**How to Use ARIA (Accessible Rich Internet Applications) in HTML and CSS**

**BY**[**FATIH KÜÇÜKKARAKURT**](https://www.makeuseof.com/author/fatih-kucukkarakurt/)

PUBLISHED 10 HOURS AGO

Small tweaks to your HTML can bring large benefits to your readers.

*Readers like you help support MUO. When you make a purchase using links on our site, we may earn an affiliate commission.*[***Read More.***](https://www.makeuseof.com/page/terms-of-use/)

**KEY TAKEAWAYS**

* **ARIA (Accessible Rich Internet Applications) is a toolkit within HTML and CSS that improves web accessibility for people with disabilities.**
* **By integrating ARIA roles, states, and properties into HTML, you can enhance the user experience and make web content more inclusive.**
* **ARIA attributes like aria-label and aria-describedby provide additional context and substitute text for elements, ensuring screen reader users receive the same level of information as sighted users.**

Advertisement: 0:10

Skip Ad

In the realm of web development, it’s vital to ensure your work is accessible to all users. ARIA (Accessible Rich Internet Applications) offers a comprehensive toolkit within HTML and CSS to make web interfaces more inclusive for people with disabilities.

Take a look at how you can use ARIA to create web content that all users can easily navigate and understand.

**Implementing ARIA in HTML**

ARIA is a standard you can use to make web applications and content more accessible. When you integrate ARIA into an HTML document, it helps improve accessibility for people with disabilities.

This is particularly important for dynamic and interactive web content, as these types of content might not be adequately described using traditional HTML elements alone. By adding ARIA roles, states, and properties, you can enhance the user experience and make the web accessible to a wider audience.

**Introduction to ARIA Roles and How to Apply Them to HTML Elements**

ARIA introduces a range of roles, which define the functionality and purpose of various elements on a webpage. These roles go beyond the traditional HTML elements and provide enhanced [semantics for assistive technologies](https://www.makeuseof.com/semantic-html-tags-to-make-website-more-accessible/).

For instance, by using the **role** attribute, you can designate an element as a **button**, a **link**, or even a **navigation** landmark. Take a look at some examples:

<**button** role="button">Click Me</**button**>  
  
<**a** role="link" href="#">Visit our website</**a**>  
  
<**nav** role="navigation">  
  <**ul**>  
    <**li**><**a** href="#">Home</**a**></**li**>  
    <**li**><**a** href="#">About</**a**></**li**>  
    <**li**><**a** href="#">Contact</**a**></**li**>  
  </**ul**>  
</**nav**>

**Applying ARIA Labels and Descriptions for Screen Readers**

Screen readers rely on text-based information to convey the content and functionality of elements to visually impaired users.

ARIA offers attributes like aria-label and aria-describedby to provide additional context or substitute text for elements that may not have sufficient visible content. This ensures that screen reader users receive the same level of information as sighted users:

<**button** aria-label="Close" class="close-button">&times;</**button**>  
  
<**img** src="image.jpg" alt="A beautiful sunset"  
  aria-describedby="image-description">  
  
<**p** id="image-description">A serene sunset over the ocean, painting the sky  
with warm hues.</**p**>

**ARIA in CSS**

ARIA attributes often interact with CSS to provide visual cues for various states of elements. By coupling ARIA roles with corresponding CSS classes, you can achieve a more coherent and accessible interface. Consider this example of a button that uses the **aria-pressed** attribute:

<**button** role="button" class="interactive-button" aria-pressed="false">  
  Click Me  
</**button**>

Using this CSS, you can style the button according to the state of this attribute, changing the button's appearance when it's pressed:

.interactive-button[aria-pressed="true"] {  
  **background-color**: #e74c3c;  
}

**Styling ARIA Landmarks for Improved Visual Representation**

ARIA bookmarks aid in both navigation and understanding by helping to divide a webpage into meaningful sections. You can shape these bookmarks to provide clearer visual separation and improve the user experience. Here’s some sample markup for a basic structure:

<**header** role="banner" class="page-header">  
    <**h1**>My Website</**h1**>  
</**header**>  
  
<**main** role="main" class="content">  
    <**section** role="region" class="section">  
        <**h2**>Styling ARIA Landmarks for Improved Visual Representation</**h2**>  
  
        <**p**>ARIA landmarks help organize a webpage into sections that have  
        a specific purpose. These landmarks improve both accessibility and  
        user experience. You can style ARIA landmarks to create a visually  
        pleasing layout.</**p**>  
          
        <**p**>In this example, we'll style ARIA landmarks using CSS:</**p**>  
    </**section**>  
</**main**>

Which you can style as follows:

[role="banner"] {  
  **background-color**: #333;  
  **color**: #fff;  
  **padding**: 10**px**;  
}  
  
[role="main"] {  
  **background-color**: #f5f5f5;  
  **padding**: 20**px**;  
}  
  
[role="region"] {  
  **border**: 1**px** **solid** #ddd;  
  **padding**: 10**px**;  
  **background-color**: #fff;  
}

**Using ARIA-Related CSS Properties to Customize Accessibility Features**

CSS also allows you to enhance the visibility of elements when they receive focus, ensuring keyboard navigation remains user-friendly. By leveraging ARIA-related CSS properties, you can achieve this effect:

/\* Apply styles when an element receives focus \*/  
:focus {  
  **outline**: 2**px** **solid** #007bff;  
}

**Testing and Validating ARIA Implementation**

For effective ARIA implementation, thorough testing is crucial. You can use various tools to simulate interactions with assistive technologies. This testing helps identify accessibility issues and improves ARIA for inclusivity.

Diverse specialized tools can facilitate this testing, emulating how users with disabilities interact with content. Identifying problems like incorrect ARIA attributes or missing roles allows proactive issue resolution.

Modern web browsers include developer tools to inspect ARIA attributes and states. This helps ensure proper implementation aligning with accessibility guidelines. The real-time evaluation identifies potential problems and refines ARIA for inclusivity.

**Common ARIA Patterns and Best Practices**

You can examine various ARIA patterns (such as **role**, **state**, and **property**) and what you should pay attention to when using these patterns as follows. In this way, you can make websites and applications more accessible and user-friendly.

**Creating Accessible Navigation Menus and Focus Management**

Incorporating ARIA attributes into navigation menus can significantly enhance the user experience, particularly for those relying on screen readers. These attributes play a crucial role in making keyboard navigation fluid and comprehensible. By employing the role attribute, alongside the power of JavaScript, you have the opportunity to craft dynamic menus that seamlessly adjust based on user interactions.

For example, [consider a navigation menu](https://www.makeuseof.com/responsive-navigation-bar-using-html-and-css/) that expands and collapses sub-menus when a user interacts with it. You can use the ARIA roles **menu**, **menuitem**, and **menuitemcheckbox** to create a menu structure that's accessible to screen readers. Here's a simplified HTML and JavaScript snippet to illustrate this concept:

<**nav** role="menu">  
  <**button** role="menuitem" aria-haspopup="true" aria-expanded="false"  
    id="menuButton">Menu</**button**>  
  
  <**ul** role="menu" aria-labelledby="menuButton">  
    <**li** role="menuitem">  
      <**a** href="#">Home</**a**>  
    </**li**>  
  
    <**li** role="menuitem" aria-haspopup="true" aria-expanded="false">  
      Services  
  
      <**ul** role="menu" aria-label="Services Submenu">  
        <**li** role="menuitemcheckbox">  
          <**input** type="checkbox" id="service1" />  
          <**label** for="service1">Service 1</**label**>  
        </**li**>  
  
        <**li** role="menuitemcheckbox">  
          <**input** type="checkbox" id="service2" />  
          <**label** for="service2">Service 2</**label**>  
        </**li**>  
      </**ul**>  
    </**li**>  
  
    <**li** role="menuitem">  
      <**a** href="#">Contact</**a**>  
    </**li**>  
  </**ul**>  
</**nav**>  
  
<**script**>  
  const menuButton = document.getElementById('menuButton');  
  const subMenu = menuButton.nextElementSibling;  
  
  menuButton.addEventListener('click', () => {  
    const expanded = menuButton.getAttribute('aria-expanded') === 'true';  
    menuButton.setAttribute('aria-expanded', !expanded);  
    subMenu.hidden = !subMenu.hidden;  
  });  
</**script**>

**Handling Dynamic Content and ARIA Live Regions**

Creating dynamic content that updates seamlessly without requiring a full page reload can present accessibility challenges. Such content updates might not be immediately apparent to users who rely on screen readers.

To address this, you can use the **aria-live** attribute to establish parts of the webpage as live regions. These live regions, when appropriately implemented, dynamically announce changes to the screen reader user, ensuring they receive real-time information without the need for manual refreshes.

Consider a real-time commenting feature on a social media platform. Here's an example of how you might implement the aria-live attribute in HTML:

<**div** aria-live="polite" class="live-region">  
  New comment: MUO: "Just posted an amazing photo from my trip!"  
</**div**>

In this example, the aria-live attribute is set to **polite**, indicating that the screen reader should announce the content update when the user is idle. The live-region class defines the area as a live region. As new comments are posted, this live region will automatically notify screen reader users about the new content, providing them with an accessible and updated experience.

**Enhancing Form Accessibility With ARIA Attributes and Validation**

Forms are a very important part of web interaction, and you can use ARIA attributes to provide better instructions, error messages, and tips to users who fill out forms:

<**form**>  
  <**label** for="name">Name:</**label**>  
  
  <**input** type="text" id="name" aria-required="true"  
    aria-label="Enter your full name" />  
  
  <**label** for="email">Email:</**label**>  
  
  <**input** type="email" id="email" aria-required="true"  
    aria-label="Enter your email address" />  
  
  <**label** for="affiliation">Affiliation:</**label**>  
  
  <**input** type="text" id="affiliation"  
    aria-label="Enter your affiliation, if any" />  
  
  <**button** type="submit">Register</**button**>  
</**form**>

By thoughtfully integrating ARIA attributes into forms, you can create a more inclusive digital environment. This can empower all users to engage with web content seamlessly, fostering an improved and accommodating user experience.

**Implementing Accessible Modals and Dialogs**

[Modals and dialogs](https://www.makeuseof.com/build-modals-easily-with-the-html-dialog-element/) are common UI elements, but they can pose accessibility challenges. ARIA attributes like **aria-modal** and role attributes help make these components more user-friendly:

<**div** role="dialog" aria-modal="true" aria-labelledby="modal-title"  
  aria-describedby="modal-description"  
>  
  <**h2** id="modal-title">Sign Up</**h2**>  
  
  <**p** id="modal-description">Please fill in the form below to create an  
  account.</**p**>  
  
  <**form**>  
    <!-- Form fields and controls go here -->  
    <**button** type="submit">Sign Up</**button**>  
    <**button** type="button" onclick="closeModal()">Cancel</**button**>  
  </**form**>  
</**div**>

The **aria-labelledby** attribute associates the modal title with its content, enhancing screen reader comprehension, and the **aria-describedby** attribute does the same for a brief description. These ARIA attributes, combined with proper semantic HTML, contribute to a more inclusive and accessible modal or dialogue experience for all users.

**Considerations and Limitations of ARIA**

ARIA offers crucial capabilities, yet native HTML elements often possess inherent accessibility features that take precedence. ARIA becomes necessary when these innate features fall short of desired accessibility standards.

While ARIA enhances web content accessibility, its use may impact performance due to added code and interactions. To mitigate potential bottlenecks, carefully test and optimize your ARIA implementation. This ensures speed and compatibility across devices and browsers.

Working with ARIA demands a comprehensive grasp of its intricacies to avoid confusion or accessibility issues. Adhering to best practices and guidelines is vital to navigating ARIA complexities. Staying informed about industry standards ensures broad accessibility and avoids unnecessary challenges.

**Keeping Up With ARIA Updates and Standards**

The accessibility landscape is ever-evolving with continuous advancements. Staying updated on recent ARIA specifications is crucial for web content accessibility.

Try joining online communities that offer tailored solutions for ARIA challenges. These spaces offer practical strategies and expert guidance to enhance your digital inclusivity. By tapping into collective knowledge, you can enrich your understanding of how to improve user experiences.

[**What Is Google's AI Project Gemini and What Is It Working On?**](https://www.makeuseof.com/what-is-googles-ai-project-gemini/)

2 days ago

[**How to Create User Personas in ChatGPT for Better Results**](https://www.makeuseof.com/create-chatgpt-user-personas/)

2 days ago

[**7 Amazing ChatGPT Chrome Extensions for Better AI Prompts and Answers in Browsers**](https://www.makeuseof.com/chatgpt-chrome-extensions-better-ai-prompts-answers-in-browsers/)

2 days ago

[**See More**](https://www.makeuseof.com/tag/artificial-intelligence/)

**TRENDING NOW**

[**How to Install and Use CodeGPT in VS Code**](https://www.makeuseof.com/install-use-codegpt-in-vs-code/)

[**How to Build a Video Media Player Using Python**](https://www.makeuseof.com/python-video-media-player-how-to-build/)

[**How to Manage State in Astro Applications**](https://www.makeuseof.com/astro-state-manage-applications-how-to/)