



Accessible Web Mapping Apps

ARIA, WCAG and 508 Compliance

Kelly Hutchins

Tao Zhang

Agenda

- Introduction
- Section 508 and WCAG
- Knowledge and Techniques
 - Focus
 - Semantic HTML
 - WAI-ARIA and accessible components
- Demo
- Automated Testing
- Resources

Diversity of Users

- About 15% of world population lives with some form of disability
- 1 Billion People

Forms of Disability



Visual



Motor



Auditory



Cognitive

Visual

A broad range from **no vision** (total blindness) to **limited or low vision**

Motor

Users may prefer not to use a mouse, have **RSI** (Repetitive Strain Injury), or physical **paralysis** and **limited range of motion**

Auditory

Users may be **completely deaf or hard of hearing**

Cognitive

A broad range including:

- Learning disabilities
- Reading disorders(**dyslexia**)
- Attention deficit disorders(**ADHD** and **autism**)

Far more users with cognitive disabilities than all the other types of disabilities combined

Benefits of Accessibility

- Accessible interfaces is about good design and coding practice
- Good accessibility is good user experience
- Accessibility will enhance design, not destroy it

Section 508 and WCAG

Section 508

- The Rehabilitation Act of 1973
- Mandates that people with disabilities have **same access to and use** of ICT (Information and Communication Technology) comparable to those without disabilities
- Products procured by government agencies must pass Section 508 requirements
- Recent refresh incorporates WCAG 2.0 Level A and AA success criteria
 - Published: Jan. 18, 2017
 - Enforcement: Jan. 18, 2018

Overview of WCAG 2.0

Principles	Success Criteria	Level A	Level AA	Level AAA
1. Perceivable	1.1 Text Alternatives	1.1.1		
	1.2 Time-based Media	1.2.1 – 1.2.3	1.2.4 – 1.2.5	1.2.6 – 1.2.9
	1.3 Adaptable	1.3.1 – 1.3.3		
	1.4 Distinguishable	1.4.1 – 1.4.2	1.4.3 – 1.4.5	1.4.6 – 1.4.9
2. Operable	2.1 Keyboard Accessible	2.1.1 – 2.1.2		2.1.3
	2.2 Enough Time	2.2.1 – 2.2.2		2.2.3 – 2.2.5
	2.3 Seizures	2.3.1		2.3.2
	2.4 Navigable	2.4.1 – 2.4.4	2.4.5 – 2.4.7	2.4.8 – 2.4.10
3. Understandable	3.1 Readable	3.1.1	3.1.2	3.1.3 – 3.1.6
	3.2 Predictable	3.2.1 – 3.2.2	3.2.3 – 3.2.4	3.2.5
	3.3 Input Assistance	3.3.1 – 3.3.2	3.3.3 – 3.3.4	3.3.5 – 3.3.6
4. Robust	4.1 Compatible	4.1.1 – 4.1.2		

Level of Conformance

- **Level A:** Sets a minimum level of accessibility and does not achieve broad accessibility for many situations.
- **Level AA:** Generally recommended for web-based information.
- **Level AAA:** W3C does not recommend be required as general policy because it is not possible to satisfy all Level AAA Success Criteria for some content.

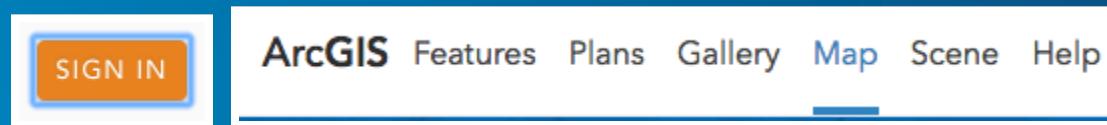
Knowledge and Techniques

- Focus
- Semantic HTML
- WAI-ARIA and accessible components

Focus

Focus Introduction

- **Focus:** Which control on the screen currently receives input from keyboard.
- Focus ring: visual focus indicator, style depending on browser and page style.



- **Tab order:** The order in which focus proceeds forward and backward through interactive elements via Tab key.

Focusable elements

- Native interactive HTML elements are focusable:
 - Text fields, Buttons, Links, Select lists ...
- (Normally) not focusable:
 - `<p>`, `<div>`, ``, `<h1>` ...
- Only focus elements that keyboard users need to interact with
- Screen reader users have ways to read focusable and non-focusable elements. (demo)

Tab order matters

WCAG 1.3.2: Reading and navigation order, as determined by DOM structure, should be logical and intuitive.

- Be careful changing visual position of elements on screen using CSS
- Avoid jumping around tab order

Offscreen elements

- Example: Calcite drawer pattern
- Prevent element from gaining focus when off screen

```
display:none;  
visibility:hidden; /* alternative */
```

- Only allow it to be focused when user can interact with it

```
display:block;  
visibility:visible; /* alternative */
```

Test focus

- Tab through page to see tab order doesn't disappear or jump out of logical sequence
- Make sure to hide offscreen content
- Rearrange elements' position in the DOM if necessary

Manage focus

- `tabindex="0"`: let natural DOM structure determine tab order
- `tabindex="-1"`: programmatically move focus (e.g., error message, menus, radio buttons, etc.)
- `tabindex="5"`: anti-pattern

Focus management example

Customized menu

```
<menu-list>
  <!-- After the user presses the down arrow key,
  focus the next available child -->
  <menu-item tabindex="0">Map</menu-item>

  <!-- call .focus() on this element -->
  <menu-item tabindex="-1">Layer</menu-item>

  <menu-item tabindex="-1">Scene</menu-item>
  <menu-item tabindex="-1">Tool</menu-item>
  <menu-item tabindex="-1">Data</menu-item>
</menu-list>
```

Example code

Keyboard traps

- Keyboard focus should not be locked or trapped at one particular element.
- **Temporary** keyboard trap is necessary for modal dialogs:
 - When modal is displayed: trap focus inside modal.
 - When modal is closed: restore focus to previously focused item.
 - Demo
 - Example code

Semantic HTML

Accessibility tree

Browser's responsibility to expose accessibility tree to assistive technologies.

Microsoft Edge's accessibility tree view

The screenshot shows the Microsoft Edge DevTools interface with the "F12" tab selected. The top navigation bar includes "DOM Explorer" (selected), "Console", "Debugger", "Network", "Performance", "Memory", "Emulation", and "Experiments". Below the navigation bar are four icons: a magnifying glass, a square, a circular arrow, and a person icon.

The left pane displays the DOM structure:

```
> <div class="span12">...</div>
</div>
▲ <div class="row">
  ▲ <div class="span12 hero-unit">
    <div class="divWithClick" id="clickMe">Click Me!</div>
    <div class="divWithClick" id="clickMe2" role="button">Click Me, with focus!
      Me, with focus!</div>
    <div id="out">Thing 2 was clicked!</div>
    <p></p>
  </div>
</div>
> <div class="row"> </div>
```

The element `<div class="divWithClick" id="clickMe2" role="button">Click Me, with focus!` is highlighted with a red border. The bottom navigation bar shows the current path: body > div.container > div.row > div.span12 > div#clickMe2.

The right pane shows the "Accessibility tree" with the following structure:

- application: Accessibility Demos -
- group: Click Me!
- button: Click Me, with focus! (highlighted)
- list:
- list:
- link: Demo Index

Chrome Canary's accessibility tree view

Etsy

Sell on Etsy Register Sign in Cart

Clothing & Accessories Jewelry Craft Supplies & Tools Weddings Entertainment Home & Living Kids & Baby Vintage

Books, Movies & Music >	Books	Music
Electronics & Accessories >	Art & Photography Books	Instrument Straps
	Blank Books	Musical Instruments
	Book Accessories	Recorded Audio
	Calendars & Planners	Sheet Music
	Children's Books	All Books, Movies & Music
	Comics & Graphic Novels	
	Craft & Hobby Books	
	Guides & How Tos	
	Literature & Fiction	
	Poetry	
	Zines & Magazines	
	Movies	



EDITORS' PICKS
Wearable tech & accessories

Elements Console Sources Network Timeline Profiles API

html body #cnav-header #cnav-header-inner nav ul#catnav-menubar.catnav-primary

Styles Event Listeners DOM Breakpoints Properties Accessibility

▼ Accessibility Tree

▼ WebArea "Etsy.com | Shop for anything from creative people everywhere"

▼ Div

▼ Div

▼ navigation

▼ menubar

- menutem "Clothing & Accessories"
- menutem "Jewelry"
- menutem "Craft Supplies & Tools"
- menutem "Weddings"
- menutem "Entertainment"
- menutem "Home & Living"
- menutem "Kids & Baby"
- menutem "Vintage"

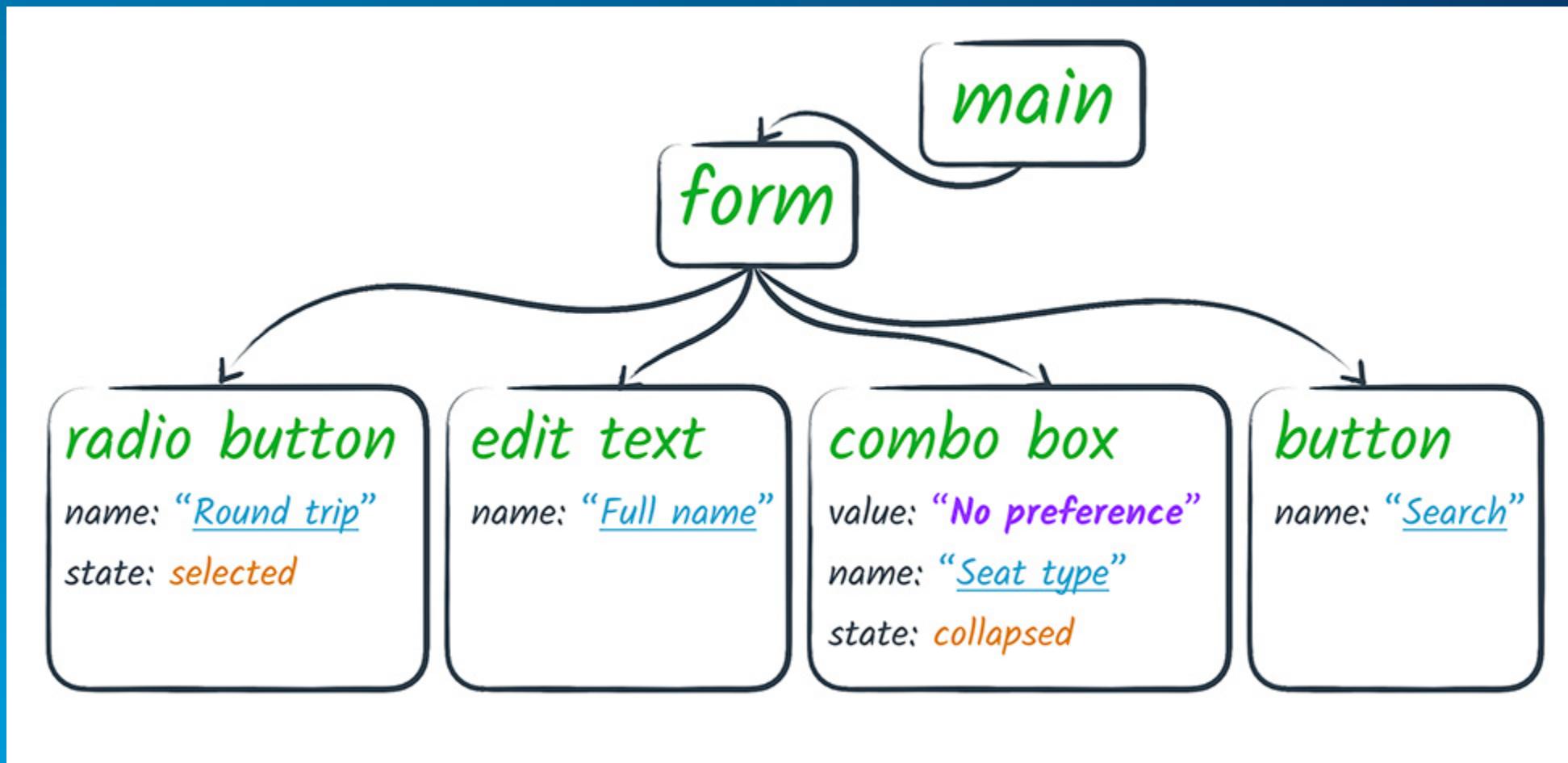
▼ ARIA Attributes

role: menubar
aria-activedescendant: catnav-primary-link-2938-link

▼ Computed Properties

▼ Name: ""
aria-labelledby: Not specified
aria-label: Not specified
title: Not specified
Role: menubar
Active descendant: a#catnav-primary-link-2938-link.text-gray

Shows how website is interpreted by assistive technologies and how accessible data are provided.



- Assistive technologies simulate and relay user interactions like click and key press to accessibility tree.
- As developers, we need to:
 - Express the semantics of page correctly.
 - Specify accessible names and descriptions.
 - Make sure important elements have correct accessible roles, states, and properties.

Semantics in native HTML

- Most HTML elements have implicit semantics (**role** and **state**).
- Native HTML elements work predictably across browsers
 - Take advantage of this!

Example:

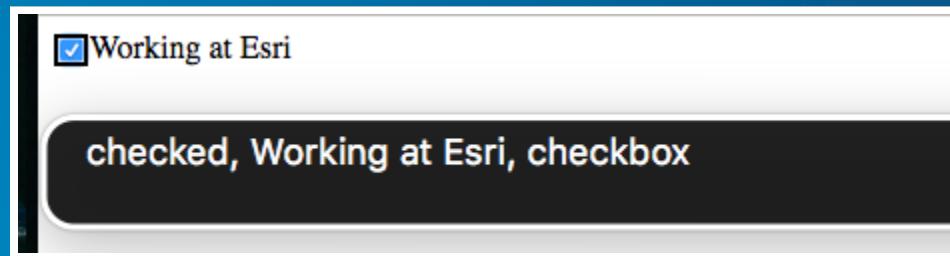
```
<a href="http://www.esri.com">Esri Homepage</a>
```

- Role="link"
- Accessible name="Esri Homepage"
- State="focusable"

Example:

```
<label><input type="checkbox" checked>Working at Esri
```

- Role="checkbox"
- Accessible name="Working at Esri"
- State="focusable checked"



Keyboard

- Native interactive HTML elements receive keyboard focus:
 - <a>, <button>, <input>...
- Interactive elements have expected interactions:
 - Link: click, tap, or Enter key
 - Button: click, tap, Enter key, or **Space** key
 - Input: click, tap, or Enter key

Neutral semantics

- Some HTML elements do not convey semantics (role or state):
 - `<div>This is a block area</div>`
 - `This is an inline area`

If the element is interactive, we need to do extra work:

- Make it **focusable**: `tabindex="0"`
- Receive **keyboard events**: Enter, Space
- **Name**: explicit label (`label`) or implicit text (`aria-label`, `aria-labelledby`)
- **Role**
- **States and properties**

WAI-ARIA

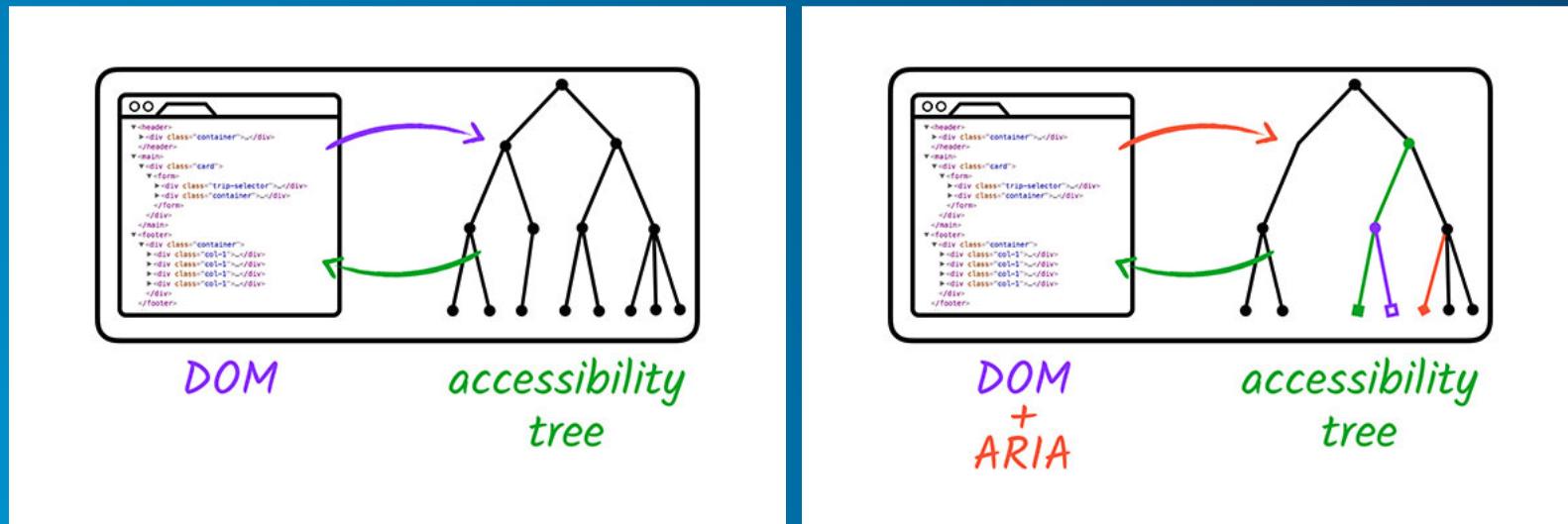
Web Accessibility Initiative – Accessible Rich Internet Applications

Why need ARIA

- Use Native HTML semantics whenever possible
- Certain semantics and design patterns make it impossible to use native HTML semantics.
 - Example: a pop-up menu, no standard HTML element
 - Example: a semantic characteristic "the user needs to know about this as soon as possible"

What is WAI-ARIA

- Specification for increasing accessibility of custom elements
 - Allows developers to modify and augment accessibility tree from standard DOM



ARIA doesn't augment any of the element's inherent behavior:

- Focusable
- Keyboard event listeners

Custom behaviors still need to be implemented

ARIA attributes

Type	Purpose	Examples
Roles	Meaning of an element	tooltip, tablist, search
Properties	Relationships and functions	aria-required, aria-controls, aria-label, aria-labelledby
States	Current interaction states	aria-checked, aria-expanded, aria-hidden

An ARIA example

```
<li tabindex="0" class="checkbox" checked>  
    Show premium content  
</li>
```

- Sighted users see a checkbox as a result of CSS `class="checkbox"`.
- Screen reader users will not know this is meant to be a checkbox.

An ARIA example

```
<li tabindex="0" class="checkbox" role="checkbox"  
aria-checked="true">  
    Show premium content  
</li>
```

Screen reader will report this as a checkbox.

Roles

- Landmarks
 - **banner**: The main header of a page; typically assigned to a header element.
 - **contentinfo**: A collection of metadata, copyright information and the like.
 - **main**: the main content of a document.
 - **navigation**: A collection of links for navigation.

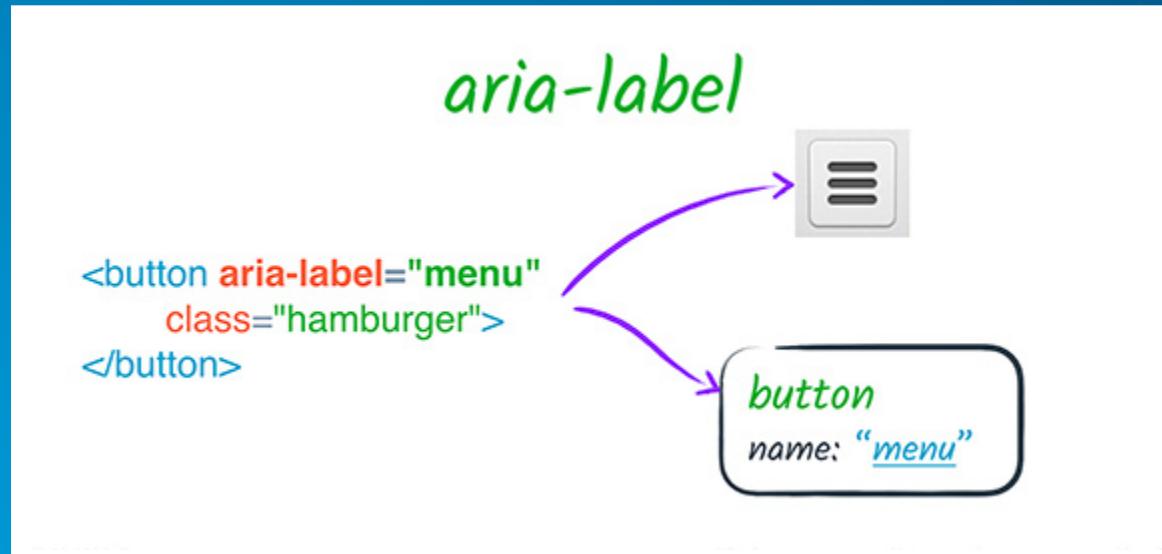
Demo

Roles

- Widgets
 - alert
 - dialog
 - data grid
 - tab
 - tablist
 - tabpanel

Properties - Labels

- aria-label
- Specifies a string as accessible label
- Overrides native labeling



Properties - Labels

- aria-labelledby
 - Specifies id of another DOM element (or a list of id)
 - Overrides all other name sources
 - Applicable to any element, not just labelable elements
 - Can specify visually hidden elements

aria-labelledby

Drink Options

`
Drink options

<div role="radiogroup"
aria-labelledby="rg-label">
...
</div>`

Water
Tea
Coffee
Cola
Ginger Ale

The diagram illustrates the mapping of an HTML element to its ARIA state. On the left, an HTML snippet is shown with a `span` element containing the text "Drink options" and a `div` element with the attribute `aria-labelledby="rg-label"`. Two purple arrows point from this `div` element to a callout box on the right. The callout box contains the text "radio group" and "name: Drink options". Above the callout box, the text "Drink Options" is followed by a list of five radio buttons: Water, Tea, Coffee, Cola, and Ginger Ale.

Relationships

- aria-owns
 - Indicates an element should be treated as parent of another separate DOM element

`<div role="menu">
 <div role="menuitem"
 aria-haspopup="true">
 New
 </div>
 <div aria-owns="submenu">
 </div>
 <!-- more items... -->
 </div> <!-- menu -->
 <div role="menu" id="submenu">
 <div role="menuitem">
 Document
 </div>
 <!-- more items... -->
 </div> <!-- submenu -->`

aria-owns

The diagram shows a main menu item labeled "New" with a submenu arrow pointing to a list of document types: Document, Spreadsheet, Presentation, Make a copy, and Form. The word "aria-owns" is written above the main menu item, with a purple arrow pointing from the text to the submenu arrow.

New	>	Document
		Spreadsheet
		Presentation
		Make a copy
		Form
		Organize

Relationships

- **aria-describedby**
 - Provides accessible description for an element
 - References elements in the DOM separated from current element

aria-describedby

```
<label for="pw">Password:</label>
<input type="password" id="pw"
       aria-describedby="pw-help">
<div id="pw-help">
  Password must be at least 12 characters
</div>
```

A diagram illustrating the `aria-describedby` relationship. On the left, there is a snippet of HTML code. In the middle, there is a visual representation of the UI. A purple arrow points from the `aria-describedby="pw-help"` attribute in the HTML to the `<div id="pw-help">` element in the UI. The UI consists of a label "Password:", an input field (a blue-bordered rectangle), and a descriptive message "Password must be at least 12 characters" located below the input field.

Password:

Password must be at least 12 characters

Relationships

- aria-controls
 - Indicates an element "controls" another element in interaction

```
<div role="scrollbar" aria-controls="main"></div>
<div id="main">
  . . .
</div>
```

Hide elements

- Elements explicitly hidden from the DOM will not be included in accessibility tree

```
[hidden] {  
  display: none; /*not rendered, no space allocated */  
}  
[invisible] {  
  visibility: hidden; /*rendered, space allocated*/  
}
```

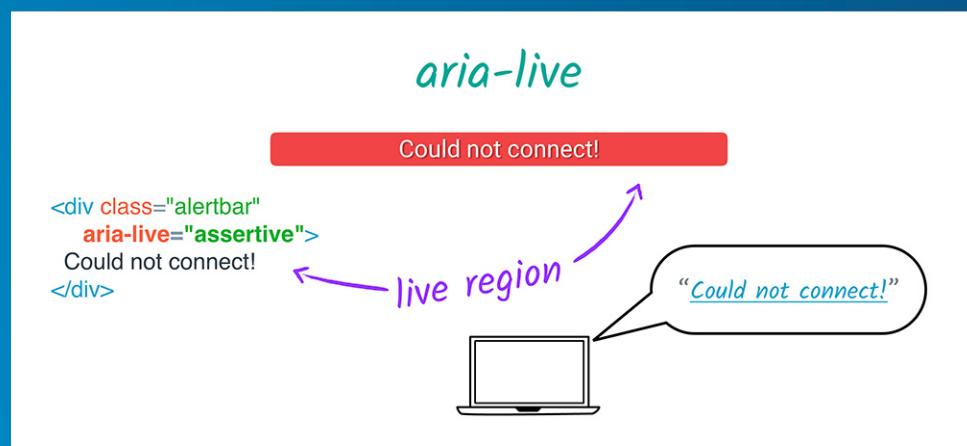
- Elements not visually rendered but not explicitly hidden is still included in accessibility tree.

```
/* Screen reader only*/
.sr-only {
    position: absolute;
    left: -10000px;
    width: 1px;
    height: 1px;
    overflow: hidden;
}
```

- aria-hidden
 - Excludes content from assistive technology that is not visually hidden.
 - Removes current element and all of its descendants from the accessibility tree.
 - Demo

Update elements

- `role="alert"`
- `aria-live`
 - Marks element as "live region" in which updates should be communicated to users immediately.
 - `aria-live="polite"`: alert user when screen reader has finished current action
 - `aria-live="assertive"`: interrupt current action and alert user immediately



ARIA best practices

1. Do not change native semantics, unless you *really* have to.

- Example: A developer wants to implement a heading which is also a button.
- Don't do this:

```
<h2 role="button">heading button</h2>
```

- Do this:

```
<h2><button>heading button</button></h2>
```

2. All interactive ARIA elements must be usable with keyboard.

- The elements should respond to standard key strokes.
 - Example: If using `role="button"`, add `tabindex="0"` and support Enter and Space actions.
- The user must be able to navigate and perform actions using keyboard.
 - Example: If allowing clicking through data grid, support navigating grid cells using keyboard.

3. Do not use `role="presentation"` or `aria-hidden="true"` on a visible and focusable element.

- This will result in focusing on "nothing".
- Don't do these:

```
<button role="presentation">Press me</button>
<button aria-hidden="true">Press me</button>
```

4. All interactive elements must have an accessible label or name.

Do this:

```
<label>
    Email
    <input type="text" placeholder="name@example.com">
</label>
```

Create Accessible Web Components

TABLE OF CONTENTS

1. **Introduction**
2. **Design Patterns and Widgets**
 - 2.1 Generally Applicable Keyboard Recommendations
 - 2.2 Accordion
 - 2.3 Alert
 - 2.4 Alert Dialog or Message Dialog
 - 2.5 Auto Complete
 - 2.6 Button
 - 2.7 Checkbox
 - 2.8 Combo Box
 - 2.9 Date Picker
 - 2.10 Dialog (Modal)
 - 2.11 Dialog (Non-Modal)
 - 2.12 Dialog (Tooltip)
 - 2.13 Grid (Simple Data Tables)
 - 2.14 Actionable, Sortable Column Header in a Grid
 - 2.15 Landmark Navigation
 - 2.16 Link
 - 2.17 Listbox
 - 2.18 Media Player
 - 2.19 Menu or Menu bar
 - 2.20 Menu Button
 - 2.21 Popup Help (aka Bubble Help)



ARIA Design Patterns

WAI-ARIA Authoring Practices 1.1

W3C Working Draft 17 March 2016



This version:

<http://www.w3.org/TR/2016/WD-wai-aria-practices-1.1-20160317/>

Latest published version:

<http://www.w3.org/TR/wai-aria-practices-1.1/>

Latest editor's draft:

<http://w3c.github.io/aria/practices/aria-practices.html>

Previous version:

<http://www.w3.org/TR/2015/WD-wai-aria-practices-1.1-20151119/>

Editors:

Matt King, Facebook, mck@fb.com

James Nurthen, Oracle Corporation, james.nurthen@oracle.com

Michael Cooper, W3C, cooper@w3.org

Michiel Bijl, The Paciello Group, mbijl@paciellogroup.com

Joseph Scheuhammer, Inclusive Design Research Centre, OCAD University (Previous Editor)

Lisa Pappas, SAS (Previous Editor)

Rich Schwerdtfeger, IBM Corporation (Previous Editor)

Copyright © 2015-2016 W3C® ([MIT](#), [ERCIM](#), [Keio](#), [Beihang](#)). W3C [liability](#), [trademark](#) and [document use](#) rules apply.

Abstract

This document provides readers with an understanding of how to use [WAI-ARIA 1.1](#) [[WAI-ARIA](#)] to create accessible rich internet applications. It describes considerations that might not be evident to most authors from the WAI-ARIA specification alone and recommends approaches to make widgets, navigation, and behaviors accessible using WAI-ARIA roles, states, and properties. This document is directed primarily to Web application developers, but the guidance is also useful for user agent and assistive technology developers.

This document is part of the WAI-ARIA suite described in the [WAI-ARIA Overview](#).

Accessible Map

- For low-vision users:
 - Color contrast
 - Color blindness
 - Scaling and images of text
- For non-sighted users:
 - Alternative text for map's core information
 - Accessibility in Google Maps

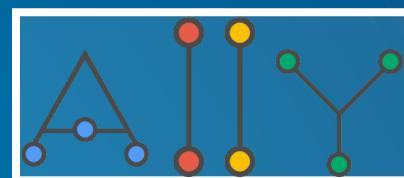
Demo

Common Accessibility Issues:

- Text alternatives
- Semantic HTML
- Tab order and focus
- Color
- Label

Automated Testing

A11Y command-line tool



by Addy Osmani

```
npm install -g ally  
ally www.esri.com > audit.txt
```

A11Y module usage

`a11y(URL, callback)` accepts a string as input and takes a callback providing a `reports` object with the accessibility audit for the supplied URL.

```
var ally = require('ally');
ally('esri.com', function (err, reports) {
    var output = JSON.parse(reports);
    var audit = output.audit; //ally formatted
    var report = output.report;
    //Chrome devtools accessibility audit form
    reports.audit.forEach(function (el) {
        // result will be PASS, FAIL or NA
        if (el.result === 'FAIL') {
            // el.heading
            // el.severity
            // el.elements
        }
    })
})
```

axe-core

Accessibility engine for automated Web UI testing by dequelabs:

```
npm install axe-core --save-dev
```

```
<script src="node_modules/axe-core/axe.min.js"></script>
<!-- Normal page content ... -->
<script>
    axe.run(function (err, results) {
        if (err) throw err;
        ok(results.violations.length === 0,
           'Should be no accessibility violations');
        // complete the async call
    });
</script>
```

Resources

- Documentation
 - W3C-WCAG 2.0
 - Interpretation of success criteria
 - WAI-ARIA Authoring Practices 1.1
- Courses
 - egghead.io: Start Building Accessible Web Applications Today
 - Udacity: Web Accessibility by Google
- a11y
- axe-core
- aXe Chrome extension
- Chrome Accessibility Developer Tools
- Some of the diagrams are adapted from Google Developers: Web Fundamentals