Professional Web Accessibility Auditing Made Easy

10 Key Guidelines

The Web Content Accessibility Guidelines (WCAG 2.0) include 61 recommendations, referred to as guidelines. Not all of these guidelines are applicable to all websites, though there are some that are more frequently relevant than others. In this section the most relevant guidelines are introduced.

While these guidelines should be foremost in your mind when auditing Web accessibility, they should not be used to the exclusion of the others. All 61 WCAG 2.0 guidelines should be considered.

1. Non-Text Content (Guideline 1.1.1 Level A)

This guideline is perhaps the most important of all the guidelines in WCAG 2.0 in terms of reducing potential barriers. It is relevant to making content accessible to those who are blind. Generally speaking, it requires that all meaningful visual content include an equivalent text alternative. The guideline once again, is as follows:

1.1.1 Non-Text Content: All non-text content (Links to an external site.) that is presented to the user has a text alternative (Links to an external site.) that serves the equivalent purpose, except for the situations listed below. (Level A)

- **Controls, Input:** If non-text content is a control or accepts user input, then it has a name (Links to an external site.) that describes its purpose. (Refer to Guideline 4.1 (Links to an external site.) for additional requirements for controls and content that accepts user input.)
- **Time-Based Media:** If non-text content is time-based media, then text alternatives at least provide descriptive identification of the non-text content. (Refer to Guideline 1.2 (Links to an external site.) for additional requirements for media.)
- **Test:** If non-text content is a test or exercise that would be invalid if presented in text (Links to an external site.), then text alternatives at least provide descriptive identification of the non-text content.
- **Sensory:** If non-text content is primarily intended to create a specific sensory experience (Links to an external site.), then text alternatives at least provide descriptive identification of the non-text content.
- **CAPTCHA** (Links to an external site.)**:** If the purpose of non-text content is to confirm that content is being accessed by a person rather than a computer, then text alternatives that identify and describe the purpose of the non-text content are provided, and alternative forms of CAPTCHA using output modes for different types of sensory perception are provided to accommodate different disabilities.
- **Decoration, Formatting, Invisible:** If non-text content is pure decoration (Links to an external site.), is used only for visual formatting, or is not presented to users, then it is implemented in a way that it can be ignored by assistive technology (Links to an external site.).

Guideline 1.1.1 Explained

The best way to make photographs, images, and other non-text content accessible to those who are blind is to provide text alternatives.

Text alternatives are easily "translated" into forms that individuals, whatever their needs, can understand. A person who is blind can read the text with a screen reader or convert the text to Braille. A person reading in a second language can translate the text. A person with low vision can magnify the text without it losing its crisp appearance. The text can be indexed by search engines to make images searchable, and so on.

Text alternatives can take on a variety of forms:

Alt Text is an attribute associated with the HTML <img> element. It should contain a short description of the meaningful elements of the image, up to a maximum of 125 characters. All images must have an alt attribute to conform with this guideline, even if an image is decorative and contains no useful information.

For meaningless images the value for alt is left empty (e.g., alt=””). This forces assistive technologies to ignore the image. If the alt attribute is missing, assistive technologies will announce the file name for the image, which will often interfere with comprehension of the relevant surrounding content.

Technical:

Example of Alt text describing a character in a book:

Example of empty Alt text used for a corner image in a user interface:

Key Point: All images must have an alt attribute, regardless of whether they are meaningful or not. Images missing alt will fail Guideline 1.1.1.

Key Point: Alt text must be no more than 125 characters in length. Assistive technologies will stop reading at this point, and move on to the content that follows.

The only times the alt attribute should be left blank is when the non-text content is:

1. Purely decorative
2. Used only for visual formatting
3. Not presented to users
4. A test or exercise where the alt text would affect the outcome

It is acceptable to leave out text alternatives that "give away" answers. For example, in an online examination to identify artists based on reproductions of their paintings, "Pablo Picasso" should ***not*** be used as the text alternative for his painting. (But it would be appropriate to describe the paintings follows: "An old man with a white beard holding a guitar. Painted mostly in shades of blue.")

2. Time-Based Media (Guideline 1.2.2 Level A)

Time-based media refers to audio content or video content or audio and video content combined. Guideline 1.2.2 is relevant to combined media, and making the audio tracks within that media accessible to those who are Deaf.

1.2.2 Captions (Prerecorded): Captions (Links to an external site.) are provided for all prerecorded (Links to an external site.) audio (Links to an external site.) content in synchronized media (Links to an external site.), except when the media is a media alternative for text (Links to an external site.) and is clearly labeled as such. (Level A)

Guideline 1.2.2 Explained

When video has meaningful audio content, particularly spoken content, that content must be accompanied by synchronized captions to conform with this guideline.

Most people are familiar with Closed Captions. These are captions contained in a text file with time intervals, along with the text to display during each interval. Providing closed captions for television is the law in many jurisdictions. There are also many places that now require captions on Internet-based video, part of Web accessibility legislation based on this particular WCAG 2.0 guideline. In Ontario for instance, it became law for larger public organizations on January 1, 2014. Educational institutions must caption all of the Web-based videos they produce.

There are also Open Captions. Unlike closed captions that can be turned on and off and are available in a separate text track in a video, open captions are burned right into the video, and cannot be turned off or otherwise separated from the video. Either type of captions will meet the success criteria for this guideline, though closed captions are a better choice where possible because they are more adaptable. Closed captions can easily be converted into a transcript or translated and swapped with a different language to create subtitles. Descriptive text describing meaningful visual elements in a video can be added to captions to be read by screen readers and take the place of audio description (see Guideline 1.2.3).

Note that in Ontario the Level A requirement of WCAG 2.0 to provide audio description with video has been removed from the AODA requirements, though audio description still remains an important accommodation for those who are blind. Providing descriptive text in captions to describe the meaningful visual elements of a video can be a good alternative to providing audio descriptions.

Organizations may choose to use a third-party video captioning service or to do the captioning themselves using video editing software captioning features or free software such as Amara (Links to an external site.).

Key Point: Captions are required by law in many jurisdictions. In Ontario the AODA makes it mandatory to include captions with prerecorded video.

3. Info and Relationships (Gudieline 1.3.1 Level A)

This guideline refers to the structuring of content in meaningful ways, and to explicit association of related content. Also see Guideline 3.3.2.)

1.3.1 Info and Relationships: Information, structure (Links to an external site.), and relationships (Links to an external site.) conveyed through presentation (Links to an external site.) can be programmatically determined (Links to an external site.) or are available in text. (Level A)

Structuring of content is typically accomplished by using HTML heading markup (h1, h2, etc.) to create topics and subtopics, though there are a variety of other ways to structure and express the relationships between elements on a webpage, which will be outlined below.

Guideline 1.3.1 Explained

You were introduced to heading markup when guideline 2.4.1 was discussed; remember that it is another way for assistive technology users to navigate by keyboard. Headings are used to represent the structure of topics and subtopics, to assist in organizing the content in a meaningful way to aid understanding and comprehension.

Headings

When a screen reader user lists the headings on a page, it provides information about the topics on the page and related subtopics for each. A screen reader will announce the text of the heading as well as its heading level (1 to 6). This can provide a quick overview of the page, and create relationships between topics and subtopics, before the user decides how to proceed through the content.

If headings are not provided, or they are used incorrectly, understanding the content becomes more difficult. When headings are used they should be nested correctly. For example, an H1 should be followed by an H2, or another H1. An H2 should be followed by an H3, or an H2 or H1. An H3 should be followed by an H4, or an H3, H2, or H1, and so on. Heading levels should not be skipped when traversing down through a heading list (e.g., H2 should not be followed by H4).

Key Point: Properly nested headings, using HTML heading markup, should be used to structure topics and subtopics within a webpage, rather than using otherwise styled large bold text.

Lists

Another way to structure content to aid understanding is to arrange sequences or collections of related items in lists, using HTML list markup (i.e., <ul>, <ol>, <li>, etc.). When a screen reader encounters a list, it will announce the presence of the list and the number of items it contains. While navigating through the list, the cursor's position in the list will be announced (e.g., "item 4 of 7"). Having this structural information can aid comprehension when one cannot see the visual structure of the list.

Avoid using split lists, in which items are numbered throughout a page, separated by other content, with each item a new list starting at a different number. Though these may look like a list when viewed, when read by a screen reader, each will be announced as a new list with only one item, rather than as part of a larger list.

When creating lists, use an ordered list (<ol>) only when the order in which list items appear is important. Otherwise use an unordered list (<ul>).

Key Point: If a collection of items on a page looks like a list, ensure that list markup has been used to create the list, rather than using simple line breaks after each item.

Form Labels

Relationships between form labels and their respective input fields should be created using the HTML <label> element and explicitly associating the two by matching the "id" attribute value of the input field, with the "for" attribute value of the <label> element.

Technical: Explicitly associate form labels with their respective form input fields by matching the for and id attributes, respectively, as follows.

<label for="username">Login Name</label>

<input type="text" id="username" />

When form elements are explicitly labelled in this way, it ensures that regardless of where the label appears on the screen, which may change if a screen is magnified or is being viewed on a smaller device, when a screen reader encounters the input field, it will be announced label properly.

Another advantage of using explicit labels is that the labels become clickable to send focus into the input field. This can be helpful for those who may have trouble targeting a small input field (e.g., a radio button or a checkbox) with a mouse pointer, giving them a larger target area to click to activate the form element.

Related form fields can also be grouped using a <fieldset> element, and labelled with the <legend> element. For example, in a registration form, all account information can be grouped, personal information can be grouped, and perhaps address information can be grouped. In each case, the <legend> element gets announced by AT along with the form element's label, clarifying the meaning and relationship between elements within the group.

Table Headers

When navigating through a table in Web content that lays out a series of columns and rows to arrange data or a collection of related information, it is important that the first row, and potentially the first column, are created using the HTML table header element (<th>). When navigating through the data cells (<td>) with a screen reader, the header cells can be announced along with the content of the data cell. This can be particularly important for larger tables where it becomes increasingly more difficult to keep track of one's location within the grid of cells.

Technical: Ensure that data tables use table header elements at the top of each column, and in some cases, at the start of each row.

In this simplistic example, a table has its first row and column marked up as header cells:

```
<table>
<tr>
  <td></td>
  <th>Email</th>
  <th>Address </th>
  <th>Phone</th>
</tr>
<tr>
  <th>John Smith</th>
  <td>jsmith@email.com</td>
  <td>110 Someplace St., Toronto, ON Canada</td>
  <td>416 647 5000</td>
</tr>
…
</table>
```

With header cells a screen reader will announce the data cells as follows:

John Smith, email, jsmith@email.com

John Smith, address, 110 Someplace Street, Toronto, ON Canada

John Smith, phone, 416 647 5000

Without header cells a screen reader will announce the data cells as follows:

jsmith@email.com

110 Someplace Street, Toronto, ON Canada

416 647 5000

Tables used to lay out non-data content should be avoided. Though in many cases layout table can be accessible, using elements to lay out content is a much more adaptable strategy. They will accommodate a wider range of technologies (e.g., those with small screens) and adapt more easily to individual user needs. Elements wrap when the screen is magnified, for example, and stay in view rather than getting pushed off the side of the screen.

There are many ways to express information about relationships in Web content; those listed here are the main ones. Refer to the How to meet 1.3.1 (Links to an external site.) for a list of other potential techniques.

4. Meaningful Sequence (Guideline 1.3.2 Level A)

There are times when what appears to be a logical path or sequence in Web content when viewed, becomes illogical when read by assistive technologies navigating through the content with a keyboard. The logical sequence for most Web content should move from top left to right, from the top of the page to the bottom.

1.3.2 Meaningful Sequence: When the sequence in which content is presented affects its meaning, a correct reading sequence (Links to an external site.) can be programmatically determined (Links to an external site.). (Level A)

Guideline 1.3.2 Explained

A common example of an illogical sequence is a form laid out in a two column table, with the labels in the left column, and the form input fields in the right column. When a screen reader encounters the form, it will first read all the labels in the left column, then it will read all the input fields in the right column. The logical or meaningful sequence in this case would be to read the first label then the first input field, followed by the second label then the second input field, and so on. To correct the illogical sequence in this case each label and form field could be given its own row in the table. Even better, to avoid using tables for layout, use a definition list (<dl>), in which the label is a definition term (<dt>) and the form field is the definition description (<dd>).

Another common example of an illogical sequence occurs when elements are reordered using CSS. Visually the content may appear to flow from left to right, but when navigating with a keyboard, the cursor may jump around the screen. For those with low vision, who may have the screen resolution magnified several times to make the screen more visible, content often appears out of view off to the side of the screen. If they are using the Tab key to navigate through links and forms, they will expect the above sequence and go looking for the cursor's focus off to the right, or back to the left after reaching the far right of the screen. If the cursor does not follow the usual sequence, the user can have considerable difficulty finding it when it leaves the visible area of the screen.

Modal Dialog Boxes

One particularly common barrier occurs when developers add modal dialog boxes to Web content. These boxes typically open over top of the page being viewed, with the content of the page below faded to bring visual focus to the contents of the box. For screen reader users, the cursor's focus often remains on the page hidden behind the box, rather than moving into the box itself. When a modal dialog box is opened, the proper sequence is to send the cursor's focus to the first element within the box, and when navigating through the box, the focus should remain inside it until the required action has been completed. Once the action has been completed, the cursor's focus should then be sent back to the page below to the position where the box was originally opened from.

Key Point: A meaningful sequence when navigating through a webpage using a keyboard is from left to right, top to bottom, much like the path the eyes take when reading a book.

5. Use of Colour (Guidelines 1.4.1 Level A &1.4.3 Level AA)

While the use of colour can help to add meaning to Web content, the use of colour on its own to *represent* meaning can create a barrier for those who cannot see colour, including those who are blind, colour blind, or have low vision. Colour can also be problematic when text of one colour appears over a background of another colour, and the contrast between the two is insufficient. Contrast is included here as colour usage, though it is addressed by Guideline 1.4.3 at Level AA, and Guideline 1.4.6 at Level AAA.

1.4.1 Use of Color: Color is not used as the only visual means of conveying information, indicating an action, prompting a response, or distinguishing a visual element. (Level A)

1.4.3 Contrast (Minimum): The visual presentation of text (Links to an external site.) and images of text (Links to an external site.) has a contrast ratio (Links to an external site.) of at least 4.5:1, except for the following: (Level AA)

- **Large Text**: Large-scale (Links to an external site.) text and images of large-scale text have a contrast ratio of at least 3:1.
- **Incidental:** Text or images of text that are part of an inactive user interface component (Links to an external site.), that are pure decoration (Links to an external site.), that are not visible to anyone, or that are part of a picture that contains significant other visual content, have no contrast requirement.
- **Logotypes:** Text that is part of a logo or brand name has no minimum contrast requirement.

Guideline 1.4.1 Explained

The most common form of colour blindness is red/green insensitivity. Of this group about half have Protanopia with a deficiency of cones to process red light, and the other half have Deuteranopia with a deficiency in cones to process green light. An estimated 10% of a population (more commonly males than females) have some form of colour blindness. For these groups, a red stop button and a green start button may not be distinguishable from one another. To avoid this potential barrier, the red button should have the word “stop” added to it and the green have the word “start” added to it.

There are other much less common forms of colour blindness, including those with achromatopsia who see no colour, and tritanopes who have trouble distinguishing blues and purples, as well as seeing shades of green as blue and oranges as red.

Another common barrier results when error messages are presented in red colour alone, without some other indicator of the error. Think about what happens if a person with a common form of colour blindness submits a form, but leaves some of the required fields empty. He or she would receive a returned form with the missing fields highlighted in red and the correct fields highlighted in green, but would not be able to tell them apart.

In terms of contrast issues, it is not uncommon for sites to use a particular colour palette with complementing colours to create a colour scheme throughout a site. Often these colours do not contrast well, which is fine in many cases, though they can create a barrier if those colours are used for text and background colours.

Technical:

It is often possible for those with full sight to intuitively identify when contrast between text and background is insufficient. There are many tools available to measure colour contrast that compare two colour codes (e.g., #ffffff or #000000 for white and black). Where there appears to be insufficient contrast, the colour codes for the text and background should be tested. These colour codes can be found most easily by right clicking on an element, then choosing Inspect Element from the browser’s popup context menu, then examining the CSS elements in the right pane of the inspector. See the video that follows for details on finding and testing colour codes. The requirement for conformance with Level AA is a contrast ratio of 4.5:1 for smaller sized text such as the text you are reading right now, and 3:1 for larger text, such as the heading at the start of this page.

For more about colour blindness visit the following resource:

Colour Blind Awareness (Links to an external site.)

6. Keyboard Accessible (Guideline 2.1.1 Level A)

This guideline is relevant to providing access for people who are blind and unlikely to be using a mouse, as well as other groups with various mobility impairments that may not be able to handle a mouse.

2.1.1 Keyboard: All functionality (Links to an external site.) of the content is operable through a keyboard interface (Links to an external site.) without requiring specific timings for individual keystrokes, except where the underlying function requires input that depends on the path of the user's movement and not just the endpoints. (Level A)
Note 1: This exception relates to the underlying function, not the input technique. For example, if using handwriting to enter text, the input technique (handwriting) requires path-dependent input but the underlying function (text input) does not.

Note 2: This does not forbid and should not discourage providing mouse input or other input methods in addition to keyboard operation.

Guideline 2.1.1 Explained

Failure to provide keyboard access to functional elements in Web content is a major barrier for many Web users. It is often overlooked by developers, who tend to be mouse users themselves. There are many people who may rely on a keyboard to navigate the Web, including those who are blind or have low vision, as well as those with mobility impairments.

Technical:

JavaScript event handlers:

Implementing keyboard access is done using JavaScript event handlers. Event handlers can be mouse specific (e.g., onmouseover) or keyboard specific (e.g., onkeypress) or they can be device independent (e.g., onfocus) working with either mouse or keyboard actions.

When developing custom interactive elements in Web content, developers should attempt to use device independent event handlers whenever possible. They may, however, use both mouse and keyboard event handlers together.

Device independent event handlers:

- onfocus
- onblur
- onselect
- onchange

Note: *Be careful when using these event handlers that they do not cause a change of context (see Guidelines 3.2.1 & 3.2.2) such as redirecting a user to a different location without their explicit request to do so.*

Paired equivalent mouse and keyboard events handlers:

- onclick (onmousedown + onmouseup) = onkeypress (onkeydown + onkeyup)
- onmousedown = onkeydown
- onmouseup = onkeyup
- ondblclick = no keyboard equivalent (avoid using this event handler)
- onmouseover = onfocus
- onmouseout = onblur

Key Point: Although onclick suggests a mouse click is required to activate an event, in reality browsers and assistive technologies have implemented onclick to function with either a mouse click or a keypress. Therefore onclick is now considered a device

independent event handler.

CSS device dependent pseudo-classes

There are also CSS pseudo-classes for mouse and keyboard events to be aware of. When creating styles use both pseudo-classes together to provide both mouse and keyboard activated styles. For example, a style might be created to highlight links with a background colour when a mouse pointer hovers over a link using the "**:hover**" style. To provide the equivalent highlighting when a user uses the Tab key to navigate to the link, use the "**:focus**" style. Here's an example using both pseudo-classes to create a background colour for links that activate with a mouse hover or with keyboard focus.

```
a:hover, a:focus{
  background: blue;
}
```

Key Point: Failing to provide keyboard access to functionality in Web content will be a barrier for a large number of Web users.

7. Provide Ways to Navigate (Guideline 2.4.1 Level A)

Providing ways of navigating around within pages makes it possible to move through elements on a page more quickly, simulating visual scanning for those who are blind. Within page navigation can also make navigating more efficient for those who rely on a keyboard.

2.4.1 Bypass Blocks: A mechanism (Links to an external site.) is available to bypass blocks of content that are repeated on multiple Web pages (Links to an external site.). (Level A)

Guideline 2.4.1 Explained

While Bypass Blocks provide the ability to skip past blocks of repetitive content, like large complex menus or long lists of links, it's better to think in terms of within page navigation to improve accessibility for those who are unable to scan and click. For screen reader users, proper use of **headings**(<h1>, <h2>, etc.) is an easy way to provide within page navigation. Screen readers then list the headings on a page, allowing the user to navigate through the list to a particular heading, then press the Enter key to jump directly to the heading on the page.

You may encounter **bypass links**, which were used historically to help screen reader users navigate page content. With this strategy, an invisible image with alt text such as "jump to main content" was placed in the top left corner of a page and linked to an anchor at the beginning of the main content. Bypass links might also be hidden links that appear when they receive focus, so sighted keyboard users can also make use of them.

While bypass links do satisfy the criteria for this guideline, today you are more likely to encounter **ARIA Landmarks** – a particular set of ARIA roles that can be added to various elements to define their purpose, and provide the ability to jump to different regions of the page. The equivalent to the main content bypass link described above is the ARIA landmark "main," which would be added as an attribute to the main element containing the primary content of the page. Screen readers can list the landmarks on a page, much like they can list the headings on a page, use the down arrow until "main" is announced, then press enter to jump to the main content region.

Technical:

Main Landmark

```
<div role="main>
main content goes here
…
```

Navigation Landmark

<ul role=”navigation>

Home

Products

...

Technical:

Full List of Landmark Roles

- **banner**: Typically associated with the header area of a page. There should only be one banner landmark per page.
- **complementary**: A section of content that complements the main content but also retains its meaning when separated from the main content. Often used with a region containing advertising, or promo items aligned down the right side of the page. There can be multiple areas defined as complementary.
- **contentinfo**: Contains the content usually found in the footer of a page, like copyright and privacy statements. There should only be one contentinfo landmark per page.
- **form**: Contains form input elements that can be edited and submitted by the user. Multiple elements can have the form role.
- **main**: The main content of the page. There should only be one main landmark per page.
- **navigation**: A collection of navigation links used to navigate the site or page. There can be multiple elements with the role navigation.
- **search**: A search tool. There can be multiple search tools.
- **application**: Represents a unique functional unit, and keyboard commands are handled by the application rather than the browser or the assistive technology itself. An embedded movie player, a calendar widget, or other customized software embedded in Web content, are examples where the application role might be used. This role should be used sparingly as it can create some confusion for screen reader users when key commands begin working differently.

When using landmarks, **each area of the page should be enclosed in elements that have a landmark role associated** with them. No content should be orphaned outside of a landmarked area. Also, **where HTML5 equivalent elements are used, a landmark role is not required**. For example, when using HTML5 <header> the banner role need not be added. When using HTML5 <nav> the navigation role need not be added. When HTML5 <footer> is used, the contentinfo role need not be added. Similarly, do not use the form role in a form element. It already has a role of form associated with it as part of the HTML specification.

In general, if there is a way to create a feature using standard HTML, use it instead of creating a custom feature and adding ARIA. For instance, don’t create a form using s and s, adding scripting to create functionality, and using CSS to make these elements look like a form. Instead, use actual <form> markup to create the form. Standard elements already have the semantics that ARIA might add to custom elements, and are likely to be more broadly supported.

Other Within Page Navigation

There are a few other cases when within page navigation can be added to improve accessibility. When there are embedded objects like a movie player or some interactive Flash, assistive technology users can benefit from a bypass link just before the object (with appropriate alt text to describe the purpose), that when followed repositions the cursor after the object. Where complex or large tables of data are presented, a bypass link can be provided to allow users to skip over the table. Without these added bypass links, navigating past these embedded objects can be time-consuming.

8. Link Purpose - In Context (Guideline 2.4.4 Level A)

When creating links it is important that the text of a link be meaningful. If it is not, a person who is blind using screen reader may be required to read the text surrounding the link, or actually follow the link, in order to determine where it leads. Either requires extra effort for screen reader users and sighted users alike, that can be avoided by simply using text that describes the destination or function of the link.

2.4.4 Link Purpose (In Context): The purpose of each link (Links to an external site.) can be determined from the link text alone or from the link text together with its programmatically determined link context (Links to an external site.), except where the purpose of the link would be ambiguous to users in general (Links to an external site.). (Level A)

Guideline 2.4.4 Explained

You have almost certainly come across links like “click here” or “more” in your travels through websites. These links on their own provide no useful information about the destination or function of the link. This particular guideline refers to links “within context,” meaning that there may be information in the surroundings that give meaning to these otherwise meaningless links. This should be distinguished from the requirements of Guideline 1.4.6 (Level AAA) which requires link text to be meaningful regardless of context.

In context a link like “more” can be made meaningful by a link that precedes it. For example, many news sites will provide a short excerpt from an article, followed by a “more” link to read more about the topic. In such cases if the title of the article is itself a link to the full article, that title gives meaning to the “more” link by virtue of its proximity. Screen readers will read the title link first followed by the "more" link, then read the next title then more, and so on. In such cases it is relatively easy to make the connection between the article title and the otherwise meaningless “more” link. In such cases the "more" link is acceptable for Level A conformance.

Screen readers will have a function that allows a user to list the links on a webpage. The user can listen to the link text for each, and jump directly to a link of interest. If those links are a series of “click here” links, that link list becomes unusable if there are no surrounding links present to add meaning. Screen readers can also sort a list of links alphabetically. In such cases, any context that might have been provided by surrounding links, such as news article titles in the example above, are lost. Hence at Level AAA, Guideline 1.4.6 requires that link text be meaningful on its own.

Though the Level A requirement is only that link text be meaningful in context, it is good practice to use link text that is meaningful on its own. This will make the links more usable for everyone, including those with full sight, who also have to resort to searching the surrounding text to figure out where a “click here” link leads.

Key Point: Link text should describe the destination of a link or its function if it operates a feature of a webpage. Do not use “click here” as link text.

Technical:

Example of a link in context. The linked title of the article above adds meaning to the "more" link found at the end of the paragraph.

<h3>Water Found on Mars</h3>
Potentially life-giving water still flows across the ancient surface of Mars from time to time, NASA scientists said Monday in revealing a potential breakthrough in both the search for life beyond Earth and human hopes to one day travel there. more

9. Error Identification (Guideline 3.3.1 Level A)

If a user makes an error when filling out a form, a message identifying the error usually appears. It is important that forms are created in such a way that screen readers can find and read error messages, otherwise the forms become unusable.

3.3.1 Error Identification: If an input error (Links to an external site.) is automatically detected, the item that is in error is identified and the error is described to the user in text. (Level A)

Guideline 3.3.1 Explained

When an error occurs after submitting a form, there are a variety of different ways to present the error messages. A common strategy is to present a short message next to the fields that were missing, such as “required field left empty.” Another is to display a box near the top of the page that lists the errors that have occurred. Typically the form is submitted and the page reloaded before the messages are displayed, in which case a screen reader would begin reading the page once more from the top left. At that point the user would have no idea that an error had occurred, and would only discover this by navigating into the page.

In the past a good strategy for the case above would have been to dynamically update the title of the page to include words like “an error occurred” (the page title is the first thing a screen reader reads) then present a message at some consistently used area of the page, often at the top of the main content area, to which a screen reader user could navigate to listen to the message. While this strategy works if the user knows where the message area appears, it still requires some effort to find.

A better strategy might be to use scripting to process the form before it gets submitted. If an error occurs, a scripted dialog would open describing the error, and once acknowledged by pressing the dialog’s OK button, the cursor’s focus would be sent to the first field in the form where an error occurred, all without submitting the form or reloading the page.

With the introduction of ARIA, it is now possible to announce messages automatically when they appear on the page using the ARIA role “alert” within the element containing the message. An ARIA alert is like a live region. When the content updates or appears, a screen reader will announce the message automatically, regardless of where the cursor might be located on the page.

Technical:

Error messages that are injected into a page dynamically should include the ARIA alert role. In the case below the message is hidden in the page next to the form input, and scripting is used to change the display:none style to display:inline, at which point the ARIA alert reads the content of the element aloud. With the same block of scripting the cursor’s focus can be sent to the first field where an error is being displayed, so the users can immediately correct the error without having to go searching for the field.

```
<label for=”firstname”>First Name</label>

<input type=”text” id=”firstname” />

<div style=”display:none” role=”alert”>
  The first name field is required.
</div>
```

In addition to presenting error messages in a way that screen readers can detect them, success messages should also be used where feasible, to indicate that a particular action has completed successfully. For instance, after completing a registration form successfully, present the message “You are now registered” or something to that effect. This removes the need to search through the page that loads after submitting the form to determine whether it was successful or not.

Key Point: Be sure that screen reader users are aware of feedback messages. Use the ARIA role=”alert” on dynamically injected messages so they get read automatically.

10. Labels and Instructions (Guideline 3.3.2 Level A)

When we explored Guideline 1.3.1, we learned about the accessibility benefits of using the HTML <label> element to explicitly describe form elements. Labels in Guideline 3.3.2 also include descriptions in general, which help a user understand how something is used. In addition to labels it is often necessary to provide additional information or instructions on how the form or other interactive elements should be used. Instead of making screen reader users navigate through a form, complex menu or other interface element to understand how its layout or function, provide instructions ahead of time.

3.3.2 Labels or Instructions: Labels (Links to an external site.) or instructions are provided when content requires user input. (Level A)

Guideline 3.3.2 Explained

There are many ways instructions can be added to describe how a particular feature is used. This can benefit all users, but is particularly important for those who cannot see how a feature is arranged. Take for instance a large complex menu that might be found across the top of a webpage. Holding a mouse pointer over a top level menu item opens a submenu; within that submenu holding a mouse pointer over an item opens a sub-submenu, and so on. Understanding how these menus operate can be difficult to grasp when you can't see what's going on. These complex menus can be made easier to use if their functionality is described first.

Technical:

A typical description can be added to a hidden element somewhere on the page, then associated with the main element containing the menu using aria-describedby="[span id]" with the id value associated with the .

In this example the functioning of a monster menu is described in a hidden element. The element is identified by the id "menu_description". The menu is contained within the identified as "monster_menu" which uses aria-describedby to associate the menu_description with the . Lastly the has a tabindex="0" added to it to make it keyboard focusable. When a screen reader user uses the Tab key to navigate to that , the contents of the associated element is read to describe how to use the menu.

Use the Tab key to enter or exit the menu. Use the left and right arrow keys to move from one top level menu item to the next and vice versa. When on a top level menu item, press the down arrow to open its submenu and to navigate down through items in that submenu. Use the up arrow to move up through that submenu, back to the top level menu item. While in a sub menu use the right arrow key to move into sub submenus when they are announced, and the left arrow to exit that sub submenu.

monster menu goes here

There are a variety of other ways to provide instructions of this sort. In the case above, the description of the monster menu is hidden away, though it could be made visible for the benefit of others.

Describing how a feature works in the surrounding text is useful for everyone. For instance, when filling out a registration form, there could be some general instructions at the start of the form that describe the organization of the form, which elements are optional, what format particular input elements take (e.g., password must be at least 8 characters), and so on. At the same time aria-describedby can be used to explicitly associate the description with the form to ensure screen reader users are aware of the instructions.

You can also use aria-label to add instructions. In the example above the text content of the menu_description element could be added directly to an aria-label attribute.

Key Point: Provide explicit instructions describing how things work, and associate those instructions with the relevant feature using aria-describedby.