Engineer

ARIA

extra markup that names a control for assistive tech

What it is

ARIA is a W3C specification of HTML attributes that tells assistive technology what a page element is, what it is called and what state it is in, where the HTML alone says too little. The letters stand for Accessible Rich Internet Applications, and the specification's full title, WAI-ARIA, adds the W3C's Web Accessibility Initiative. The current Recommendation is version 1.2.

A `role` attribute says what kind of thing an element is: a widget such as `button` or `tab`, a structural part such as `list`, one of the eight landmark roles such as `navigation` or `main`, or a live region such as `status`, whose changes a screen reader announces unasked. The `aria-*` attributes then split into properties, like `aria-labelledby`, and states, like `aria-expanded`, which change as the user acts. The browser maps all of it into the operating system's accessibility API, which is what a screen reader or a voice-control program reads.

ARIA changes what is announced and nothing else. A `div` with `role="button"` is read out as a button, yet it takes no focus and ignores Enter and Space until a `tabindex` and a script supply both. Hence what the W3C's guide Using ARIA calls the first rule of ARIA use: when a native HTML element or attribute already has the semantics and behaviour you need, use it instead of repurposing another element and adding ARIA.

```
<!-- announced as a button, behaves like a div -->
<div role="button">Send</div>

<!-- announced, focusable and operable as a button -->
<button type="button">Send</button>
```

The other half is the accessible name, the words a screen reader speaks for a control. The Accessible Name and Description Computation 1.2 fixes the order: `aria-labelledby` first, then `aria-label`, then the native label (a `label` element, an `alt`, an SVG `title`), then the element's text content for roles that take their name from it, and the `title` attribute last. So an `aria-label` replaces the visible words of a button rather than adding to them.

Use the concept in a decision

Put "What is ARIA?" to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

The decision ARIA puts in front of a team is build or reuse. WCAG 2.2 success criterion 4.1.2, Name, Role, Value, asks that every user interface component expose its name and role to software, let its user-settable states and values be set programmatically, and report when they change. A native `button`, `select` or `details` meets that by default. A custom tab strip, combobox or tree view meets it only through ARIA and the script that keeps the ARIA true.

That makes a custom widget a recurring cost: keyboard handling, focus management, states that track every interaction and testing in more than one screen reader, at every redesign. Judge a component library by which pattern from the W3C's ARIA Authoring Practices Guide each widget follows, and by whether that pattern's keyboard behaviour is there when the mouse is put away.

The legal pressure now reaches private companies. The Web Accessibility Directive, (EU) 2016/2102, covers public-sector sites and apps. The European Accessibility Act, Directive (EU) 2019/882, applies from 28 June 2025 to a range of consumer products and services, e-commerce and consumer banking among them, with microenterprises that provide services exempt. Both lean on the harmonised standard EN 301 549, whose web requirements are WCAG at Level AA. Name, Role, Value sits at Level A, so a control whose name or role stays hidden from a screen reader fails the lowest level an auditor checks.

What it looks like in practice

This site's templates use ARIA sparingly. The top bar on every inner page is a `div` with `role="banner"`, not the usual `header` element. The template's comment explains why: the stylesheet styles a bare `header` as the home page's full-height hero, so the element would drag that geometry onto every inner page, while the role gives the landmark and nothing else. The role is what puts the brand mark, the link home, inside a landmark. That is the narrow case the first rule allows. The site menu goes the other way: a `details` and `summary` pair, which leaves the open state, the button semantics and `aria-expanded` to the browser. The menu's `nav` and the footer's `nav` carry different `aria-label` values, since two navigation landmarks with one name break the rule engine's uniqueness rule and the site's accessibility check gates on zero findings. The retained portfolio-filter implementation uses a `role="status"` region designed to stay silent until the reader types. That filter is currently disabled with `searchEnabled = false`; it illustrates the retained implementation, not an interaction available on the published pages.

The record of taking the site to WCAG 2.2 Level AA supplies the other half of the banner story: when the rule engine's best-practice rules were switched on, one of them failed at once, because on every inner page the brand mark belonged to no landmark at all. The same page lists the absence of testing with real assistive technology among the limits it publishes.

The write-up of eleven defects the browser called healthy shows how far markup is from the experience. All eleven lived in how the page rendered, not in what its markup declared, and markup is all ARIA touches. Eight footer links stayed reachable with Tab after a style had switched off pointer interaction with them and changed nothing else: the same shape as `aria-hidden` on a focusable element, where the screen reader loses the element and the keyboard keeps it. And only seventy of the engine's hundred and five rules were switched on, with the one that flags a missing landmark among those left off.

The last page shows the same problem on a platform where ARIA plays no part. Naming all seventeen interface icons for screen readers describes a SwiftUI application: no HTML and no `aria-` attribute anywhere. Its accessibility label is SwiftUI's own, the native counterpart of `aria-label` and not an instance of it. Controls drawn with system symbols were read out by the symbol's internal name, so the send button announced as "arrow up circle, button", and seventeen icon-only controls described their artwork instead of their function. Each got a label naming the action and tracking its state where that matters, and a linter

now requires a label within eight lines of the construction that creates an icon-only control, so an eighteenth control gets in only with a label. The page states the limit, and it holds for `aria-label` too: the check confirms that some label sits close by, while whether it is the right one is something only a person listening can judge.

Where it goes wrong

A role announces a widget the keyboard never reaches. A `div` gets `role="button"` and a click handler. The screen reader says “button”, yet Tab passes over it and Enter does nothing. The third rule of ARIA use asks that every interactive ARIA control work from the keyboard. Use the native element, or build the whole keyboard model of the APG pattern and test with the mouse unplugged.

Focus lands on something the screen reader treats as absent. `aria-hidden="true"` sits on a focusable element or its ancestor, such as an off-screen drawer. Tab still moves there, and the user hears silence. The fourth rule of ARIA use rules this out. Remove hidden content from the tab order too, with `hidden`, `display: none` or `inert`, and keep `aria-hidden` for decoration that never takes focus.

A name goes where the element accepts none. `aria-label` on a `div`, `span` or `p` is meant to explain it, but WAI-ARIA 1.2 prohibits naming the `generic` and `paragraph` roles, and screen readers ignore it or read it inconsistently. On a control the label replaces the visible words, and when the two differ, a voice-control user who says what is on screen misses. Name elements that can carry a name, and start any `aria-label` with the visible words, as WCAG 2.5.3, Label in Name, asks.

Two roles contradict each other. An explicit role replaces the element’s own. A link given `role="button"` announces as a button but scrolls the page on Space; `role="tab"` on an `h2` drops it from the heading list; `role="navigation"` on a `ul` loses the list and its item count. The second rule of ARIA use is to change native semantics only when you must, and ARIA in HTML lists which roles each element may take.

A live region chatters. `aria-live="assertive"` or `role="alert"` sits on a result count or a countdown that changes every keystroke or second, and the screen reader interrupts the user for each value. Use polite updates, which

`role="status"` gives by definition, announce once input settles, and keep the region quiet at load.

Further reading

- WAI-ARIA 1.2 — every role, state and property, which attributes each role supports, and which roles may not be named.
- ARIA in HTML — the roles and attributes each HTML element may carry, and which of them are redundant.
- Using ARIA — the W3C's practical guide, with the rules of ARIA use and an example of each.
- Accessible Name and Description Computation 1.2 — the algorithm that decides what a screen reader says a control is called.
- Understanding Name, Role, Value — what WCAG 2.2 success criterion 4.1.2 asks of every component, with techniques and failures.
- ARIA Authoring Practices Guide — the markup and keyboard behaviour for each common widget pattern, with working examples.



Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is ARIA?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/aria/