
Association for
Computing Machinery

Advancing Computing as a Science & Profession

# Association of Computing Machinery (ACM) Style Guide, WCAG 2.0 Level A

ACM
12/19/16

## TABLE OF CONTENTS

# OVERVIEW

The Association of Computing Machinery (ACM) is committed to diversity, inclusion and accessibility in everything we do. These core values are fundamental to the way we operate our organization and support our constituents. ACM works with SSB BART Group to ensure compliance with accessibility standards through audits and policy development.

This style guide is intended to assist developers, designers, project managers, or other employees engaged in developing or creating content for the web. This style guide includes WCAG 2.0 Level A high priority accessibility best practices, examples, and implementation techniques for the associated best practices and examples.

# MEDIA TYPES

## IMAGES

All images provided within a web site or web application must have meaningful alternative text. Meaningful text equivalents, provided either via the `alt` attribute or via an external `longdesc` file, ensure that users with disabilities can access image content. Screen reader software announces image alternatives to users who are blind or visually impaired. Users who have low vision rely on alternative text when images are small, have poor contrast or when images are magnified to large sizes and become pixilated. Users of voice-recognition software utilize the alternative text of the image to gain focus on actionable images.

### BEST PRACTICE: PROVIDE ALTERNATIVE TEXT FOR IMAGES

#### EXAMPLE

**Figure 1: All images that convey meaning must provide meaningful alternative text**

```
<img alt="Man with cavity"
src="https://s3-us-west-
2.amazonaws.com/s.cdpn.io/t-
168/cav1.jpg">

<img alt="Young woman with
cavity" src="https://s3-us-west-
2.amazonaws.com/s.cdpn.io/t-
168/cav2.jpg">

<img alt="Pig eating chocolate
cake" src="https://s3-us-west-
2.amazonaws.com/s.cdpn.io/t-
168/pigcake.jpg">
```

#### RECOMMENDATIONS

Add an appropriate `alt` attribute for images including poster images of videos that do not contain `alt` attributes. An `alt` attribute should be a concise and meaningful replacement for the image.

If the image does not convey any meaning, such as a spacer image or separation line, or if the image is redundant to adjacent text enter a null `alt` attribute (`alt=""`) should be assigned, the image should be turned into a CSS background image.

## BEST PRACTICE: ENSURE COMPLEX IMAGES PROVIDE SUFFICIENT DESCRIPTIONS

### EXAMPLE

| City | Consumption Amount |
|---|---|
| Las Vegas | High |
| New York City | Medium |
| Raleigh | Medium |
| Miami | Medium |
| Austin | Small |
| Seattle | Small |
| ... | ... |
| ... | ... |

**Figure 2: Complex images require a description of the image along with alternative text to provide context to the image**

```
<div id="chocmap">
<img src="https://s3-us-west-2.amazonaws.com/s.cdpn.io/t-
168/chocmap.png" alt="Map showing chocolate consumption capitals
Seattle, Los Angeles, Austin, Chicago, New York, Raleigh, and Miami">
<img src="https://s3-us-west-2.amazonaws.com/s.cdpn.io/t-
168/chocmap2.png" alt="Chocolate Consumption Map">
<table class="table">
…
</table>
```

### RECOMMENDATIONS

Create an appropriate description that describes the image in detail. Ideally the longdesc attribute should not be used to reference another page as this description may not be keyboard accessible and is only often available to screen reader users (based on the browser implementation). A longer description should be provided on the same page of the image or via a link near the image.

## BEST PRACTICE: ENSURE CSS BACKGROUND IMAGES THAT CONVEY MEANING HAVE TEXTUAL AND VISIBLE EQUIVALENTS

### EXAMPLE

```
<a class="header-btn search-btn"
href="#" title="Search" aria-
expanded="false">Search</a>
<style>
.btn-container .search-btn {
    background-image:
url("http://www.ssbbartgroup.com/w
content/themes/ssbbart/css/../image
s/layout/icon-search.png");
}
</style>
```


**Figure 3: The Search button is a background image. A visual textual equivalent is provided on screen.**

### RECOMMENDATIONS

Alternative text should be provided on-screen, as CSS positioned off-screen text, or by using a standard inline `img` with equivalent alternative text in addition to the background image.

The key to meeting this requirement is that the alternative for the background image must be accessible to users of assistive technology as well as visually to users with low vision who cannot view CSS images but that may not be using assistive technology.

## COLOR AND CONTRAST

When colors are used as the sole method for identifying screen elements, controls, or giving instructions, persons who are blind, color blind, or have low vision may find the web page unusable.

### BEST PRACTICE: ENSURE COLOR IS NOT THE SOLE MEANS OF COMMUNICATING INFORMATION OR INDICATING ERROR MESSAGES

### EXAMPLE

```
<h1>Contact Information</h1>
<p>* = Required Field</p>
<label class="required" for="fname">*First
Name</label>
<input type="text" id="fname">
<label class="required" for="lname">*Last
Name</label>
<input type="text" id="lname">
<label for="email">Email Address</label>
<input type="text" id="email">
```

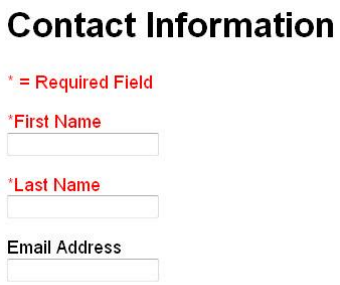



**Figure 4: The forms provide a visual cue as well as color to indicate required fields.**

### RECOMMENDATIONS

Ensure that all information communicated via color is available through some other method of identification, such as on-screen text labels. For indicating required fields, one method is to use a character to indicate the required field. The character and its meaning should be described in instructions located at the beginning of the form.

### BEST PRACTICE: ENSURE TEXT AND IMAGES OF TEXT PROVIDE SUFFICIENT CONTRAST

### EXAMPLE

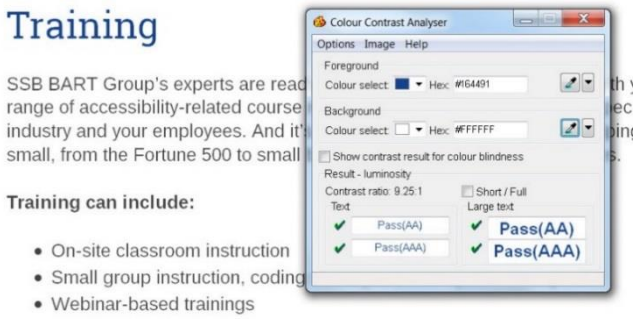



**Figure 5: Blue text against a white background provides sufficient contrast, with a luminosity contrast ratio of 9.25:1.**

### RECOMMENDATIONS

Text less than 18 point or bold text less than 14 point must have a luminosity contrast ratio of 4.5:1 or more. Text 18 point or larger or bold text 14 point or larger must have a luminosity contrast ratio of 3:1 or more.

## NAVIGATION

Navigation requirements ensure that accessible navigation structures are provided for users with disabilities. Provision of accessible navigation structures will affect multiple types of users with disabilities and is a key requirement across all leading accessibility standards.

### BEST PRACTICE: PROVIDE A MECHANISM FOR SKIPPING PAST REPETITIVE NAVIGATION LINKS

### EXAMPLE

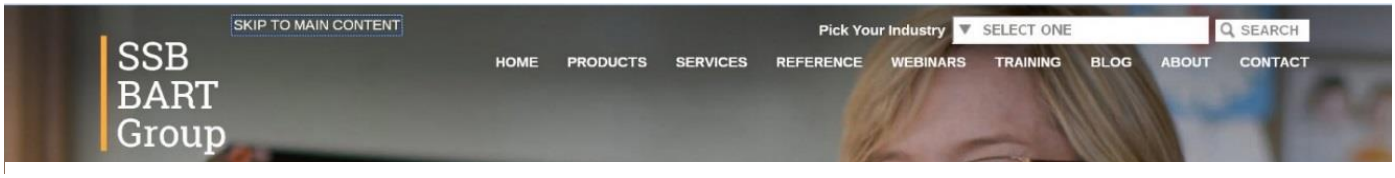



**Figure 6: The page contains a "Skip to Main Content Link" in order to skip past navigation links.**

```
<style type="text/css">
a.nav {
      position: absolute;
      left: -10em;
      top: -10em;}
a.nav:focus, a.nav:active {
      position: absolute;
      left: .5em;
      top: .5em;
      background-color:lightblue;}
</style>
```

```
<body>
<div style="margin-top:1.5em;">
<a class="nav"
href="#jumptocontent">Skip to main
content </a>
<a href="[URL of first link]">First
Repetitive Link</a>
(many intervening links and other
content)
<a href="[URL of last link]">Last
Repetitive Link</a>
</div>
<a name="jumptocontent" tabIndex="-
1"> </a>
```

```
<!-- (main content area on the page, ideally beginning with an h1 element) -->
```

```
</body>
```

RECOMMENDATIONS

Include a mechanism allowing users to skip past groups of links. When links appear at the top of the page, the skip link should be the very first active element in the page (i.e. the first tab stop), and its target should be the beginning of the main content area that is unique to that page. The easiest way to accomplish this requirement is through the use of intra-document links. There are multiple 'forms' that a skip link can take, such as a text link, an image link, or something else.

### BEST PRACTICE: ENSURE PAGES DO NOT AUTOMATICALLY REFRESH

EXAMPLE

Non-Compliant Example

```
<body
onload=setTimeout("location.href=
'http://www.example.com'",5000)>
  <a href="#">Reload</a>
</body>
```

Compliant Example

```
<body>
<p>Please reload this page!
<a href="#">Reload</a>
</body>
```

RECOMMENDATIONS

If a page must refresh itself allow the user to control whether or how the page refreshes. Often page refreshes are performed using a meta refresh with a time-out (or a page being redirect to itself after a certain amount of time). The page refresh must be able to be :

- Turned off
- Adjusted or
- Extended

There are exceptions to when pages can refresh including if 20 hours has elapsed, if there is an exception for real-time data transactions, or when it is essential to the purpose of the page.

## PAGE STRUCTURE

Page structure is concerned with ensuring that proper structural markup is used within pages. This is especially important for assistive technology which may or may not be programmed to perform the same level of "coping" that can be found in browsers (which are created by multinational corporations) or assistive technology brands that cost several hundred dollars or more. Even in the case of "better" assistive technologies, poor markup will cause rendering problems, to the point of reading text out of order or even not reading it at all.

### BEST PRACTICE: ENSURE THAT THE READING ORDER OF CONTENT IS LOGICAL

EXAMPLE

```
<div id="main-container">
<div>
<div class="column1">Name:</div>
<div class="column2">Jane Doe</div>
```

Name: Jane Doe

Age: 35

```
</div>
<div>
<div class="column1">Age:</div>
<div class="column2">35</div>
</div>
</div>
```

Figure 7: The visual text on the page is displayed in a logical order in the DOM to ensure screen reader users and users who remove styles from the page render the content in a logical order.

#### RECOMMENDATIONS

Alter the relative order of the rendered page so that content including structural markup and the content to which it applies are read logically by assistive technology. This can be achieved by re-ordering the source code or inserting content into the DOM in an a logical location relative to related content. Ensure that elements of the DOM are ordered in the same manner as they would be read visually on the page.

### BEST PRACTICE: ENSURE CONTENT IS VISIBLE TO ASSISTIVE TECHNOLOGIES

#### EXAMPLE

Non-Compliant Example

```
<style>
p.note:before {
  content: "Note: "
}
</style>
<body>
<p class="note">Hello World</p>
</body>
```

Note: Hello World

Figure 8: The text "Note" is provided within the HTML to ensure content is rendered by assistive technologies

Compliant Example

```
<p>Note: Hello World</p>
```

#### RECOMMENDATIONS

Until assistive technology products are able to fully access content that exists outside of the DOM, all content that an assistive technology user is expected to access should be rendered in more standard ways that are reliably part of the page's DOM.

## KEYBOARD ACCESSIBILITY

All functionality must be actionable regardless of the input method used. This requirement is necessary to ensure that people who are blind, people with low vision, and people with dexterity impairments who do not use the mouse can access all functionality.

### BEST PRACTICE: AVOID THE SOLE USE OF DEVICE-DEPENDENT EVENT HANDLERS

#### EXAMPLE

```
<div tabindex="0"
onclick="alert('you clicked.
onclick was activated');"
onkeypress="if
(event.keyCode == 13) {
```

View this onclick with onKeyPress example

Figure 9: `onClick` and `onKeyPress` are used to provide keyboard and mouse functionality to the custom control.

```
alert('you pressed enter');
}">View this onclick with
onKeyPress example</div>
```

### RECOMMENDATIONS

Insert a redundant device-dependent event handler to allow users of assistive technologies a method to access the functionality. Event handlers that depend on a device, generally the mouse, should contain redundant non-device-dependent event handlers.

Ensure that when keyboard event handlers are associated with elements that the element must be able to receive keyboard focus. For elements that do not typically receive focus such as the `div` or `img` element a `tabIndex` attribute of 0 or greater must be set on the element. Note, anchor tags without href attributes but with `onClick` attributes will not by default receive keyboard focus. The following device dependent event handlers correspond to the device independent event handlers:

| Event Handler | Alternative |
|---|---|
| onClick | onKeyPress |
| onMouseDown | onKeyDown |
| onMouseUp | onKeyUp |
| onMouseOver | onFocus |
| onMouseOut | onBlur |
| onDblClick | onKeyDown |
| ng-Click (Angular) | Any Key Event |

## BEST PRACTICE: ENSURE ALL ACTIVE ELEMENTS RECEIVE KEYBOARD FOCUS

### EXAMPLE

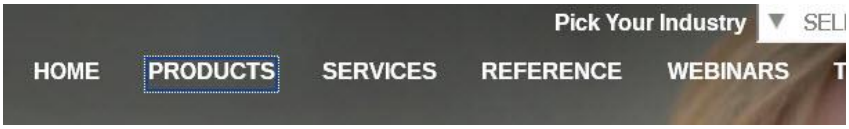



**Figure 10: The "Products" link provides visual focus as well as programmatic focus. Any element that can be activated with the mouse, must also be activated with the keyboard.**

### RECOMMENDATIONS

Ensure that all actionable elements are accessible from the keyboard. This can be achieved by using standard HTML elements that are, by default, actionable from the keyboard, such as anchors and inputs. Other elements can be used to simulate user interface components such the `div`, `td`, and `span` tags, however, the `tabIndex` attribute must be assigned with 0, indicating the element's placement in the tab order. In addition, developers must convey to users of assistive technology that the element is actionable by assigning a role, and provide keyboard access through the use of device independent event handlers (i.e. `onkeypress`, `onkeydown`, `onfocus`, etc.)

## BEST PRACTICE: ENSURE CUSTOM CONTROLS PROVIDE PROPER TEXTUAL ROLES AND DESCRIPTIONS

EXAMPLE

```
<a href="..." title="Home - Tab - Selected">Home<span
class="hidden"> Tab - Selected</span></a>
<a href="..." title="User Activity - Tab">User Activity<span
class="hidden"> Tab</span></a>
<a href="..." title="Contact Info - Tab">Contact Us<span
class="hidden"> Tab</span></a>

<style>
.hidden {
  position:
absolute;
  left:-500px;
  top:0;
  width:1;
  height:1;
  overflow:hidden;
}
</style>
```


**Figure 11: The tabs indicate a role of link as well as the selection state.**

RECOMMENDATIONS

Ensure that proper textual descriptions are provided for custom components within the application. Indicate the role and state of the element. ARIA should also be used to indicate the control's role and state. For example, for a `div` or `span` element that represents a button, the role of "button" should be used. For elements that simulate links, the role should be "link". For anchors and buttons, the `onclick` event handles both keyboard and mouse clicking. For elements other than button or anchor, both `onclick` and keyboard event handlers will need to be used. The keyboard event handler such as `onKeyUp` will receive all key up events, and thus, must be modified to only trigger on the enter or space key for buttons or the enter key for anchors.

## FOCUS CONTROL

Focus control is a user's ability to control keyboard and reading focus within a web page or application. Keyboard focus is the location where keyboard actions will be interpreted by the application. It is often indicated visually by the cursor or a selection highlight, or programmatic dotted rectangle. Reading focus is the location where a screen reader begins to render content. Users who are blind, have low vision, or have mobility impairment rely heavily on proper control of keyboard and reading focus when browsing web based content.

### BEST PRACTICE: AVOID USING EVENT HANDLERS THAT TRIGGER FOCUS OR CONTEXT CHANGES ON USER INPUT

EXAMPLE

Non-Compliant Example

```
<label for="select1">Go to W3C Specifcation</label>
  <select id="select1" onchange="goToPage('select1')">
```

```
    <option value="http://www.w3.org/TR/HTML4">HTML 4
</option>
    <option value="http://www.w3.org/TR/CSS21">CSS 2.1
</option>
    <option value="http://www.w3.org/TR/WCAG10">Web Content
Accessibility Guideines 1.0</option>
    <option value="http://www.w3.org/TR/WCAG20">Web Content
Accessibility Guideines 2.0</option>
    <option value="http://www.w3.org/TR/UAAG10">User Agent
Accessibility Guidelines 1.0</option>
    <option value="http://www.w3.org/TR/ATAG10">Authoring
Tool Accessibility Guidelines 1.0</option>
  </select>
```


**Figure 12: Non-Compliant Example - The onChange event causes a new page to load when pressing the down arrow in the dropdown menu.**

Compliant Example

```
<p>
  <label for="select2">W3C Specifcations </label>
  <select id="select2" >
    <option value="http://www.w3.org/TR/HTML4">HTML 4 </option>
    <option value="http://www.w3.org/TR/CSS21">CSS 2.1 </option>
    <option value="http://www.w3.org/TR/WCAG10">Web Content
Accessibility Guideines 1.0</option>
    <option value="http://www.w3.org/TR/WCAG20">Web Content
Accessibility Guideines 2.0</option>
    <option value="http://www.w3.org/TR/UAAG10">User Agent
Accessibility Guidelines 1.0</option>
    <option value="http://www.w3.org/TR/ATAG10">Authoring Tool
Accessibility Guidelines 1.0</option>
  </select>
  <input type="button" value="Go To Specification"
onclick="goToPage('select2')"/>
</p>
```

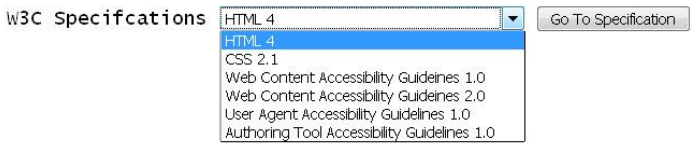



**Figure 13: Compliant Example - A button is provided after the dropdown menu to give the user control over changes in content.**

### RECOMMENDATIONS

For dropdown selection controls, add in a form button (i.e. Apply) that allows the user to trigger the form update event. Ensure that this element has a text value that properly communicates its function.

When an additional button is not an option, developers can use the `onBlur` event which will be fired when the user tabs out of the field to trigger focus changes. To preserve the 'one click' functionality for mouse users, the `onBlur` event should be coupled with the `onMouseDown` event which should trigger the change for mouse users. This approach should allow mouse and keyboard users to select the item without requiring an associated button.

## FORMS

Maximizing the ability to successfully fill out online forms is critical to ensuring that all users have access to the full functionality of web applications. Unless properly marked up, users with disabilities face significant challenges in filling out and submitting forms online. To ensure that users with disabilities can effectively utilize online forms, developers must ensure forms use proper markup, provide logical field grouping, and make sense when rendered by assistive technology.

### BEST PRACTICE: PROVIDE A VALID LABEL FOR FORM FIELDS

#### EXAMPLE

```
<label for="ship1">Ship To:
</label>
<input type="text" id="ship1"
name="shiptoaddr1" size="30">
```


**Figure 14: The "Ship To" label is directly associated with the input field by referencing the id of the input field.**

### RECOMMENDATIONS

Define form labels using the label element whenever an on-screen label is present. The label element's value is then explicitly associated with the relevant form field using the for and id attributes. Use of the label element provides an added benefit as it increases the target area that users can click to focus the form field.

### BEST PRACTICE: ENSURE CAPTCHAS ARE ACCESSIBLE BOTH VISUALLY AND AUDIBLY

#### EXAMPLE


**Figure 15: Google's reCAPTCHA provides audio and visual support for user input.**

### RECOMMENDATIONS

Add an alternative version of the CAPTCHA that uses a different sense such as an audible version of the CAPTCHA content. In addition, make sure the alternative is made prominently available and is implemented in an accessible way. For example, for an audio CAPTCHA provide a link to play the audio, and focus the edit field where the characters must be typed. The audio must be audible or else it won't be usable by users who are blind or visually impaired. The visual CAPTCHA should continue to be used to facilitate use by people who are deaf or hard of hearing

## BEST PRACTICE: AVOID USE OF PLACEHOLDER VALUES TO LABEL OR EXPLAIN INPUT

### EXAMPLE

```
<div data-ssb_ev="keydown,keyup" data-ssb_ts="4.5">
<label class="screen-reader-
text" for="s" data-
ssb_ev="keydown,keyup" data-ssb_an="search
for:" data-ssb_ts="4.5">Search for:</label>
<input id="search-
input" placeholder="Search" name="s" value="
" title="Search" data-
ssb_ev="keydown,keyup" data-
ssb_an="search" data-
ssb_ts="4.5" type="search">
<input id="search-submit" value="Go" data-
ssb_ev="keydown,keyup" data-
ssb_an="go" data-ssb_ts="4.5" type="submit">
</div>
```

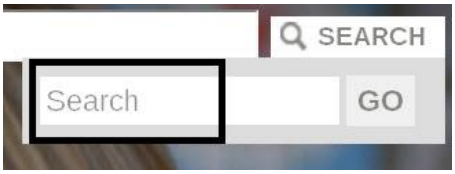



**Figure 16: Search field provides placeholder as well as an off-screen label. Placeholder is not taking place of the label for the form field.**

### RECOMMENDATIONS

Ensure that the `placeholder` attribute is not a replacement for the label element nor is it redundant with the label. Whenever multiple labels are associated with a single form field, use the `aria-labelledby` attribute, `aria-label` attribute or the `title` attribute to provide all the relevant text information for the field.

## BEST PRACTICE: ENSURE ELEMENTS WITH MULTIPLE LABELS ARE RENDERED APPROPRIATELY

### EXAMPLE

```
<label id="l1">Sodium
intake:</label>
<input aria-labelledby="l1
span1" type="text" id="s1"
/>
<span id="span1">mg</span>
```

Sodium intake: mg

**Figure 17: Sodium Intake form provides two labels through an explicit label element and `aria-labelledby`.**

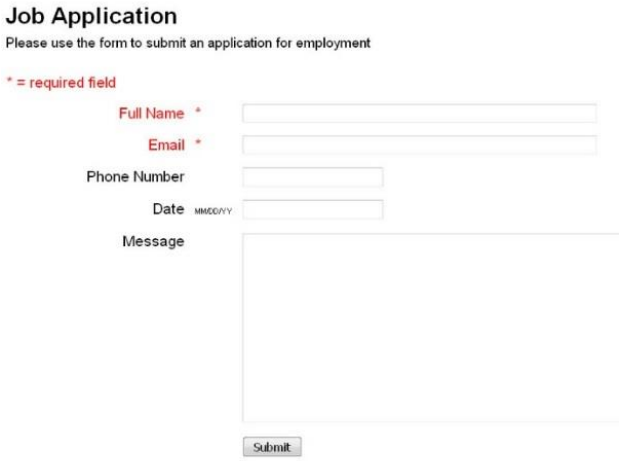
### RECOMMENDATIONS

Whenever multiple labels are associated with a single form field, utilize the `aria-labelledby`, `aria-label`, or `title` attribute or other appropriate markup or method to provide all the relevant text information for the field. Be sure to validate the implementation with Assistive Technology testing.

### BEST PRACTICE: ENSURE FORM FIELD CONSTRAINTS ARE CLEARLY INDICATED

#### EXAMPLE

```
<div>* = Required Field</div>
<form>
<label for="FullName">Full
Name*</label>
<input type="text" id="FullName">
<label for="Email">Email*</label>
<input type="text" id="Email">
<label for="pnumber">Phone
Number</label>
<input type="text" id="pnumber">
<label for=”date”>Date <span
id=”dateFormat”>
MM/DD/YY</span></label>
<input type=”text” id=”date”>
...
</form>
```




#### RECOMMENDATIONS

Ensure that all label text, including field name, required state, and any required formatting information is provided. This is best done by enclosing this information in the `label` element. There are additional techniques that can be used to associate constraints with fields including the `aria-required` attribute, the HTML5 `required` attribute and other mechanisms to assign/associate an accessible name to a field such as `aria-label` and `aria-labelledby`. The `aria-describedby` attribute can also be used to associate form field instructions and constraints with a form field. Keep in mind that constraints including required or optional fields must be communicated both visually and programmatically.

## DATA TABLES

Properly structured data tables ensure that all users can access data that is presented along rows and columns. Data table structural markup will identify and associate table header cells with data cells in the body of a table. Users who are blind have significant difficulty accessing page content when proper table formatting is not provided.

### BEST PRACTICE: ENSURE DATA TABLE HEADERS ARE PROPERLY IDENTIFIED

#### EXAMPLE

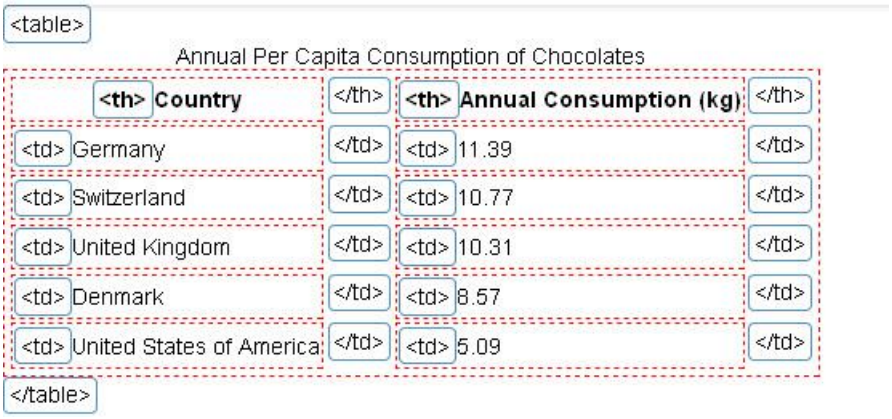



**Figure 18: Simple table using `th` tags for table headers and `td` tags for table data cells.**

### RECOMMENDATIONS

Specify table headers for the table utilizing the `th` element. Upon identifying table headers, ensure that the headers are properly associated with table content. For complex tables with multiple levels/headers, alternative approaches may be required such as splitting up a complex table into multiple simple tables.

## BEST PRACTICE: ENSURE HEADERS AND CELLS ARE PROPERLY ASSOCIATED

### EXAMPLE

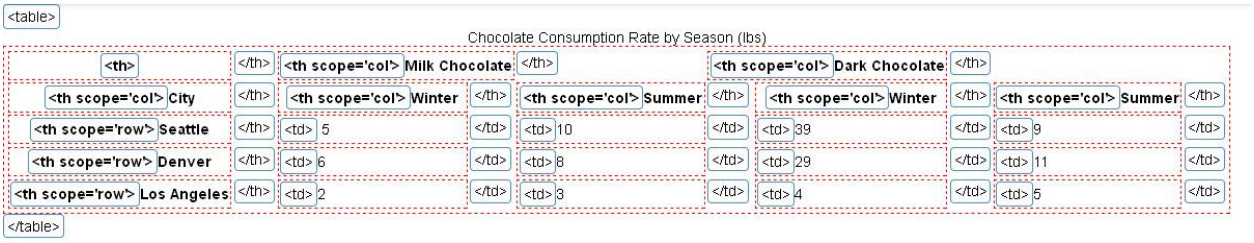



**Figure 19: Complex table with `th` tags and `scope` attribute on the columns and rows and `td` tags for table data cells.**

### RECOMMENDATIONS

Associating data cells with headers can be accomplished in two ways, the first works well for less complex data tables and enjoys a far greater level of support by assistive technologies like screen readers. The recommended implementation is to enclose the column header cells and (optionally) the row header cells in the `th` element (rather than `td`), AND to set the scope attribute to either `scope=col` (column headers) or `scope=row` (row headers). This explicitly associates the header cell with the other data cells in the same column (or row). When a header cell spans multiple columns or rows the appropriate `scope="colgroup"` or `scope="rowgroup"` should be used in lieu of `scope="col"` or `scope="row"`.

## BEST PRACTICE: ENSURE COMPLEX DATA TABLE HEADERS ARE PROPERLY IDENTIFIED

### EXAMPLE

<table>
<tr><td></td><td></td><th scope='colgroup'>Winter</th><td></td><th scope='colgroup'>Summer</th></tr>
<tr><td></td><td></td><th scope='col'>Morning</th><th scope='col'>Afternoon</th><th scope='col'>Morning</th><th scope='col'>Afternoon</th></tr>
<tr><th scope='row'>Denver</th><td>9</td><td>36</td><td>55</td><td>95</td></tr>
<tr><th scope='row'>Washington</th><td>20</td><td>40</td><td>70</td><td>92</td></tr>
</table>

**Figure 20: Complex table showing `th` tag and `scope` attribute on columns and rows, `colgroup` for merged cells, and `id` and `headers` to map `td` cells to `th` cells.**

### RECOMMENDATIONS

Complex data tables are any data table with more than one row or column of header cells and/or multiple levels. Data tables must identify the header cells that label the data cells in the same column or row. Indicating that a cell is a header cell via the `th` element allows screen readers and other assistive technologies to identify data tables from layout tables and provide properly navigate within them. For complex data tables, the header cells must be identified with an `id` so that each data cell (`td`) has a `header` attribute that references the `id` attribute of the `th` elements that are headers for the data cell.

## DIALOGS

Dialogs typically contain standard HTML elements and content that are displayed on top of the original page. The visual metaphor usually involves a graying-out or lightboard effect of the original page content behind the dialog, Persons who use a mouse typically benefit from functionality that makes the dialog appear at the screen position of the link (or other active element) that was clicked, and that either enforces its modality by trapping mouse clicks outside the dialog or its non-modality by dismissing the dialog if the mouse is clicked outside of it. To ensure the accessibility of these simulated controls to persons using assistive technology or the keyboard, several functional and textual requirements must be met.

### BEST PRACTICE: ENSURE DIALOGS USE PROPER STRUCTURE

### EXAMPLE

```
<div role="dialog" aria-
modal="true"
aria-labelledby="login">
    <h2 id="login" class=”sr-
only”>Log In</h2>
    ...
    <button>OK</button>
    <button>CANCEL</button>
</div>
```

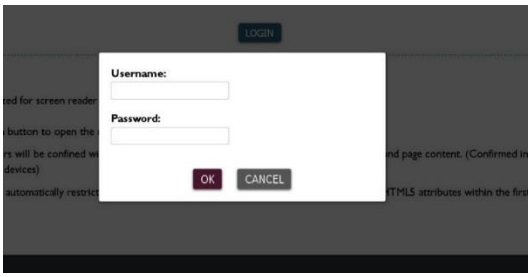



**Figure 21: Log In dialog with `role="dialog"` and `aria-labelledby` to provide a label for the dialog.**

### RECOMMENDATIONS

Use semantic structures such as the dialog element or an ARIA role of `dialog` (when appropriate). When the dialog is not modal or the dialog role is not appropriate other roles such as `role="region"` should be used with `aria-labelledby`.

## BEST PRACTICE: ENSURE THAT KEYBOARD FOCUS REMAINS WITHIN MODAL DIALOGS

### EXAMPLE

```
...
<div id="otherpagecontent">
<a aria-label="Create Report - Opens dialog"
href="javascript:openLayer();document
.getElementById('popup_layer1').focus
();">
Create Report</a>
</div>
...
<div role="dialog" aria-modal="true"
aria-labelledby="modal-title" aria-
describedby="desc"
tabindex="-1">
  <button id="b1"
onblur="document.getElementById('b1')
.focus();"
aria-label="close">X</button>
  <h2 id="modal-title">Create
Report</h2>
  <p id="desc">Dialog description</p>
  <p>Dialog text</p>
</div>
```

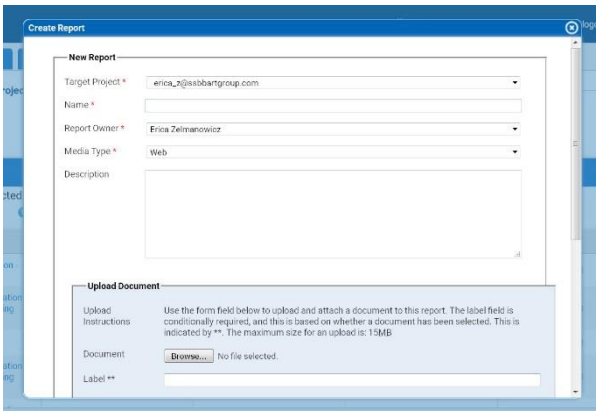



**Figure 22: Focus remains in the dialog when tabbing through the dialog content by using JavaScript focus.**

### RECOMMENDATIONS

Developers must ensure that when a modal dialog is open, focus remains within the dialog. This can be done by using `onFocus` and `onBlur` and other JavaScript techniques to manage the focus appropriately. Ensure that tab and shift+tab are handled appropriately. Ideally focus should wrap from the last element to the first element. It may also be beneficial to hide the rest of the page content from the screen reader user by setting `aria-hidden="true"` on the other page content.

## ARIA AND DYNAMIC CONTENT

ARIA is a W3C specification for the creation of Accessible Rich Internet Applications. ARIA aims to provide support to users of assistive technology in three main areas that were not previously addressed by the (X)HTML specifications: indicating main structural areas of a page, creation of roles and properties of user interface elements, and as a method to indicate alerts, page changes and dynamically updating information.

Valid ARIA markup should always be used. Developers should set ARIA properties correctly by including roles based on the intended purpose. When ARIA attributes (state, roles, and properties) are not used correctly assistive technology may not function as expected. For example, setting a role of dialog forces assistive technology into "forms" mode disallows the use of the virtual or browse cursor. Another example is that if the content enclosed in the element is primarily static content that is not actionable, a role of dialog is likely not appropriate. Any content in a dialog must be obtainable either by directly tabbing or be associated with content that is in the tab order.

## BEST PRACTICE: AVOID FORCED FOCUS CHANGES THAT ARE NOT USER-INITIATED

EXAMPLE

```
<body onload="setTimer(2000);">
  <label for="fname">First
Name:</label>
  <input type=text id="fname">
  <br>
  <label for="lname">Last
Name:</label>
  <input type="text id="lname">
  <p>The following section is
updated every 2 seconds with
current data</p>
  <div tabIndex=0 accessKey="k"
id="dynam" style="border:solid;
border-color:blue; border-
width:thin;">Dynamic Content</div>
</body>
```

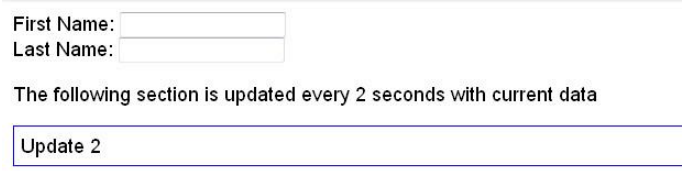



**Figure 23: An access key ("k") is provided to allow keyboard only users to move focus to the newly updated content.**

RECOMMENDATIONS

Ensure that focus is not forcibly shifted to a new location without user interventions except when immediate user interaction is needed and instructions are provided to the user indicating where focus will be moved (for example emergency alerts and time out warnings that allow the user to request additional time. It may be useful to provide an option to allow forced focus changes that are not triggered by user interaction to be disabled as a user option.

## BEST PRACTICE: INDICATE LIVE REGIONS FOR DYNAMICALLY CHANGING CONTENT

EXAMPLE

The highest point in Florida is 345 feet above sea level.

**Figure 24: ARIA live region is provided to render updated content to screen reader users. aria-live="polite" will render content when the user pauses navigation.**

```
<span id="LiveRegion1" aria-live="polite" aria-atomic="false">The highest
          point in Florida is 345 feet above sea level.</span>
```

RECOMMENDATIONS

Mark areas of the page that change as live regions using the WAI-ARIA states and properties module. In order for documents/application that include live region properties to validate, they should be served as HTML 5 or should be prefixed by the WAI-ARIA namespace aria-live. Each live region may declare the following properties:

1. `aria-live` (`off, polite, or assertive`): the level of priority of an alert, or the amount of verbosity screen reader users can expect from an alert or live region.
2. `aria-labelledby`: points to IDREFS that provide labels for the widget

3. `aria-describedby`: points to IDREFS that provide more descriptive text about the widget
4. `aria-controls`: points to IDREFS (a list of space-delimited references to unique document identifiers) that reference elements that are controlled by the current element
5. `aria-atomic` (`true/false`): whether an entire live region or just the updated portion should be announced when a change occurs
6. `aria-relevant` (`additions, removals, text, all`): indicates the relevant changes that occur within a region. This property accepts a space-delimited list of property values.

## BEST PRACTICE: ENSURE CONTENT UPDATES DEFINE FOCUS UPDATES APPROPRIATELY

### EXAMPLE

```
...
<form>
<p>Enter text in the search field
and results will be displayed
incrementally below
the search text field<p>
<label for="textSearch">Incremental
Search:</label>
<input type="text" id="txtSearch"
onkeyup="updateContent(this.value)"
>
</form>
<p>Results:
<div tabIndex=-1 accessKey="s"
id="txtToUpdate"></div></p>
...
...
<script type="text/javascript">
var xmlHttp;
...
function updateContent(str) //
called by onKeyUp
{
...
xmlHttp=GetXmlHttpObject();  //
function not shown
...
xmlHttp.onreadystatechange=stateCha
nged;
xmlHttp.open("GET",url+"?param="+st
r,true);  // url is defined in code
not shown
xmlHttp.send(null);
}
...
function stateChanged() {
if(xmlHttp.readyState==4)
  {

document.getElementById("txtToUpdat
e").innerHTML=xmlHttp.responseText;

document.getElementById("txtToUpdat
e").focus();
  }
}
...
</script>
```

### RECOMMENDATIONS

Developers should use the JavaScript focus method to set focus to the part of the page that has changed. Placing a `tabindex` of -1 on elements that do not usually receive focus such as (`div`) and (`span`) allows these elements to gain focus via the JavaScript focus method without appearing in the tab order. If focus is given to an anchor, the content of the anchor is automatically focused. Developers should be cautious when assigning focus to HTML elements that are typically not designed to receive focus. For example, focus should not be set on fieldsets or legends as unpredictable results may occur.

# RESOURCES

## SSB BART GROUP

Our ultimate goal at SSB is to create a world where digital systems can be made readily accessible to users with disabilities—enabling digital technology to become a profound empowering force in their lives. See more information regarding our services, free webinars, and blog posts on the SSB BART Group Website.

## SSB BART GROUP LABS

SSB BART Group Labs provides working, functional examples of a variety of accessibility concepts including forms, ARIA, dialogs, and many more. The examples are used to demonstrate the various techniques available to produce accessible web content.

## ARIA

Accessible Rich Internet Applications (ARIA) provides accessibility information, such as name, state, role, and value of an element in order to assist assistive technology users interact with digital content. The ARIA 1.1 Specification provides information on ARIA roles, states, and properties for widgets.

SSB BART Group also provides guidance on using ARIA on the web.

## W3C WEB ACCESSIBILITY INITIATIVE (WAI)

The Web Accessibility Initiative (WAI) develops strategies, guidelines, and resources to help make the Web accessible to people with disabilities.