Users who are unable to translate between languages themselves, will be able to have unfamiliar languages translated by machine translators.

It is also good practice to identify the primary language of a document, either with markup (as shown below) or through HTTP headers.

Example.

```
<HTML lang="fr">
   ....rest of an HTML document written in French...
</HTML>
```

End example.

4.3 Text markup

The following sections discuss ways to add structure to pieces of text.

4.3.1 Emphasis

Checkpoints in this section: 3.3.

The proper HTML elements should be used to mark up emphasis: `EM` and `STRONG`. The `B` and `I` elements should not be used; they are used to create a visual presentation effect. The EM and STRONG elements were designed to indicate structural emphasis that may be rendered in a variety of ways (font style changes, speech inflection changes, etc.)

4.3.2 Acronyms and abbreviations

Checkpoints in this section: 4.2.

Mark up abbreviations and acronyms with `ABBR` and `ACRONYM` and use "`title`" to indicate the expansion:

Example.

```
<P>Welcome to the <ACRONYM title="World Wide Web">WWW</ACRONYM>!
```

End example.

4.3.3 Quotations

Checkpoints in this section: 3.7.

The `Q` and `BLOCKQUOTE` elements mark up inline and block quotations, respectively.

Example.

This example marks up a longer quotation with `BLOCKQUOTE`:

```
<BLOCKQUOTE cite="http://www.shakespeare.com/loveslabourlost">
  <P>Remuneration! O! that's the Latin word for three farthings.
     --- William Shakespeare (Love's Labor Lost).
  </P>
</BLOCKQUOTE>
```

End example.

4.3.4 Markup and style sheets rather than images: The example of math

Checkpoints in this section: 3.1.

Using markup (and style sheets) where possible rather than images (e.g., a mathematical equation) promotes accessibility for the following reasons:

- Text may be magnified or interpreted as speech or braille.
- Search engines can use text information.

As an example, consider these techniques for putting mathematics on the Web:

- Ensure that users know what variables represent, for example, in the equation "F = m * a", indicate that F= Force, m = mass, a = acceleration.
- For straightforward equations, use characters, as in "x + y = z"
- For more complex equations, mark them up with MathML ([MATHML]) or TeX. **Note.** MathML can be used to create very accessible documents but currently is not as widely supported or used as TeX.
- Provide a text description of the equation and, where possible, use character entity references to create the mathematical symbols. A text equivalent must be provided if the equation is represented by one or more images.

TeX is commonly used to create technical papers which are then converted to HTML for publication on the Web. However, converters tend to generate images, use deprecated markup, and use tables for layout. Consequently, content providers should:

1. Make the original TeX (or LaTeX) document available on the Web. There is a system called "AsTeR" ([ASTER]) that can create an auditory rendition of TeX and LaTeX documents. Also, IBM has a plug-in for Netscape and Internet Explorer that reads TeX/LaTeX documents and some of MathML (refer to [HYPERMEDIA]).
2. Ensure that the HTML created by the conversion process is accessible. Provide a single description of the equation (rather than "alt" text on every generated image as there may be small images for bits and pieces of the equation).

4.3.5 Miscellaneous structural markup

The HTML 4.0 specification defines the following structural elements for miscellaneous markup needs:

`CITE`
: Contains a citation or a reference to other sources.

`DFN`

Indicates that this is the defining instance of the enclosed term.

`CODE`
Designates a fragment of computer code.

`SAMP`
Designates sample output from programs, scripts, etc.

`KBD`
Indicates text to be entered by the user.

`VAR`
Indicates an instance of a variable or program argument.

`INS`
Indicates text inserted into a document.

`DEL`
Indicates text deleted from a document.

4.4 Lists

Checkpoints in this section: 3.6.

The HTML list elements `DL`, `UL`, and `OL` should only be used to create lists, not for formatting effects such as indentation.

Ordered lists help non-visual users navigate. Non-visual users may "get lost" in lists, especially in nested lists and those that do not indicate the specific nest level for each list item. Until user agents provide a means to identify list context clearly (e.g., by supporting the ':before' pseudo-element in CSS2), content developers should include contextual clues in their lists.

For numbered lists, compound numbers are more informative than simple numbers. Thus, a list numbered "1, 1.1, 1.2, 1.2.1, 1.3, 2, 2.1," provides more context than the same list without compound numbers, which might be formatted as follows:

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

and would be spoken as "1, 1, 2, 1, 2, 3, 2, 1", conveying no information about list depth.

[CSS1] and [CSS2] allow users to control number styles (for all list, not just ordered) through user style sheets.

Example.

The following CSS2 style sheet 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.

Until either CSS2 is widely supported or user agents allow users to control rendering of lists through other means, authors should consider providing contextual clues in unnumbered nested lists. Non-visual users may have difficulties knowing where a list begins and ends and where each list item starts. For example, if a list entry wraps to the next line on the screen, it may appear to be two separate items in the list. This may pose a problem for legacy screen readers.

4.4.1 Use style sheets to change list bullets

To change the "bullet" style of unordered list items created with the `LI` element, use style sheets. In CSS, it is possible to specify a fallback bullet style (e.g., ’disc’) if a bullet image cannot be loaded.

Example.

```
<HEAD>
<TITLE>Using style sheets to change bullets</TITLE>
<STYLE type="text/css">
   UL { list-style: url(star.gif) disc }
</STYLE>
</HEAD>
<BODY>
<UL>
   <LI>Audrey
   <LI>Laurie
   <LI>Alice
</UL>
```

End example.

To further ensure that users understand differences between list items indicated visually, content developers should provide a text label before or after the list item phrase:

Example.

In this example, new information is communicated through text ("New"), font style (bold), and color (yellow bullet, red text on yellow background).

```
<HEAD>
<TITLE>Bullet styles example</TITLE>
<STYLE type="text/css">
   .newtxt { font-weight: bold;
             color: red;
             background-color: yellow }
   .newbullet { list-style : url(yellow.gif) disc }
</STYLE>
</HEAD>
<BODY>
<UL>
   <LI class="newbullet">Roth IRA <SPAN class="newtext">New</SPAN></LI>
```

```
   <LI> 401(k)</LI>
</UL>
</BODY>
```

End example.

Avoid using images as bullets in definition lists created with `DL`, `DT`, and `DD`. However, if this method is used, be sure to provide a text equivalent for the images.

Deprecated example.

```
<HEAD>
<TITLE>Deprecated example using image in DL lists</TITLE>
</HEAD>
<BODY>
<DL>
   <DD><IMG src="star.gif" alt="* ">Audrey
   <DD><IMG src="star.gif" alt="* ">Laurie
   <DD><IMG src="star.gif" alt="* ">Alice
</DL>
```

Content developers should avoid list styles where bullets provide additional (visual) information. However, if this is done, be sure to provide a text equivalent describing meaning of the bullet:

Deprecated example.

```
<DL>
<DD><IMG src="red.gif" alt="New:">Roth IRA</DD>
<DD><IMG src="yellow.gif" alt="Old:">401(k)</DD>
</DL>
```

4.5 Tables

This section discusses the accessibility of tables and elements that one can put in a `TABLE` element. Two types of tables are discussed: tables used to organize data, and tables used to create a visual layout of the page.

4.5.1 Tables of data

Checkpoints in this section: 5.5, 5.1, 5.2, and 5.6.

Content developers may make HTML 4.0 data tables more accessible in a number of ways:

- Identify structural groups of rows (`THEAD` for repeated table headers, `TFOOT` for repeated table footers, and `TBODY` for other groups of rows) and groups of columns (`COLGROUP` and `COL`). Label table elements with the "`scope`", "`headers`", and "`axis`" attributes so that future browsers and assistive technologies will be able to select data from a table by filtering on categories. This markup will also help browsers linearize tables (also called table "serialization". A row-based linear version may be created by reading the row header, then preceding each cell with the cell’s column header. Or, the linearization might be column-based. Note that the natural language writing direction may affect column layout (and thus ordering). In HTML, the "`dir`" attribute specifies column layout order (e.g., dir="rtl" specifies right-to-left layout).

- Do not use PRE to create a tabular layout of text -- use the TABLE element so that assistive technologies may recognize that it is a table.
- Provide a caption via the CAPTION element.
- Provide a summary via the "summary" attribute. Summaries are especially useful for non-visual readers.
- Provide terse substitutes for header labels with the "abbr" attribute on TH. These will be particularly useful for future speaking technologies that can read row and column labels for each cell. Abbreviations cut down on repetition and reading time.
- Future browsers and assistive technologies will be able to automatically translate tables into linear sequences or navigate a table cell by cell if data is labeled appropriately. The WAI Evaluation and Repair working group is tracking the progress of tools as well as developing their own that will allow users to navigate tables cell by cell. Refer to [WAI-ER].

This markup will allow accessible browsers and other user agents to restructure or navigate tables in a non-visual manner.

For information about table headers, refer to the table header algorithm and discussion in the HTML 4.0 Recommendation ([HTML40], section 11.4.3).

Example.

This example shows how to associate data cells (created with TD) with their corresponding headers by means of the "headers" attribute. The "headers" attribute specifies a list of header cells (row and column labels) associated with the current data cell. This requires each header cell to have an "id" attribute.

```
<TABLE border="1"
       summary="This table charts the number of
                cups of coffee consumed by each senator,
                the type of coffee (decaf or regular),
                and whether taken with sugar.">
  <CAPTION>Cups of coffee consumed by each senator</CAPTION>
  <TR>
      <TH id="header1">Name</TH>
      <TH id="header2">Cups</TH>
      <TH id="header3" abbr="Type">Type of  Coffee</TH>
      <TH id="header4">Sugar?</TH>
  <TR>
      <TD headers="header1">T. Sexton</TD>
      <TD headers="header2">10</TD>
      <TD headers="header3">Espresso</TD>
      <TD headers="header4">No</TD>
  <TR>
      <TD headers="header1">J. Dinnen</TD>
      <TD headers="header2">5</TD>
      <TD headers="header3">Decaf</TD>
     <TD headers="header4">Yes</TD>
</TABLE>
```

End example.

A speech synthesizer might render this tables as follows:

```
Caption: Cups of coffee consumed by each senator
Summary: This table charts the number of cups of coffee
         consumed by each senator, the type of coffee
        (decaf or regular), and whether taken with sugar.
Name: T. Sexton, Cups: 10, Type: Espresso, Sugar: No
Name: J. Dinnen, Cups: 5, Type: Decaf, Sugar: Yes
```

A visual user agent might render this table as follows:

Cups of coffee consumed by each senator

Name	Cups	Type of Coffee	Sugar?
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

[Description of coffee table]

The next example associates the same header (`TH`) and data (`TD`) cells as before, but this time uses the "`scope`" attribute rather than "`headers`". "Scope" must have one of the following values: "row", "col", "rowgroup", or "colgroup." Scope specifies the set of data cells to be associated with the current header cell. This method is particularly useful for simple tables. It should be noted that the spoken rendering of this table would be identical to that of the previous example. A choice between the "headers" and "scope" attributes is dependent on the complexity of the table. It does not affect the output so long as the relationships between header and data cells are made clear in the markup.

Example.

```
<TABLE border="1"
       summary="This table charts ...">
   <CAPTION>Cups of coffee consumed by each senator</CAPTION>
   <TR>
        <TH scope="col">Name</TH>
        <TH scope="col">Cups</TH>
        <TH scope="col" abbr="Type">Type of Coffee</TH>
        <TH scope="col">Sugar?</TH>
  <TR>
        <TD>T. Sexton</TD>  <TD>10</TD>
        <TD>Espresso</TD>   <TD>No</TD>
  <TR>
        <TD>J. Dinnen</TD>  <TD>5</TD>
        <TD>Decaf</TD>       <TD>Yes</TD>
</TABLE>
```

End example.

The following example shows how to create categories within a table using the "`axis`" attribute.

Example.

```
<TABLE border="1">
  <CAPTION>Travel Expense Report</CAPTION>
  <TR>
       <TH></TH>
       <TH id="header2" axis="expenses">Meals
       <TH id="header3" axis="expenses">Hotels
       <TH id="header4" axis="expenses">Transport
       <TD>subtotals</TD>
```

```
    <TR>
         <TH id="header6" axis="location">San Jose
         <TH> <TH> <TH> <TD>
    <TR>
        <TD id="header7" axis="date">25-Aug-97
        <TD headers="header6 header7 header2">37.74
        <TD headers="header6 header7 header3">112.00
        <TD headers="header6 header7 header4">45.00
        <TD>
    <TR>
        <TD id="header8" axis="date">26-Aug-97
        <TD headers="header6 header8 header2">27.28
        <TD headers="header6 header8 header3">112.00
        <TD headers="header6 header8 header4">45.00
        <TD>
    <TR>
        <TD>subtotals
        <TD>65.02
        <TD>224.00
        <TD>90.00
        <TD>379.02
    <TR>
        <TH id="header10" axis="location">Seattle
        <TH> <TH> <TH> <TD>
    <TR>
        <TD id="header11" axis="date">27-Aug-97
        <TD headers="header10 header11 header2">96.25
        <TD headers="header10 header11 header3">109.00
        <TD headers="header10 header11 header4">36.00
        <TD>
    <TR>
        <TD id="header12" axis="date">28-Aug-97
        <TD headers="header10 header12 header2">35.00
        <TD headers="header10 header12 header3">109.00
        <TD headers="header10 header12 header4">36.00
        <TD>
    <TR>
        <TD>subtotals
        <TD>131.25
        <TD>218.00
        <TD>72.00
        <TD>421.25
    <TR>
        <TH>Totals
        <TD>196.27
        <TD>442.00
        <TD>162.00
        <TD>800.27
</TABLE>
```

End example.

This table lists travel expenses at two locations: San Jose and Seattle, by date, and category (meals, hotels, and transport). The following image shows how a visual user agent might render it. [Description of travel table]

Travel Expense Report

	Meals	Hotels	Transport	subtotals
San Jose				
25-Aug-97	37.74	112.00	45.00	
26-Aug-97	27.28	112.00	45.00	
subtotals	65.02	224.00	90.00	379.02
Seattle				
27-Aug-97	96.25	109.00	36.00	
28-Aug-97	35.00	109.00	36.00	
subtotals	131.25	218.00	72.00	421.25
Totals	196.27	442.00	162.00	**800.27**

4.5.2 Avoid tables for layout

Checkpoints in this section: 5.3 and 5.4.

Authors should use style sheets for layout and positioning. However, when it is necessary to use a table for layout, the table must linearize in a readable order. When a table is linearized, the contents of the cells become a series of paragraphs (e.g., down the page) one after another. Cells should make sense when read in order (row-wise or column-wise) and should include structural elements (that create paragraphs, headers, lists, etc.) so the page makes sense after linearization.

Also, when using tables to create a layout, do not use structural markup to create visual formatting. For example, the TH (table header) element, is usually displayed visually as centered, and bold. If a cell is not actually a header for a row or column of data, use style sheets or formatting attributes of the element.

4.5.3 Wrapped text in tables

Checkpoints in this section: 10.3.

Tables used to lay out pages and some data tables where cell text wraps pose problems for older screen readers that do not interpret the source HTML or browsers that do not allow navigation of individual table cells. These screen readers will read across the page, reading sentences on the same row from different columns as one sentence.

For example, if a table is rendered like this on the screen:

```
There is a 30% chance of              Classes at the University of Wisconsin
rain showers this morning, but they   will resume on September 3rd.
should stop before the weekend.
```

This might be read by a screen reader as:

```
There is a 30% chance of Classes at the University of Wisconsin
rain showers this morning, but they will resume on September 3rd.
should stop before the weekend.
```

Screen readers that read the source HTML will recognize the structure of each cell, but for older screen readers, content developers may minimize the risk of word wrapping by limiting the amount of text in each cell. Also, the longest chunks of text should all be in the last column (rightmost for left-to-right

tables). This way, if they wrap, they will still be read coherently. Content developers should test tables for wrapping with a browser window dimension of "640x480".

Since table markup is structural, and we suggest separating structure from presentation, we recommend using style sheets to create layout, alignment, and presentation effects. Thus, the two columns in the above example could have been created using style sheets. Please refer to the section on style sheets for more information.

Quicktest! To get a better understanding of how a screen reader would read a table, run a piece of paper down the page and read your table line by line.

4.5.4 Backward compatibility issues for tables

In HTML 3.2 browsers, the rows of a `TFOOT` element will appear before the table body.

4.6 Links

Checkpoints in this section: 13.1, and 13.6.

Good link text should not be overly general; don’t use "click here." Not only is this phrase device-dependent (it implies a pointing device) it says nothing about what is to be found if the link if followed. Instead of "click here", link text should indicate the nature of the link target, as in "more information about sea lions" or "text-only version of this page". Note that for the latter case (and other format- or language-specific documents), content developers are encouraged to use content negotiation instead, so that users who prefer text versions will have them served automatically.

In addition to clear link text, content developers may specify a value of the "title" attribute that clearly and accurately describes the target of the link.

If more than one link on a page shares the same link text, all those links should point to the same resource. Such consistency will help page design as well as accessibility.

If two or more links refer to different targets but share the same link text, distinguish the links by specifying a different value for the "`title`" attribute of each `A` element.

"Auditory users" -- people who are blind, have difficulty seeing, or who are using devices with small or no displays -- are unable to scan the page quickly with their eyes. To get an overview of a page or to quickly find a link, these users will often tab from one link to the next or review a list of available links on a page.

Thus, for a series of related links, include introductory information in the first link, then distinguishing information in the links that follow. This will provide context information for users reading them in sequence.

Example.

```
<A href="my-doc.html">My document is available in HTML</A>,
<A href="my-doc.pdf" title="My document in PDF">PDF</A>,
<A href="my-doc.txt" title="My document in text">plain text</A>
```

End example.

When an image is used as the content of a link, specify a text equivalent for the image.

Example.

```
<A href="routes.html">
   <IMG src="topo.html"
        alt="Current routes at Boulders Climbing Gym">
</A>
```

End example.

4.6.1 Grouping and bypassing links

When links are grouped into logical sets (for example, in a navigation bar that appears on every page in a site) they should be marked up as a unit. Navigation bars are usually the first thing someone encounters on a page. For users with speech synthesizers, this means having to hear a number of links on every page before reaching the interesting content of a page. There are several ways to allow users to bypass groups of links (as users with vision do when they see the same set on each page):

- Include a link that allows users to skip over the set of navigation links.
- Use the HTML 4.0 "`tabindex`" attribute to allow users to jump to an anchor after the set of navigation links. This attribute is not yet widely supported.
- Provide a style sheet that allows users to hide the set of navigation links.

In the future, user agents will allow users to skip over elements such as navigation bars.

In HTML, use the `DIV`, `SPAN`, `P`, or `FRAME` elements to group links then identify the group with the "`id`" or "`class`" attributes.

Example.

In this example, the `P` element groups a set of links, the "class" attribute identifies it as a navigation bar (e.g., for style sheets), "`tabindex`" is set on an anchor following the group, and a link at the beginning of the group links to the anchor after the group.

```
<HEAD>
<TITLE>How to use our site</TITLE>
</HEAD>
<BODY>
  <P class="nav">
    [<A href="#how">Bypass navigation bar</A>]
    [<A href="home.html">Home</A>]
    [<A href="search.html">Search</A>]
    [<A href="new.html">New and highlighted</A>]
    [<A href="sitemap.html">Site map</A>]
  </P>
  <H1><A name="how" tabindex="1">How to use our site</A></H1>
<!-- content of page -->
</BODY>
```

End example.

4.6.2 Keyboard access

Keyboard access to active elements of a page is important for many users who cannot use a pointing device. User agents may include features that allow users to bind keyboard strokes to certain actions. HTML 4.0 also allows content developers to specify keyboard shortcuts in documents via the "`tabindex`" attribute.

Example.

In this example, if the user activates the "C" key, the link will be followed.

```
<A accesskey="C" href="doc.html" hreflang="en"
   title="XYZ company home page">
      XYZ company home page</A>
```

End example.

4.7 Images and image maps

The following sections discuss accessibility of images (including simple animations such as GIF animations) and image maps.

For information about math represented as images, refer to the section on using text markup and style sheets rather than images.

4.7.1 Text equivalents for images

Checkpoints in this section: 1.1.

When using `IMG`, specify a short text equivalent with the "`alt`" attribute. **Note.** The value of this attribute is referred to as "alt-text".

Example.

```
<IMG src="magnifyingglass.gif" alt="Search">
```

End example.

When using `OBJECT`, specify a text equivalent in the body of the OBJECT element:

Example.

```
<OBJECT data="magnifyingglass.gif" type="image/gif">
   Search
</OBJECT>
```

End example.

When a short text equivalent does not suffice to adequately convey the function or role of an image, provide additional information in a file designated by the "longdesc" attribute:

Example.

```
<IMG src="97sales.gif" alt="Sales for 1997"
     longdesc="sales97.html">
```

In sales97.html:

> A chart showing how sales in 1997 progressed. The chart
> is a bar-chart showing percentage increases in sales
> by month. Sales in January were up 10% from December 1996,
> sales in February dropped 3%, ..

End example.

For user agents that don’t support "longdesc", provide a description link as well next to the graphic:

Example.

```
<IMG src="97sales.gif" alt="Sales for 1997" longdesc="sales.html">
<A href="sales.html" title="Description of 1997 sales figures">[D]</A>
```

End example.

When using OBJECT, specify the the longer text equivalent within the element’s content:

Example.

```
<OBJECT data="97sales.gif" type="image/gif">
       Sales in 1997 were down subsequent to our
       <A href="anticipated.html">anticipated
       purchase</A> ...
</OBJECT>
```

End example.

Note that OBJECT content, unlike "alt" text, can include markup. Thus, content developers can provide a link to additional information from within the OBJECT element:

Example.

```
<OBJECT data="97sales.gif" type="image/gif">
       Chart of our Sales in 1997.
       A <A href="desc.html">textual description</A> is available.
</OBJECT>
```

End example.

4.7.2 Invisible d-links

Note. Invisible d-links are deprecated in favor of the "longdesc" attribute.

An invisible d-link is a small (1-pixel) or transparent image whose "`alt`" attribute value is "D-link" or "D" and is part of the content of an `A` element. Like other d-links, it refers to a text equivalent of the associated image. Like other links, users can tab to it. Invisible d-links thus provide a (temporary) solution for designers who wish to avoid visible d-links for stylistic reasons.

4.7.3 Ascii art

Checkpoints in this section: 13.10.

Avoid ascii art (character illustrations) and use real images instead since it is easier to supply a text equivalent for images. However, if ascii art must be used provide a link to jump over the ASCII art, as follows.

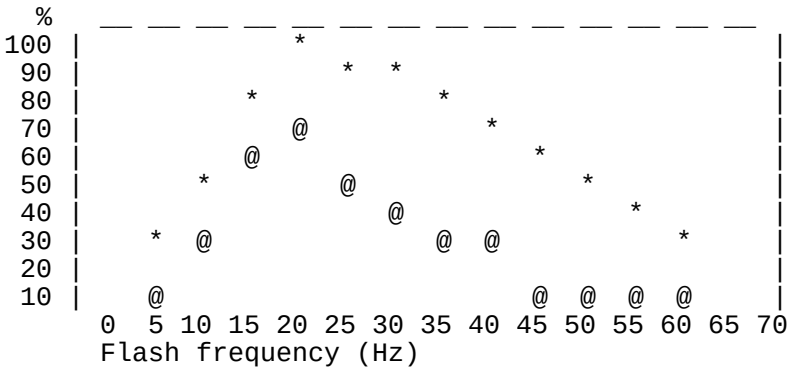
Example.

```
<P>
<a href="#post-art">skip over ASCII art</a>
<!-- ASCII art goes here -->
<a name="post-art">caption for ASCII art</a>
```

End example.

ASCII art may also be marked up as follows [skip over ascii figure or consult a description of chart]:

Example.



End example.

Another option for smaller ascii art is to use an `ABBR` element with "`title`".

Example.

```
<P><ABBR title="smiley in ascii art">:-)</ABBR>
```

End example.

If the ASCII art is complex, ensure that the text equivalent adequately describes it.

Another way to replace ascii art is to use human language substitutes. For example, <wink> might substitute for a winking smiley: ;-). Or, the word "therefore" can replace arrows consisting of dashes and greater than signs (e.g., -->), and the word "great" for the uncommon abbreviation "gr8".

4.7.4 Image maps

An image map is an image that has "active regions". When the user selects one of the regions, some action takes place -- a link may be followed, information may be sent to a server, etc. To make an image map accessible, content developers must ensure that each action associated with a visual region may be activated without a pointing device.

Image maps are created with the `MAP` element. HTML allows two types of image maps: client-side (the user's browser processes a URI) and server-side (the server processes click coordinates). For all image maps, content developers must supply a text equivalent.

Content developers should create client-side image maps (with "`usemap`") rather than server-side image maps (with "`ismap`") because server-side image maps require a specific input device. If server-side image maps must be used (e.g., because the geometry of a region cannot be represented with values of the `shape` attribute), authors must provide the same functionality or information in an alternative accessible format. One way to achieve this is to provide a textual link for each active region so that each link is navigable with the keyboard. If you must use a server-side image map, please consult the section on server-side image maps

4.7.5 Client-side image maps

Checkpoints in this section: 1.5, 9.1, and 10.5.

Provide text equivalents for image maps since they convey visual information.

If `AREA` is used to create the map, use the "`alt`" attribute:

Example.

```
<IMG src="welcome.gif" alt="Image map of areas in the library"
     usemap="#map1">
<MAP name="map1">
  <AREA shape="rect" coords="0,0,30,30"
        href="reference.html" alt="Reference">
  <AREA shape="rect" coords="34,34,100,100"
        href="media.html" alt="Audio visual lab">
</MAP>
```

End example.

The following example illustrates the same idea, but uses `OBJECT` instead of IMG to insert the image to provide more information about the image:

Example.

```
<OBJECT data="welcome.gif" type="image/gif" usemap="#map1">
    There are several areas in the library including
    the <A href="reference.html">Reference</A> section and the
    <A href="media.html">Audio Visual Lab</A>.
</OBJECT>
<MAP name="map1">
  <AREA shape="rect" coords="0,0,30,30"
        href="reference.html" alt="Reference">
  <AREA shape="rect" coords="34,34,100,100"
        href="media.html" alt="Audio visual lab">
</MAP>
```

End example.

In addition to providing a text equivalent, provide redundant textual links. If the A element is used instead of AREA, the content developer may describe the active regions and provide redundant links at the same time:

Example.

```
<OBJECT data="navbar1.gif" type="image/gif" usemap="#map1">
<MAP name="map1">
  <P>Navigate the site.
  [<A href="guide.html" shape="rect"
     coords="0,0,118,28">Access Guide</A>]
  [<A href="shortcut.html" shape="rect"
     coords="118,0,184,28">Go</A>]
  [<A href="search.html" shape="circle"
     coords="184.200,60">Search</A>]
  [<A href="top10.html" shape="poly"
     coords="276,0,276,28,100,200,50,50,276,0">
       Top Ten</A>]
</MAP>
</OBJECT>
```

End example.

Note that in the previous example, the MAP element is the content of the OBJECT element so that the alternative links will only be displayed if the image map (navbar1.gif) is not.

Note also that links have been separated by brackets ([]). This is to prevent older screen readers from reading several adjacent links as a single link as well as helps sighted users distinguish between links visually.

Content developers should make sure they include printable characters (such as brackets or a vertical bar (|)) surrounded by spaces between adjacent links.

4.7.6 Server-side image maps

Checkpoints in this section: 1.2.

When a server-side image map must be used, content developers should provide an alternative list of image map choices. There are three techniques:

- Include the alternative links within the body of an `OBJECT` element (refer to the previous example illustrating links in the OBJECT element).
- If `IMG` is used to insert the image, provide an alternative list of links after it and indicate the existence and location of the alternative list (e.g., via that "`alt`" attribute).

  Example.

  ```
  <A href="http://myserver.com/cgi-bin/imagemap/my-map">
  <IMG src="welcome.gif" alt="Welcome! (Text links follow)" ismap>
  </A>

 <P>[<A href="reference.html">Reference</A>]
    [<A href="media.html">Audio Visual Lab</A>]
  ```

  End example.

- If other approaches don't make the image map accessible, create an alternative page that is accessible.

Server-side and client-side image maps may be used as submit buttons in Forms. For more information, refer to the section Graphical buttons.

4.8 Applets and other programmatic objects

While applets may be included in a document with either the `APPLET` or `OBJECT` element, `OBJECT` is the preferred method.

4.8.1 Text equivalents for applets and programmatic objects

If `OBJECT` is used, provide a text equivalent in the content of the element:

Example.

```
<OBJECT classid="java:Press.class" width="500" height="500">
      As temperature increases, the molecules in the balloon...
</OBJECT>
```

End example.

A more complex example takes advantage of the fact the OBJECT elements may be embedded to provide for alternative representations of information:

Example.

```
<OBJECT classid="java:Press.class" width="500" height="500">
   <OBJECT data="Pressure.mpeg" type="video/mpeg">
      <OBJECT data="Pressure.gif" type="image/gif">
         As temperature increases, the molecules in the balloon...
      </OBJECT>
   </OBJECT>
</OBJECT>
```

End example.

If `APPLET` is used, provide a text equivalent with the "`alt`" attribute *and* in the content in the APPLET element. This enables the content to transform gracefully for those user agents that only support one of the two mechanisms ("alt" or content).

Deprecated example.

```
<APPLET code="Press.class" width="500" height="500"
        alt="Java applet: how temperature affects pressure">
      As temperature increases, the molecules in the balloon...
</APPLET>
```

4.8.2 Directly accessible applets

Checkpoints in this section: 8.1

If an applet (created with either `OBJECT` or `APPLET`) requires user interaction (e.g., the ability to manipulate a physics experiment) that cannot be duplicated in an alternative format, make the applet directly accessible.

For more information about developing accessible applets, please refer to [JAVAACCESS] and [IBMJAVA]. These companies have been developing an Accessibility API as well as making the Java Swing classes accessible.

4.8.3 Audio and Video produced by dynamic objects

Checkpoints in this section: 8.1 and 1.4.

Provide a text equivalent as for an image and auditory descriptions of visual information and captions where necessary. If an applet creates motion, developers should provide a mechanism for freezing this motion (for an example, refer to [TRACE]). Also, please refer to the next section for information about making audio and video presentations accessible.

4.9 Audio and video

4.9.1 Audio information

Auditory presentations must be accompanied by *text transcripts*, textual equivalents of auditory events. When these transcripts are presented synchronously with a video presentation they are called *captions* and are used by people who cannot hear the audio track of the video material.

Some media formats (e.g., QuickTime 3.0 and SMIL) allow captions and video descriptions to be added to the multimedia clip. SAMI allows captions to be added. The following example demonstrates that captions should include speech as well as other sounds in the environment that help viewers understand what is going on.

Example.

Captions for a scene from "E.T." The phone rings three times, then is answered.

[phone rings]

[ring]

[ring]

Hello?"

End example.

Until the format you are using supports alternative tracks, two versions of the movie could be made available, one with captions and descriptive video, and one without. Some technologies, such as SMIL and SAMI, allow separate audio/visual files to be combined with text files via a synchronization file to create captioned audio and movies.

Some technologies also allow the user to choose from multiple sets of captions to match their reading skills. For more information see the SMIL 1.0 ([SMIL]) specification.

Equivalents for sounds can be provided in the form of a text phrase on the page that links to a text transcript or description of the sound file. The link to the transcript should appear in a highly visible location such as at the top of the page. However, if a script is automatically loading a sound, it should also be able to automatically load a visual indication that the sound is currently being played and provide a description or transcript of the sound.

Note. Some controversy surrounds this technique because the browser should load the visual form of the information instead of the auditory form if the user preferences are set to do so. However, strategies must also work with today’s browsers.

For more information, please refer to [NCAM].

4.9.2 Visual information and motion

Checkpoints in this section: 1.3, and 7.3.

Auditory descriptions of the visual track provide narration of the key visual elements without interfering with the audio or dialogue of a movie. Key visual elements include actions, settings, body language, graphics, and displayed text. Auditory descriptions are used primarily by people who are blind to follow the action and other non-auditory information in video material.

Example.

Here’s an example of a collated text transcript of a clip from "The Lion King" (available at [DVS]). Note that the Describer is providing the auditory description of the video track and that the description has been integrated into the transcript.

Simba: Yeah!

Describer: Simba races outside, followed by his parents. Sarabi smiles and nudges Simba gently toward his father. The two sit side-by-side, watching the golden sunrise.

Mufasa: Look Simba, everything the light touches is our kingdom.

Simba: Wow.

End example.

Note. If there is no important visual information, for example, an animated talking head that describes (through prerecorded speech) how to use the site, then an auditory description is not necessary.

For movies, provide auditory descriptions that are synchronized with the original audio. Refer to the section on audio information for more information about multimedia formats.

4.9.3 Collated text transcripts

Collated text transcripts allow access by people with both visual and hearing disabilities. They also provide everyone with the ability to index and search for information contained in audio/visual materials.

Collated text transcripts include spoken dialogue as well as any other significant sounds including on-screen and off-screen sounds, music, laughter, applause, etc. In other words, all of the text that appears in captions as well as all of the descriptions provided in the auditory description.

4.9.4 Text equivalents for multimedia

When necessary, a text equivalent should be provided for visual information to enable understanding of the page. For example, consider a repeating animation that shows cloud cover and precipitation as part of a weather status report. Since the animation is supplementing the rest of the weather report (that is presented in natural language - text), a less verbose description of the animation is necessary. However, if the animation appears in a pedagogical setting where students are learning about cloud formations in relation to land mass, then the animation ought to be described for those who can not view the animation but who also want to learn the lesson.

See also the section on text style for controlling blinking.

4.9.5 Embedding multimedia objects

Other objects, such as those requiring a plug-in, should also use the `OBJECT` element. However, for backward compatibility with Netscape browsers, use the proprietary EMBED element within the OBJECT element as follows:

Deprecated example.

```
<OBJECT classid="clsid:A12BCD3F-GH4I-56JK-xyz"
codebase="http://site.com/content.cab" width=100 height=80>
<PARAM name="Movie" value="moviename.swf">
```

```
      <EMBED src="moviename.swf" width=100 height=80
      pluginspage="http://www.macromedia.com/shockwave/download/">
      </EMBED>

      <NOEMBED>
          <IMG alt="Still from Movie"
                   src="moviename.gif" width=100 height=80>
      </NOEMBED>

  </OBJECT>
```

End example.

For more information refer to [MACROMEDIA].

4.10 Frames

For visually enabled users, frames may organize a page into different zones. For non-visual users, relationships between the content in frames (e.g., one frame has a table of contents, another the contents themselves) must be conveyed through other means.

Frames as implemented today (with the `FRAMESET`, `FRAME`, and `IFRAME` elements) are problematic for several reasons:

- Without scripting, they tend to break the "previous page" functionality offered by browsers.
- It is impossible to refer to the "current state" of a frameset with a URI; once a frameset changes contents, the original URI no longer applies.
- Opening a frame in a new browser window can disorient or simply annoy users.

In the following sections, we discuss how to make frames more accessible. We also provide an alternative to frames that uses HTML 4.0 and CSS and addresses many of the limitations of today's frame implementations.

4.10.1 Title frames for easy orientation

Checkpoints in this section: 12.1.

Example.

Use the "title" attribute to name frames.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
<TITLE>A simple frameset document</TITLE>
</HEAD>
<FRAMESET cols="10%, 90%"
          title="Our library of electronic documents">
    <FRAME src="nav.html" title="Navigation bar">
    <FRAME src="doc.html" title="Documents">
    <NOFRAMES>
       <A href="lib.html" title="Library link">
             Select to go to the electronic library</A>
```

```
    </NOFRAMES>
</FRAMESET>
```

End example.

4.10.2 Text equivalents for frames

Checkpoints in this section: 12.2.

Example.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
  <HEAD>
    <TITLE>Today's news</TITLE>
  </HEAD>

  <FRAMESET cols="10%,*,10%">

  <FRAMESET rows="20%,*">
    <FRAME src="promo.html" name="promo" title="promotions">
    <FRAME src="sitenavbar.html" name="navbar"
       title="Sitewide navigation bar" longdesc="frameset-desc.html#navbar">
  </FRAMESET>

  <FRAME src="story.html" name="story" title="Selected story - main content"
     longdesc="frameset-desc.html#story">

  <FRAMESET rows="*,20%">
    <FRAME src="headlines.html" name="index" title="Index of other
      national headlines" longdesc="frameset-desc.html#headlines">
    <FRAME src="ad.html" name="adspace" title="Advertising">
  </FRAMESET>

  <NOFRAMES>
    <p><a href="noframes.html">No frames version</a></p>
    <p><a href="frameset-desc.html">Descriptions of frames.</a></p>
  </NOFRAMES>

  </FRAMESET>
</HTML>
```

frameset-desc.html might say something like:

```
#Navbar - this frame provides links to the major
          sections of the site:  World News, National News,
          Local News, Technological News,
          and Entertainment News.

#Story  - this frame displays the currently selected story.

#Index  - this frame provides links to the day's
          headline stories within this section.
```

End example.

Note that if the a frame's contents change, the text equivalent will no longer apply. Also, links to

descriptions of a frame should be provided along with other alternative content in the NOFRAMES element of a FRAMESET.

4.10.3 Ensure documents are readable without frames

Example.

In this example, if the user reads "top.html":

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
<TITLE>This is top.html</TITLE>
</HEAD>
<FRAMESET cols="50%, 50%" title="Our big document">
    <FRAME src="main.html" title="Where the content is displayed">
    <FRAME src="table_of_contents.html" title="Table of Contents">
    <NOFRAMES>
        <A href="table_of_contents.html">Table of Contents.</A>
     <!-- other navigational links that are available in main.html
          are available here also. -->
    </NOFRAMES>
</FRAMESET>
</HTML>
```

and the user agent is not displaying frames, the user will have access (via a link) to a non-frames version of the same information.

End example.

4.10.4 Always make the source of a frame an HTML document

Checkpoints in this section: 6.2.

Content developers must provide text equivalents of frames so that their contents and the relationships between frames make sense. Note that as the contents of a frame change, so must change any description. This is not possible if an IMG is inserted directly into a frame. Thus, content developers should always make the source ("src") of a frame an HTML file. Images may be inserted into the HTML file and their text alternatives will evolve correctly.

Example.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
<TITLE>A correct frameset document</TITLE>
</HEAD>
<FRAMESET cols="100%" title="Evolving frameset">
<FRAME name="goodframe" src="apples.html" title="Apples">
</FRAMESET>
</HTML>

   <!-- In apples.html -->
   <P><IMG src="apples.gif" alt="Apples">
```

End example.

The following deprecated example should be avoided since it inserts `IMG` directly in a frame:

Deprecated example.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
<TITLE>A bad frameset document</TITLE>
</HEAD>
<FRAMESET cols="100%" title="Static frameset">
   <FRAME name="badframe"
          src="apples.gif" title="Apples">
</FRAMESET>
</HTML>
```

Note that if, for example, a link causes a new image to be inserted into the frame:

```
  <P>Visit a beautiful grove of
  <A target="badframe" href="oranges.gif" title="Oranges">oranges</A>
```

the initial title of the frame ("Apples") will no longer match the current content of the frame ("Oranges").

4.10.5 Avoid opening a new window as the target of a frame

Checkpoints in this section: 10.1.

Content developers should avoid specifying a new window as the target of a frame with target="_blank".

4.10.6 Alternatives to frames

One of the most common uses of frames is to split the user's browser window into two parts: a navigation window and a content window. As an alternative to frames, we encourage you to try the following:

1. Create one document for the navigation mechanism (call it "nav.html"). A separate document means that the navigation mechanism may be shared by more than one document.
2. In each document requiring the navigation mechanism, include it at the bottom of the document with the following (or similar) `OBJECT` markup:

   Example.

   ```
   <P>
   <OBJECT data="nav.html">
   Go to the <A href="nav.html">table of contents</A>
   </OBJECT>
   ```

   Putting the navigation mechanism at the end of the document means that when style sheets are turned off, users have access to the document's important information first.
3. Use style sheets to position the navigation mechanism where you want on the screen. For example,

the following CSS rule floats the navigation bar to the left of the page and makes it take up 25% of the available horizontal space:

```
OBJECT { float: left; width: 25% }
```

The following CSS rule attaches the navigation mechanism to the bottom-left corner of the page of the page and keeps it there even if the user scrolls down the page:

```
OBJECT { position: fixed; left: 0; bottom: 0 }
```

Note. Navigation mechanisms or other content may be inserted in a document by means of server-side includes.

4.11 Forms

This section discusses the accessibility of forms and form controls that one can put in a `FORM` element.

4.11.1 Make controls keyboard accessible

Checkpoints in this section: 9.4 and 9.5.

Refer to the section on keyboard access for more information.

4.11.2 Group form controls

Content developers should group information where natural and appropriate. When form controls can be grouped into logical units, use the `FIELDSET` element and label those units with the `LEGEND` element:

Example.

```
<FORM action="http://somesite.com/adduser" method="post">
   <FIELDSET>
   <LEGEND>Personal information</LEGEND>
   <LABEL for="firstname">First name: </LABEL>
   <INPUT type="text" id="firstname" tabindex="1">
   <LABEL for="lastname">Last name: </LABEL>
   <INPUT type="text" id="lastname" tabindex="2">
   ...more personal information...
   </FIELDSET>
   <FIELDSET>
   <LEGEND>Medical History</LEGEND>
   ...medical history information...
   </FIELDSET>
</FORM>
```

End example.

4.11.3 Label form controls explicitly

Checkpoints in this section: 12.4 and 10.2. An example of `LABEL` used with "`for`" in HTML 4.0 is given in the previous section.

4.11.4 Group menu options

Content developers should group information where natural and appropriate. For long lists of menu selections (which may be difficult to track), content developers should group `SELECT` items (defined by `OPTION`) into a hierarchy using the `OPTGROUP` element. Specifies a label for the group of options with the `label` attribute on OPTGROUP.

Example.

```
<FORM action="http://somesite.com/prog/someprog" method="post">
 <P>
 <SELECT name="ComOS">
     <OPTGROUP label="PortMaster 3">
       <OPTION label="3.7.1" value="pm3_3.7.1">PortMaster 3 with ComOS 3.7.1
       <OPTION label="3.7" value="pm3_3.7">PortMaster 3 with ComOS 3.7
       <OPTION label="3.5" value="pm3_3.5">PortMaster 3 with ComOS 3.5
     </OPTGROUP>
     <OPTGROUP label="PortMaster 2">
       <OPTION label="3.7" value="pm2_3.7">PortMaster 2 with ComOS 3.7
       <OPTION label="3.5" value="pm2_3.5">PortMaster 2 with ComOS 3.5
     </OPTGROUP>
     <OPTGROUP label="IRX">
       <OPTION label="3.7R" value="IRX_3.7R">IRX with ComOS 3.7R
       <OPTION label="3.5R" value="IRX_3.5R">IRX with ComOS 3.5R
     </OPTGROUP>
 </SELECT>
</FORM>
```

End example.

4.11.5 Keyboard access to forms

In the next example, we specify a tabbing order among elements (in order, "field2", "field1", "submit") with "`tabindex`":

Example.

```
   <FORM action="submit" method="post">
   <P>
   <INPUT tabindex="2" type="text" name="field1">
   <INPUT tabindex="1" type="text" name="field2">
   <INPUT tabindex="3" type="submit" name="submit">
   </FORM>
```

End example.

This example assigns "U" as the accesskey (via "`accesskey`"). Typing "U" gives focus to the label, which in turn gives focus to the input control, so that the user can input text.

Example.

```
   <FORM action="submit" method="post">
   <P>
```

```
        <LABEL for="user" accesskey="U">name</LABEL>
        <INPUT type="text" id="user">
   </FORM>
```

End example.

4.11.6 Graphical buttons

Using images to decorate buttons allows developers to make their forms unique and easier to understand. Using an image for a button (e.g., with the `INPUT` element or `BUTTON`) is not inherently inaccessible - assuming a text equivalent is provided for the image.

However, a graphical form submit button created with `INPUT`, type="image" creates a type of server-side image map. Whenever the button is clicked with a mouse, the x and y coordinates of the mouse click are sent to the server as part of the form submission.

In the Image and Image Maps section, we discuss why server-side images ought to be avoided, and suggest using client-side image maps instead. In HTML 4.0, graphical buttons may now be client-side image maps. To preserve the functionality provided by the server, authors have the following options, as stated in the HTML 4.0 Recommendation ([HTML40], section 17.4.1):

> If the server takes different actions depending on the location clicked, users of non-graphical browsers will be disadvantaged.
> For this reason, authors should consider alternate approaches:
>
> - Use multiple submit buttons (each with its own image) in place of a single graphical submit button. Authors may use style sheets to control the positioning of these buttons.
> - Use a client-side image map together with scripting.

4.11.7 Techniques for specific controls

Checkpoints in this section: 10.4.

Example.

Some legacy assistive technologies require initial text in form controls such as `TEXTAREA` in order to function properly.

```
<FORM action="http://somesite.com/prog/text-read" method="post">
     <P>
     <TEXTAREA name=yourname rows="20" cols="80">
     Please enter your name here.
     </TEXTAREA>
     <INPUT type="submit" value="Send"><INPUT type="reset">
     </P>
</FORM>
```

End example.

Provide a text equivalent for images used as "submit" buttons:

Example.

```
<FORM action="http://somesite.com/prog/text-read" method="post">
<P>
<INPUT type="image" name=submit src="button.gif" alt="Submit">
</FORM>
```

End example.

Also refer to the section on keyboard access since this applies to form controls.

4.11.8 Backward compatibility issues for forms

In some HTML 3.2 browsers,

- The `BUTTON` element does not appear
- `INPUT` with type="button"> will appear as a text input field

4.12 Scripts

This section discusses the accessibility of scripts included in a document via the `SCRIPT` element.

4.12.1 Graceful transformation of scripts

Checkpoints in this section: 6.3.

Content developers must ensure that pages are accessible with scripts turned off or in browsers that don’t support scripts.

- Avoid creating content on the fly on the client. If a user’s browser does not handle scripts, no content will be generated or displayed. However, this is different than displaying or hiding already existing content by using a combination of style sheets and scripting; if there is no script, then the content is always shown. This also does not rule out generating pages on the fly on the server-side and delivering them to the client.
- Avoid creating links that use "javascript" as the URI. If a user is not using scripts, then they won’t be able to link since the browser can’t create the link content.

  Deprecated example. This is a dead-end link for a user agent where scripts are not supported or not loaded.

  ```
  <A href="javascript:">...</A>
  ```

4.12.2 Device-independent event handlers

Checkpoints in this section: 9.3 and 6.4.

An event handler is a script that is invoked when a certain event occurs (e.g, the mouse moves, a key is pressed, the document is loaded, etc.). In HTML 4.0, event handlers are attached to elements via event handler attributes (the attributes beginning with "on", as in "onkeyup").

Some event handlers, when invoked, produce purely decorative effects such as highlighting an image or changing the color of an element’s text. Other event handlers produce much more substantial effects, such as carrying out a calculation, providing important information to the user, or submitting a form. For event handlers that do more than just change the presentation of an element, content developers should do the following:

1. Use application-level event triggers rather than user interaction-level triggers. In HTML 4.0, application-level event attributes are "onfocus", "onblur" (the opposite of "onfocus"), and "onselect". Note that these attributes are designed to be device-independent, but are implemented as keyboard specific events in current browsers.
2. Otherwise, if you must use device-dependent attributes, provide redundant input mechanisms (i.e., specify two handlers for the same element):
   - Use "onmousedown" with "onkeydown".
   - Use "onmouseup" with "onkeyup"
   - Use "onclick" with "onkeypress"

   Note that there is no keyboard equivalent to double-clicking ("ondblclick") in HTML 4.0.
3. Do not write event handlers that rely on mouse coordinates since this prevents device-independent input.

4.12.3 Alternative presentation of scripts

One way to accomplish this is with the `NOSCRIPT` element. The content of this element is rendered when scripts are not enabled.

Example.

```
<SCRIPT type="text/tcl">
 ...some Tcl script to show a billboard of sports scores...
</SCRIPT>
<NOSCRIPT>
   <P>Results from yesterday’s games:</P>
   <DL>
      <DT>Bulls 91,  Sonics 80.
      <DD><A href="bullsonic.html">Bulls vs. Sonics game highlights</A>
      ...more scores...
   </DL>
</NOSCRIPT>
```

End example.

---

5 CSS Techniques

Checkpoints in this section: 3.3.

The following sections list some techniques for using CSS to design accessible documents and some techniques for writing effective style sheets. In HTML, style sheets may be specified externally via the `LINK` element, in the document head via the `STYLE` element, or for a specific element via the `style`

attribute.

CSS1 ([[CSS1]) and CSS2 ([[CSS2]) allow content developers to duplicate most HTML 4.0 presentation capabilities and offer more power with less cost. However, until most users have browsers that support style sheets, not every presentation idiom may be expressed satisfactorily with style sheets. We also provide examples of how to use HTML 4.0 features (e.g., tables, bitmap text) more accessibly when they must be used.

See also the section on text markup.

5.1 Guidelines for creating style sheets

Checkpoints in this section: 6.1 and 3.4.

Here are guidelines for creating style sheets that promote accessibility:

- Use a minimal number of style sheet for your site
- If you have more than one, use the same "class" name for the same concept in all of the style sheets.
- Use linked style sheets rather than embedded styles, and avoid inline style sheets.
- Content developers should not write "!important" rules. Users should where necessary.
- Use the "em" unit to set font sizes.
- Use relative length units and percentages. CSS allows you to use relative units even in absolute positioning. Thus, you may position an image to be offset by "3em" from the top of its containing element. This is a fixed distance, but is relative to the current font size, so it scales nicely.
- Only use absolute length units when the physical characteristics of the output medium are known.
- Always specify a fallback generic font.
- Use numbers, not names, for colors.
- Provide a text equivalent for any important image or text generated by style sheets (e.g., via the ’background-image’, ’list-style’, or ’content’ properties). **Note.** Text generated by style sheets is part of the document source and will not be available to assistive technologies that access content through DOM, level 1 ([DOM1]).
- Be sure to validate that your pages are still readable without style sheets.

Some examples follow.

Example.

Use em to set font sizes, as in:

```
H1 { font-size: 2em }
```

rather than:

```
H1 { font-size: 12pt }
```

End example.

Example.

Use relative length units and percentages.

```
BODY { margin-left: 15%; margin-right: 10%}
```

End example.

Example.

Only use absolute length units when the physical characteristics of the output medium are known.

```
.businesscard { font-size: 8pt }
```

End example.

Example.

Always specify a fallback generic font:

```
BODY { font-family: "Gill Sans", sans-serif }
```

End example.

Example.

Use numbers, not names, for colors:

```
H1 {color: #808000}
H1 {color: rgb(50%,50%,0%)}
```

End example.

5.2 Fonts

Instead of using deprecated presentation elements and attributes , use the many CSS properties to control font characteristics: ’font-family’, ’font-size’, ’font-size-adjust’, ’font-stretch’, ’font-style’, ’font-variant’, and ’font-weight’.

The following CSS2 properties can be used to control font information: ’font’, ’font-family’, ’font-size’, ’font-size-adjust’, ’font-stretch’, ’font-style’, ’font-variant’, and ’font-weight’.

Use them instead of the following deprecated font elements and attributes in HTML: `FONT`, `BASEFONT`, "face", and "size".

If you must use HTML elements to control font information, use `BIG` and `SMALL`, which are not deprecated.

Example.

```
<STYLE type="text/css">
```

```
   P.important { font-weight: bold }
   P.less-important { font-weight: lighter; font-size: smaller }
   H2.subsection { font-family: Helvetica, sans-serif }
</STYLE>
```

End example.

5.3 Text style

Checkpoints in this section: 7.2.

The following CSS2 properties can be used to style text:

- Case: ’text-transform’ (for uppercase, lowercase, and capitalization).
- Shadow effects: ’text-shadow’
- Underlines, overlinks, blinking: ’text-decoration’. **Note.** If blinking content (e.g., a headline that appears and disappears at regular intervals) is used, provide a mechanism for stopping the blinking. In CSS, ’text-decoration: blink’ will cause content to blink and will allow users to stop the effect by turning off style sheets or overriding the rule in a user style sheet. Do not use the BLINK and MARQUEE elements. These elements are not part of any W3C specification for HTML (i.e., they are non-standard elements).

5.3.1 Text instead of images

Content developers should use style sheets to style text rather than representing text in images. Using text instead of images means that the information will be available to a greater number of users (with speech synthesizers, braille displays, graphical displays, etc.). Using style sheets will also allow users to override author styles and change colors or fonts sizes more easily.

If it is necessary to use a bitmap to create a text effect (special font, transformation, shadows, etc.) the bitmap must be accessible (see the sections on text equivalents and alternative pages).

Example.

In this example, the inserted image shows the large red characters "Example", and is captured by the value of the "alt" attribute.

```
<P>This is an
   <IMG src="BigRedExample.gif" alt="example">
   of what we mean.
</P>
```

End example.

5.4 Text formatting

The following CSS2 properties can be used to control the formatting and position of text:

- Indentation: ’text-indent’. Do not use the `BLOCKQUOTE` or any other structural element to indent text.

- Letter/word spacing: ’letter-spacing’, ’word-spacing’. For example instead of writing "H E L L O" (which users generally recognize as the word "hello" but would hear as individual letters), authors may create the same visual effect with the ’word-spacing’ property applied to "HELLO". Text without spaces will be transformed more effectively to speech.
- White space: ’white-space’. This property controls the white space processing of an element’s content.
- Text direction: ’direction’, ’unicode-bidi’.
- The :first-letter and :first-line pseudo-elements allow authors to refer to the first letter or line of a paragraph of text.

The following example shows how to use style sheets to create a drop-cap effect.

Example.

```
<HEAD>
<TITLE>Drop caps</TITLE>
<STYLE type="text/css">
      .dropcap { font-size : 120%; font-family : Helvetica }
</STYLE>
</HEAD>
<BODY>
<P><SPAN class="dropcap">O</SPAN>nce upon a time...
</BODY>
```

Note. As of the writing of this document, the CSS pseudo-element ’:first-letter’, which allows content developers to refer to the first letter of a chunk of text, is not widely supported.

5.5 Colors

Checkpoints in this section: 2.1 and 2.2.

Use these CSS properties to specify colors:

- ’color’, for foreground text color.
- ’background-color’, for background colors.
- ’border-color’, ’outline-color’ for border colors.
- For link colors, refer to the :link, :visited, and :active pseudo-classes.

Ensure that information is not conveyed through color alone. For example, when asking for input from users, do not write "Please select an item from those listed in green." Instead, ensure that information is available through other style effects (e.g., a font effect) and through context (e.g,. comprehensive text links).

For instance, in this document, examples are styled by default (through style sheets) as follows:

- They are surrounded by a border.
- They use a different background color.
- They begin with the word "Example" (or "Deprecated Example".
- They also end with the phrase "End example", but that phrase is hidden by default with ’display: none’. For user agents that don’t support style sheets or when style sheets are turned off, this text

helps delineate the end of an example for readers who may not be able to see the border around the example.

Quicktest! To test whether your document still works without colors, examine it with a monochrome monitor or browser colors turned off. Also, try setting up a color scheme in your browser that only uses black, white, and the four browser-safe greys and see how your page holds up.

Quicktest! To test whether color contrast is sufficient to be read by people with color deficiencies or by those with low resolution monitors, print pages on a black and white printer (with backgrounds and colors appearing in grayscale). Also try taking the printout and copying it for two or three generations to see how it degrades. This will show you where you need to add redundant cues (example: hyperlinks are usually underlined on Web pages), or whether the cues are two small or indistinct to hold up well.

For more information about colors and contrasts, refer to [LIGHTHOUSE].

5.6 Layout, positioning, layering, and alignment

Layout, positioning, layering, and alignment should be done through style sheets (notably by using CSS floats and absolute positioning):

- ’text-indent’, ’text-align’, ’word-spacing’, ’font-stretch’. Each of these properties allows users to control spacing without adding additional spaces. Use ’text-align: center’ instead of the deprecated `CENTER` element.
- ’margin’, ’margin-top’, ’margin-right’, ’margin-bottom’, ’margin-left’. With these properties, authors can create space on four sides of an element’s content instead of adding non-breaking spaces ( ), which are non-standard mark-up, to create space around an element.
- ’float’, ’position’, ’top’, ’right’, ’bottom’, ’left’. With these properties, the user can control the visual position of almost any element in a manner independent of where the element appears in the document. Authors should always design documents that make sense without style sheets (i.e., the document should be written in a "logical" order) and then apply style sheets to achieve visual effects. The positioning properties may be used to create margin notes (which may be automatically numbered), side bars, frame-like effects, simple headers and footers, and more.
- The ’empty-cells’ property allows users to leave table cells empty and still give them proper borders on the screen or on paper. A data cell that is meant to be empty should not be filled with white space or a non-breaking space just to achieve a visual effect.

5.6.1 If you must use images as spacers

Provide text equivalents for all images, including invisible or transparent images.

If content developers cannot use style sheets and must use invisible or transparent images (e.g., with `IMG`) to lay out images on the page, they should specify `alt=""` for them.

Deprecated example.

Authors should **not** use spaces for the value of "`alt`" to prevents the words from running together when the image is not loaded:

```
my poem requires a big space<IMG src="10pttab.gif" alt="   ">here
```

In this next example, an image is used to force a graphic to appear in a certain position:

```
<IMG src="spacer.gif" alt="spacer">
<IMG src="colorfulwheel.gif" alt="The wheel of fortune">
```

End example.

5.7 Rules and borders

Rules and borders may convey the notion of "separation" to visually enabled users but that meaning cannot be inferred out of a visual context.

Use these CSS properties to specify border styles:

- ’border’, ’border-width’, ’border-style’, ’border-color’.
- ’border-spacing’ and ’border-collapse’ for tables.
- ’outline, ’outline-color’, ’outline-style’, and ’outline-width’ for dynamic outlines.

Authors should use style sheets to create rules and borders.

Example.

In this example, the `H1` element will have a top border that is 2px thick, red, and separated from the content by 1em:

```
<HEAD>
<TITLE>Redline with style sheets</TITLE>
<STYLE type="text/css">
     H1 { padding-top: 1em; border-top: 2px red }
</STYLE>
</HEAD>
<BODY>
<H1>Chapter 8 - Auditory and Tactile Displays</H1>
</BODY>
```

End example.

If a rule (e.g., the `HR` element) is used to indicate structure, be sure to indicate the structure in a non-visual way as well. (e.g., by using structural markup).

Example.

In this example, the DIV element is used to create a navigation bar, which includes a horizontal separator.

```
<DIV class="navigation-bar">
   <HR>
   <A rel="Next" href="next.html">[Next page]</A>
   <A rel="Previous" href="previous.html">[Prevous page]</A>
   <A rel="First" href="first.html">[First page]</A>
```

```
</DIV>
```

End example.

Checkpoint Map

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

Guideline 1:

1.1 Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). *This includes*: images, graphical representations of text (including symbols), image map regions, animations (e.g., animated GIFs), applets and programmatic objects, ascii art, frames, scripts, images used as list bullets, spacers, graphical buttons, sounds (played with or without user interaction), stand-alone audio files, audio tracks of video, and video. [Priority 1] (Checkpoint 1.1 in guidelines)
Refer to 3.2 Text equivalents and
4.7.1 Text equivalents for images

1.2 Provide redundant text links for each active region of a server-side image map. [Priority 1] (Checkpoint 1.2 in guidelines)
Refer to 3.2 Text equivalents and
4.7.6 Server-side image maps

1.3 Until user agents can automatically read aloud the text equivalent of a visual track, provide an auditory description of the important information of the visual track of a multimedia presentation. [Priority 1] (Checkpoint 1.3 in guidelines)
Refer to 4.9.2 Visual information and motion

1.4 For any time-based multimedia presentation (e.g., a movie or animation), synchronize equivalent alternatives (e.g., captions or auditory descriptions of the visual track) with the presentation. [Priority 1] (Checkpoint 1.4 in guidelines)
Refer to 4.8.3 Audio and Video produced by dynamic objects

1.5 Until user agents render text equivalents for client-side image map links, provide redundant text links for each active region of a client-side image map. [Priority 3] (Checkpoint 1.5 in guidelines)
Refer to 3.2 Text equivalents and
4.7.5 Client-side image maps

Guideline 2:

2.1 Ensure that all information conveyed with color is also available without color, for example from context or markup. [Priority 1] (Checkpoint 2.1 in guidelines)
Refer to 5.5 Colors

2.2 Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen. [Priority 2 for images, Priority 3 for text]. (Checkpoint 2.2 in guidelines)
Refer to 5.5 Colors

Guideline 3:

3.1 When an appropriate markup language exists, use markup rather than images to convey information. [Priority 2] (Checkpoint 3.1 in guidelines)
Refer to 4.3.4 Markup and style sheets rather than images: The example of math
3.2 Create documents that validate to published formal grammars. [Priority 2] (Checkpoint 3.2 in guidelines)
Refer to 4.1 Document structure and metadata
3.3 Use style sheets to control layout and presentation. [Priority 2] (Checkpoint 3.3 in guidelines)
Refer to 3.2 Text equivalents and
4.3.1 Emphasis and
5 CSS Techniques
3.4 Use relative rather than absolute units in markup language attribute values and style sheet property values. [Priority 2] (Checkpoint 3.4 in guidelines)
Refer to 5.1 Guidelines for creating style sheets
3.5 Use header elements to convey document structure and use them according to specification. [Priority 2] (Checkpoint 3.5 in guidelines)
Refer to 4.1.2 Section headers
3.6 Mark up lists and list items properly. [Priority 2] (Checkpoint 3.6 in guidelines)
Refer to 4.4 Lists
3.7 Mark up quotations. Do not use quotation markup for formatting effects such as indentation. [Priority 2] (Checkpoint 3.7 in guidelines)
Refer to 4.3.3 Quotations

Guideline 4:

4.1 Clearly identify changes in the natural language of a document's text and any text equivalents (e.g., captions). [Priority 1] (Checkpoint 4.1 in guidelines)
Refer to 4.2 Language information
4.2 Specify the expansion of each abbreviation or acronym in a document where it first occurs. [Priority 3] (Checkpoint 4.2 in guidelines)
Refer to 4.3.2 Acronyms and abbreviations
4.3 Identify the primary natural language of a document. [Priority 3] (Checkpoint 4.3 in guidelines)
Refer to 4.2 Language information

Guideline 5:

5.1 For data tables, identify row and column headers. [Priority 1] (Checkpoint 5.1 in guidelines)
Refer to 4.5.1 Tables of data
5.2 For data tables that have two or more logical levels of row or column headers, use markup to associate data cells and header cells. [Priority 1] (Checkpoint 5.2 in guidelines)
Refer to 4.5.1 Tables of data
5.3 Do not use tables for layout unless the table makes sense when linearized. Otherwise, if the table does not make sense, provide an alternative equivalent (which may be a linearized version). [Priority 2] (Checkpoint 5.3 in guidelines)
Refer to 4.5.2 Avoid tables for layout
5.4 If a table is used for layout, do not use any structural markup for the purpose of visual formatting. [Priority 2] (Checkpoint 5.4 in guidelines)

Refer to 4.5.2 Avoid tables for layout

5.5 Provide summaries for tables. [Priority 3] (Checkpoint 5.5 in guidelines)

Refer to 4.5.1 Tables of data

5.6 Provide abbreviations for header labels. [Priority 3] (Checkpoint 5.6 in guidelines)

Refer to 4.5.1 Tables of data

Guideline 6:

6.1 Organize documents so they may be read without style sheets. For example, when an HTML document is rendered without associated style sheets, it must still be possible to read the document. [Priority 1] (Checkpoint 6.1 in guidelines)

Refer to 5.1 Guidelines for creating style sheets

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

Refer to 4.10.4 Always make the source of a frame an HTML document

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

Refer to 4.12.1 Graceful transformation of scripts

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

Refer to 4.12.2 Device-independent event handlers

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

Refer to 3.3 Alternative pages

Guideline 7:

7.1 Until user agents allow users to control flickering, avoid causing the screen to flicker. [Priority 1] (Checkpoint 7.1 in guidelines)

Refer to 3.9 Screen flicker

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

Refer to 5.3 Text style

7.3 Until user agents allow users to freeze moving content, avoid movement in pages. [Priority 2] (Checkpoint 7.3 in guidelines)

Refer to 4.9.2 Visual information and motion

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

Refer to 3.8 Automatic page refresh

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

Refer to 3.8 Automatic page refresh

Guideline 8:

8.1 Make programmatic elements such as scripts and applets directly accessible or compatible with

assistive technologies [Priority 1 if functionality is important and not presented elsewhere, otherwise Priority 2.] (Checkpoint 8.1 in guidelines)

Refer to 4.8.2 Directly accessible applets and
4.8.3 Audio and Video produced by dynamic objects

Guideline 9:

9.1 Provide client-side image maps instead of server-side image maps except where the regions cannot be defined with an available geometric shape. [Priority 1] (Checkpoint 9.1 in guidelines)

Refer to 4.7.5 Client-side image maps

9.2 Ensure that any element that has its own interface can be operated in a device-independent manner. [Priority 2] (Checkpoint 9.2 in guidelines)

Refer to 3.4 Keyboard access

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

Refer to 4.12.2 Device-independent event handlers

9.4 Create a logical tab order through links, form controls, and objects. [Priority 3] (Checkpoint 9.4 in guidelines)

Refer to 4.11.1 Make controls keyboard accessible

9.5 Provide keyboard shortcuts to important links (including those in client-side image maps), form controls, and groups of form controls. [Priority 3] (Checkpoint 9.5 in guidelines)

Refer to 4.11.1 Make controls keyboard accessible

Guideline 10:

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

Refer to 4.10.5 Avoid opening a new window as the target of a frame

10.2 Until user agents support explicit associations between labels and form controls, for all form controls with implicitly associated labels, ensure that the label is properly positioned. [Priority 2] (Checkpoint 10.2 in guidelines)

Refer to 4.11.3 Label form controls explicitly

10.3 Until user agents (including assistive technologies) render side-by-side text correctly, provide a linear text alternative (on the current page or some other) for *all* tables that lay out text in parallel, word-wrapped columns. [Priority 3] (Checkpoint 10.3 in guidelines)

Refer to 4.5.3 Wrapped text in tables

10.4 Until user agents handle empty controls correctly, include default, place-holding characters in edit boxes and text areas. [Priority 3] (Checkpoint 10.4 in guidelines)

Refer to 4.11.7 Techniques for specific controls

10.5 Until user agents (including assistive technologies) render adjacent links distinctly, include non-link, printable characters (surrounded by spaces) between adjacent links. [Priority 3] (Checkpoint 10.5 in guidelines)

Refer to 4.7.5 Client-side image maps

Guideline 11:

11.1 Use W3C technologies when they are available and appropriate for a task and use the latest

versions when supported. [Priority 2] (Checkpoint 11.1 in guidelines)
Refer to 3.12 Browser Support
11.2 Avoid deprecated features of W3C technologies. [Priority 2] (Checkpoint 11.2 in guidelines)
Refer to
11.3 Provide information so that users may receive documents according to their preferences (e.g., language, content type, etc.) [Priority 3] (Checkpoint 11.3 in guidelines)
Refer to 3.7 Content negotiation
11.4 If, after best efforts, you cannot create an accessible page, provide a link to an alternative page that uses W3C technologies, is accessible, has equivalent information (or functionality), and is updated as often as the inaccessible (original) page. [Priority 1] (Checkpoint 11.4 in guidelines)
Refer to 3.3 Alternative pages

Guideline 12:

12.1 Title each frame to facilitate frame identification and navigation. [Priority 1] (Checkpoint 12.1 in guidelines)
Refer to 3.2 Text equivalents and
4.10.1 Title frames for easy orientation
12.2 Describe the purpose of frames and how frames relate to each other if it is not obvious by frame titles alone. [Priority 2] (Checkpoint 12.2 in guidelines)
Refer to 3.2 Text equivalents and
4.10.2 Text equivalents for frames
12.3 Divide large blocks of information into more manageable groups where natural and appropriate. [Priority 2] (Checkpoint 12.3 in guidelines)
Refer to 4.1.4 Structural grouping
12.4 Associate labels explicitly with their controls. [Priority 2] (Checkpoint 12.4 in guidelines)
Refer to 4.11.3 Label form controls explicitly

Guideline 13:

13.1 Clearly identify the target of each link. [Priority 2] (Checkpoint 13.1 in guidelines)
Refer to 4.6 Links
13.2 Provide metadata to add semantic information to pages and sites. [Priority 2] (Checkpoint 13.2 in guidelines)
Refer to 3.5 Navigation and
4.1.1 Metadata and
4.1.3 Link metadata and navigation tools
13.3 Provide information about the general layout of a site (e.g., a site map or table of contents). [Priority 2] (Checkpoint 13.3 in guidelines)
Refer to 3.5 Navigation
13.4 Use navigation mechanisms in a consistent manner. [Priority 2] (Checkpoint 13.4 in guidelines)
Refer to 3.5 Navigation
13.5 Provide navigation bars to highlight and give access to the navigation mechanism. [Priority 3] (Checkpoint 13.5 in guidelines)
Refer to 3.5 Navigation
13.6 Group related links, identify the group (for user agents), and, until user agents do so, provide a way to bypass the group. [Priority 3] (Checkpoint 13.6 in guidelines)
Refer to 4.6 Links

13.7 If search functions are provided, enable different types of searches for different skill levels and preferences. [Priority 3] (Checkpoint 13.7 in guidelines)
Refer to 3.5 Navigation
13.8 Place distinguishing information at the beginning of headings, paragraphs, lists, etc. [Priority 3] (Checkpoint 13.8 in guidelines)
Refer to 3.6 Comprehension
13.9 Provide information about document collections (i.e., documents comprising multiple pages.). [Priority 3] (Checkpoint 13.9 in guidelines)
Refer to 3.10 Bundled documents
13.10 Provide a means to skip over multi-line ASCII art. [Priority 3] (Checkpoint 13.10 in guidelines)
Refer to 3.2 Text equivalents and
4.7.3 Ascii art

Guideline 14:

14.1 Use the clearest and simplest language appropriate for a site's content. [Priority 1] (Checkpoint 14.1 in guidelines)
Refer to 3.6 Comprehension
14.2 Supplement text with graphic or auditory presentations where they will facilitate comprehension of the page. [Priority 3] (Checkpoint 14.2 in guidelines)
Refer to 3.6 Comprehension
14.3 Create a style of presentation that is consistent across pages. [Priority 3] (Checkpoint 14.3 in guidelines)
Refer to 3.5 Navigation

Index of HTML elements and attributes

Checkpoints in this section: 11.2.

Elements

Linear version of HTML 4.0 element index.

This index lists all elements in HTML 4.0. The first column of this table links to the definition of the element in the HTML 4.0 specification ([HTML40]). Elements that are deprecated in HTML 4.0 are followed by an asterisk (*). Elements that are obsolete in HTML 4.0 or don't exist in a W3C specification of HTML (2.0, 3.2, 4.0) do not appear in this table.

The second column indicates other W3C specifications for HTML that included each element. The third column indicates the element's role.

The last column lists the sections in the current document where the element is discussed. An entry of "N/A" means that the element is not discussed in this document.

Element name	Defined also in	Role	Techniques

A	2.0, 3.2	Structure	4.6 Links, 4.7.2 Invisible d-links, 4.7.5 Client-side image maps
ABBR		Structure	4.3.2 Acronyms and abbreviations, 4.7.3 Ascii art
ACRONYM		Structure	4.3.2 Acronyms and abbreviations
ADDRESS	2.0, 3.2	Metadata	4.1.1 Metadata
APPLET*	3.2	Replaced	4.8 Applets and other programmatic objects, 4.8.1 Text equivalents for applets and programmatic objects, 4.8.2 Directly accessible applets
AREA	3.2	Structure	4.7.5 Client-side image maps
B	2.0, 3.2	Presentation	4.3.1 Emphasis
BASE	2.0, 3.2	Processing	N/A
BASEFONT*	3.2	Presentation	5.2 Fonts
BDO		Processing	N/A
BIG	3.2	Presentation	5.2 Fonts
BLOCKQUOTE	2.0, 3.2	Structure	3.1 Structure vs. Presentation, 4.3.3 Quotations, 5.4 Text formatting
BODY	2.0, 3.2	Structure	N/A
BR	2.0, 3.2	Presentation	N/A
BUTTON		Structure	4.11.6 Graphical buttons, 4.11.8 Backward compatibility issues for forms
CAPTION	3.2	Structure	4.1.4 Structural grouping, 4.5.1 Tables of data
CENTER*	3.2	Presentation	5.6 Layout, positioning, layering, and alignment
CITE	2.0, 3.2	Structure	4.3.5 Miscellaneous structural markup
CODE	2.0, 3.2	Structure	4.3.5 Miscellaneous structural markup
COL		Structure	4.5.1 Tables of data
COLGROUP		Structure	4.1.4 Structural grouping, 4.5.1 Tables of data
DD	2.0, 3.2	Structure	4.4.1 Use style sheets to change list bullets
DEL		Metadata	4.3.5 Miscellaneous structural markup
DFN	3.2	Structure	4.3.5 Miscellaneous structural markup
DIR*	2.0, 3.2	Structure	N/A
DIV	3.2	Structure	4.6.1 Grouping and bypassing links
DL	2.0, 3.2	Structure	4.1.4 Structural grouping, 4.4 Lists, 4.4.1 Use style sheets to change list bullets
DT	2.0, 3.2	Structure	4.4.1 Use style sheets to change list bullets
EM	2.0, 3.2	Structure	4.3.1 Emphasis
FIELDSET		Structure	4.1.4 Structural grouping, 4.11.2 Group form controls
FONT*	3.2	Presentation	5.2 Fonts
FORM	2.0, 3.2	Structure	4.11 Forms

FRAME		Replaced	4.6.1 Grouping and bypassing links, 4.10 Frames
FRAMESET		Presentation	4.10 Frames, 4.10.2 Text equivalents for frames
H1	2.0, 3.2	Structure	3.1 Structure vs. Presentation, 4.1.2 Section headers, 4.1.4 Structural grouping
HEAD	2.0, 3.2	Structure	N/A
HR	2.0, 3.2	Presentation	3.1 Structure vs. Presentation, 4.1.2 Section headers
HTML	2.0, 3.2	Structure	N/A
I	2.0, 3.2	Presentation	4.3.1 Emphasis
IFRAME		Replaced	4.10 Frames
IMG	2.0, 3.2	Replaced	4.7.1 Text equivalents for images, 4.7.6 Server-side image maps, 4.10.4 Always make the source of a frame an HTML document, 5.6.1 If you must use images as spacers
INPUT	2.0, 3.2	Structure	4.11.6 Graphical buttons, 4.11.8 Backward compatibility issues for forms
INS		Metadata	4.3.5 Miscellaneous structural markup
ISINDEX*	2.0, 3.2	Structure	N/A
KBD	2.0, 3.2	Structure	4.3.5 Miscellaneous structural markup
LABEL		Structure	4.11.3 Label form controls explicitly
LEGEND		Structure	4.1.4 Structural grouping, 4.11.2 Group form controls
LI	2.0, 3.2	Structure	4.4.1 Use style sheets to change list bullets
LINK	2.0, 3.2	Metadata	3.3 Alternative pages, 4.1.1 Metadata, 4.1.3 Link metadata and navigation tools, 5 CSS Techniques
MAP	3.2	Structure	4.7.4 Image maps
MENU*	2.0, 3.2	Structure	N/A
META	2.0, 3.2	Metadata	3.8 Automatic page refresh, 4.1.1 Metadata
NOFRAMES		Alternative	4.10.2 Text equivalents for frames
NOSCRIPT		Alternative	4.12.3 Alternative presentation of scripts
OBJECT		Replaced	3.2.1 Overview of technologies, 4.7.1 Text equivalents for images, 4.7.5 Client-side image maps, 4.7.6 Server-side image maps, 4.8 Applets and other programmatic objects, 4.8.1 Text equivalents for applets and programmatic objects, 4.8.2 Directly accessible applets, 4.9.5 Embedding multimedia objects, 4.10.6 Alternatives to frames
OL	2.0, 3.2	Structure	4.1.4 Structural grouping, 4.4 Lists
OPTGROUP		Structure	4.1.4 Structural grouping, 4.11.4 Group menu options
OPTION	2.0, 3.2	Structure	4.11.4 Group menu options
P	2.0, 3.2	Structure	4.1.4 Structural grouping, 4.6.1 Grouping and bypassing links

PARAM	3.2	Processing	N/A
PRE	2.0, 3.2	Presentation	4.5.1 Tables of data
Q		Structure	4.3.3 Quotations
S*		Presentation	N/A
SAMP	2.0, 3.2	Structure	4.3.5 Miscellaneous structural markup
SCRIPT	3.2 (DTD)	Processing	4.12 Scripts
SELECT	2.0, 3.2	Structure	4.11.4 Group menu options
SMALL	3.2	Presentation	5.2 Fonts
SPAN		Structure	4.6.1 Grouping and bypassing links
STRIKE*	3.2	Presentation	N/A
STRONG	2.0, 3.2	Structure	4.3.1 Emphasis
STYLE	3.2 (DTD)	Processing	5 CSS Techniques
SUB	3.2	Presentation	N/A
SUP	3.2	Presentation	N/A
TABLE	3.2	Structure	4.5 Tables
TBODY		Structure	4.1.4 Structural grouping, 4.5.1 Tables of data
TD	3.2	Structure	4.5.1 Tables of data
TEXTAREA	2.0, 3.2	Structure	4.11.7 Techniques for specific controls
TFOOT		Structure	4.1.4 Structural grouping, 4.5.1 Tables of data, 4.5.4 Backward compatibility issues for tables
TH	3.2	Structure	4.5.1 Tables of data
THEAD		Structure	4.1.4 Structural grouping, 4.5.1 Tables of data
TITLE	2.0, 3.2	Metadata	4.1.1 Metadata
TR	3.2	Structure	N/A
TT	2.0, 3.2	Presentation	N/A
U*	3.2	Presentation	N/A
UL	2.0, 3.2	Structure	4.1.4 Structural grouping, 4.4 Lists
VAR	2.0, 3.2	Structure	4.3.5 Miscellaneous structural markup

Attributes

Linear version of HTML 4.0 attribute index.

This index lists some attributes in HTML 4.0 that affect accessibility and what elements they apply to. The first column of this table links to the definition of the attribute in the HTML 4.0 specification ([HTML40]). Attributes and elements that are deprecated in HTML 4.0 ([HTML40]) are followed by an asterisk (*). Attributes and elements that are obsolete in HTML 4.0 or don’t exist in a W3C specification of HTML (2.0, 3.2, 4.0) do not appear in this table. Attributes that apply to most elements of HTML 4.0 are indicated as such; please consult the HTML 4.0 specification for the exact list of elements with this

attribute.

The second column indicates other W3C specifications for HTML that included each attribute. The third column indicates the elements that take each attribute. The fourth column indicates the attribute's role.

The last column lists the sections in the current document where the attribute is discussed. An entry of "N/A" means that the attribute is not discussed in this document.

Attribute name	**Applies to elements**	**Role**	**Techniques**
abbr	TD, TH	Alternative	4.5.1 Tables of data
accesskey	A, AREA, BUTTON, INPUT, LABEL, LEGEND, TEXTAREA	User Interface	4.11.5 Keyboard access to forms
alt	APPLET, AREA, IMG, INPUT	Alternative	3.2.1 Overview of technologies, 4.7.1 Text equivalents for images, 4.7.2 Invisible d-links, 4.7.5 Client-side image maps, 4.7.6 Server-side image maps, 4.8.1 Text equivalents for applets and programmatic objects, 5.6.1 If you must use images as spacers
axis	TD, TH	Structure	4.5.1 Tables of data
class	Most elements	Structure	4.6.1 Grouping and bypassing links
dir	Most elements	Processing	4.5.1 Tables of data
for	LABEL	Structure	4.11.3 Label form controls explicitly
headers	TD, TH	Structure	4.5.1 Tables of data, 4.5.1 Tables of data
hreflang	A, LINK	Metadata	3.7 Content negotiation
id	Most elements	Structure	4.6.1 Grouping and bypassing links
label	OPTION	Alternative	4.11.4 Group menu options
lang	Most elements	Metadata	4.2 Language information
longdesc	IMG, FRAME, IFRAME	Alternative	3.2.1 Overview of technologies, 4.7.1 Text equivalents for images
scope	TD, TH	Structure	4.5.1 Tables of data
style	Most elements	Processing	5 CSS Techniques
summary	TABLE	Alternative	4.5.1 Tables of data
tabindex	A, AREA, BUTTON, INPUT, OBJECT, SELECT, TEXTAREA	User Interface	4.6.1 Grouping and bypassing links, 4.6.2 Keyboard access, 4.11.5 Keyboard access to forms
title	Most elements	Metadata	3.2.2 Backward Compatibility, 4.1.1 Metadata, 4.3.2 Acronyms and abbreviations, 4.6 Links, 4.7.3 Ascii art
usemap	IMG, INPUT, OBJECT	Processing	4.7.4 Image maps

The following is the list of HTML 4.0 attributes not directly related to accessibility. Content developers should use style sheets instead of presentation attributes. For even handler attributes, please refer to the section on device-independent event handlers for more detail.

Other structural attributes:
start*, value*, rowspan, colspan, span
Other presentation attributes:
align*, valign*, clear*, nowrap*, char, charoff, hspace*, vspace*, cellpadding, cellspacing, compact*, face*, size*, background*, bgcolor*, color*, text*, link*, alink*, vlink*, border, noshade*, rules, size (deprecated according to element), marginheight, marginwidth, frame, frameborder, rows, cols
Other processing instruction attributes:
ismap, coords, shape
Other user interface attributes:
target, scrolling, noresize
Other metadata attributes:
type, cite, datetime
Event handler attributes:
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onload, onload, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onreset, onselect, onsubmit, onunload

---

Acknowledgments

Web Content Guidelines Working Group Co-Chairs:
Chuck Letourneau, Starling Access Services
Gregg Vanderheiden, Trace Research and Development
W3C Team contacts:
Judy Brewer and Daniel Dardailler
We wish to thank the following people who have contributed their time and valuable comments to shaping these guidelines:
Harvey Bingham, Kevin Carey, Chetz Colwell, Neal Ewers, Geoff Freed, Al Gilman, Larry Goldberg, Jon Gunderson, Eric Hansen, Phill Jenkins, Leonard Kasday, George Kerscher, Marja-Riitta Koivunen, Josh Krieger, Scott Luebking, William Loughborough, Murray Maloney, Charles McCathieNevile, MegaZone (Livingston Enterprises), Masafumi Nakane, Mark Novak, Charles Oppermann, Mike Paciello, David Pawson, Michael Pieper, Greg Rosmaita, Liam Quinn, Dave Raggett, T.V. Raman, Robert Savellis, Jutta Treviranus, Steve Tyler, Jaap van Lelieveld, and Jason White

The original draft of this document is based on "The Unified Web Site Accessibility Guidelines" ([UWSAG]) compiled by the Trace R & D Center at the University of Wisconsin. That document includes a list of additional contributors.

Reference specifications

For the latest version of any W3C specification please consult the list of W3C Technical Reports.

[CSS1]
"CSS, level 1 Recommendation", B. Bos, H. Wium Lie, eds., 17 December 1996, revised 11 January 1999. The CSS1 Recommendation is: http://www.w3.org/TR/1999/REC-CSS1-19990111.
The latest version of CSS1 is available at: http://www.w3.org/TR/REC-CSS1.

[CSS2]
"CSS, level 2 Recommendation", B. Bos, H. Wium Lie, C. Lilley, and I. Jacobs, eds., 12 May 1998. The CSS2 Recommendation is: http://www.w3.org/TR/1998/REC-CSS2-19980512.
The latest version of CSS2 is available at: http://www.w3.org/TR/REC-CSS2.

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

[HTML40]
"HTML 4.0 Recommendation", D. Raggett, A. Le Hors, and I. Jacobs, eds., 17 December 1997, revised 24 April 1998. The HTML 4.0 Recommendation is:
http://www.w3.org/TR/1998/REC-html40-19980424.
The latest version of HTML 4.0 is available at: http://www.w3.org/TR/REC-html40.

[HTML32]
"HTML 3.2 Recommendation", D. Raggett, ed., 14 January 1997. The latest version of HTML 3.2 is available at: http://www.w3.org/TR/REC-html32.

[MATHML]
"Mathematical Markup Language", P. Ion and R. Miner, eds., 7 April 1998. The MathML 1.0 Recommendation is: http://www.w3.org/TR/1998/REC-MathML-19980407.
The latest version of MathML 1.0 is available at: http://www.w3.org/TRREC-MathML.

[PNG]
"PNG (Portable Network Graphics) Specification", T. Boutell, ed., T. Lane, contributing ed., 1 October 1996. The latest version of PNG 1.0 is: http://www.w3.org/TR/REC-png.

[RDF]
"Resource Description Framework (RDF) Model and Syntax Specification", O. Lassila, R. Swick, eds., 22 February 1999. The RDF Recommendation is:
http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.
The latest version of RDF 1.0 is available at: http://www.w3.org/TR/REC-rdf-syntax

[RFC2068]
"HTTP Version 1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, and T. Berners-Lee, January 1997.

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

[TECHNIQUES]
"Techniques for Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, I. Jacobs, eds. This document explains how to implement the checkpoints defined in "Web Content Accessibility Guidelines 1.0". The latest draft of the techniques is available at:

http://www.w3.org/TR/WAI-WEBCONTENT-TECHS/

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

[WAI-UA-SUPPORT]
This page documents known support by user agents (including assistive technologies) of some accessibility features listed in this document. The page is available at: http://www.w3.org/WAI/Resources/WAI-UA-Support

[WAI-USERAGENT]
"User Agent Accessibility Guidelines", J. Gunderson and I. Jacobs, eds. The latest Working Draft of these guidelines for designing accessible user agents is available at: http://www.w3.org/TR/WAI-USERAGENT/

[WCAG-ICONS]
Information about conformance icons for this document and how to use them is available at http://www.w3.org/WAI/WCAG1-Conformance.html

[UWSAG]
"The Unified Web Site Accessibility Guidelines", G. Vanderheiden, W. Chisholm, eds. The Unified Web Site Guidelines were compiled by the Trace R & D Center at the University of Wisconsin under funding from the National Institute on Disability and Rehabilitation Research (NIDRR), U.S. Dept. of Education. This document is available at: http://www.tracecenter.org/docs/html_guidelines/version8.htm

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

Services

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

[ASTER]
For information about ASTER, an "Audio System For Technical Readings", consult T. V. Raman’s home page.

[BOBBY]
Bobby is an automatic accessibility validation tool developed by Cast.

[BROWSERCAPS]
BrowserCaps.

[CSSVAL]
The W3C CSS Validation Service.

[DVS]
DVS Descriptive Video Services.

[HACKER]
Hacker, Diana. (1993). A Pocket Style Manual. St. Martin’s Press, Inc. 175 Fifth Avenue, New York, NY 10010.

[HOMEPAGEREADER]
IBM's Home Page Reader.
[HTMLVAL]
The W3C HTML Validation Service.
[HYPERMEDIA]
IBM's techexplorer Hypermedia Browser.
[IBMJAVA]
IBM Guidelines for Writing Accessible Applications Using 100% Pure Java are available from IBM Special Needs Systems.
[JAVAACCESS]
Information about Java Accessibility and Usability is available from the Trace R&D Center.
[JAWS]
Henter-Joyce's Jaws screen reader.
[LIGHTHOUSE]
The Lighthouse provides information about accessible colors and contrasts.
[LYNX]
Lynx is a text-only browser.
[LYNXME]
Lynx-me is a Lynx emulator.
[LYNXVIEW]
Lynx Viewer is a Lynx emulator.
[MACROMEDIA]
Flash OBJECT and EMBED Tag Syntax from Macromedia.
[NCAM]
The National Center for Accessible Media includes information about captioning and audio description on the Web.
[PWWEBSPEAK]
The Productivity Works' pwWebSpeak.
[SPOOL]
Spool, J.M., Sconlong, T., Schroeder, W., Snyder, C., DeAngelo, T. (1997). Web Site Usability: A Designer's Guide User Interface Engineering, 800 Turnpike St, Suite 101, North Andover, MA 01845.
[TECHHEAD]
Tech Head provides some information about the Fog index described in [SPOOL].
[TRACE]
The Trace Research & Development Center. Consult this site for a variet of information about accessibility, including a scrolling Java applet that may be frozen by the user.
[WAI-ER]
The WAI Evaluation and Repair Working Group
[WALSH]
Walsh, Norman. (1997). "A Guide to XML." In "XML: Principles, Tools, and Techniques." Dan Connolly, Ed. O'Reilly & Associates, 101 Morris St, Sebastopol, CA 95472. pp 97-107.
[WEBREVIEW]
webreview.com style sheet browser compatibility charts.
[WINVISION]
Artic's WinVision.