Barkhausen AI

NOTE

Reading the accessibility tree: what platform documentation says software sees

The accessibility tree is the structured representation a browser builds from a web page's markup.

“Every leap begins as a crackle.”

KIND	Note
DATE	2026
SUBJECTS	Engines & documentation; Conventions & policy
LICENSE	CC-BY-4.0
AUTHOR	Barkhausen AI

ABSTRACT

The accessibility tree is the structured representation a browser builds from a web page's markup. As defined by the WAI-ARIA 1.2 specification, it is a tree of accessible objects, each node exposing an element's role, states, and properties through the platform accessibility API; Chromium's engineering documentation describes its shape as derived from the Document Object Model (DOM) and modifiable through ARIA attributes. This note reads what platform and vendor documentation says consumes that representation: assistive technology, the Playwright browser-testing framework, and browser-automation and agent systems. The documentation shows a split — some named agent systems are documented as operating on the accessibility tree or ARIA semantics, others as working from screenshots. What the tree carries is structural and semantic, not the pixel-level appearance a screenshot captures, and that distinction maps onto how differently these systems are documented to perceive a page.

CONTENTS

A web page, as software receives it, is not only a stream of markup and pixels. Browsers build from that markup a second structure — the accessibility tree — whose purpose is to expose the page's meaning to software that does not read pixels. This note assembles what platform and vendor documentation says that structure is, how it is built, and which software is documented to consume it, from the W3C specifications that define it to the software — including browser-automation and agent systems — documented to read it. It is the companion collection to the whitepaper on how search optimization, generative engine optimization, and accessibility relate, which cites this note where its account of the machine readers turns to the accessibility tree. Every claim below is drawn from primary documentation, and quoted where the wording carries the point.

1 What the browser builds

The accessibility tree is defined normatively in the WAI-ARIA 1.2 specification, a W3C Recommendation published in 2023 [1]. Its glossary defines the accessibility tree as a "Tree of accessible objects that represents the structure of the user interface (UI)," adding that "Each node in the accessibility tree represents an element in the UI as exposed through the accessibility API; for example, a push button, a check box, or container" [1].

Where that tree comes from is described in Chromium's engineering documentation, which states that "The shape of the accessibility tree is determined by the DOM tree (occasionally influenced by CSS), and the accessible semantics of a DOM element can be modified by adding ARIA attributes" [2]. The tree is not authored directly. The browser derives it from the document, and the document's structure determines the tree's shape.

2 What ARIA changes in the tree

ARIA — Accessible Rich Internet Applications — is the vocabulary for the semantics the tree carries. WAI-ARIA 1.2 describes itself as providing "an ontology of roles, states, and properties that define accessible user interface elements" [1]; a role is an object's "Main indicator of type," a state expresses characteristics "that may change in response to user action or automated processes," and a property carries attributes "essential to the nature of a given object" [1]. Because ARIA attributes can, in Chromium's words, modify "the accessible semantics of a DOM element" [2], an author can change how an element is represented in the tree — its role, name, or state — separately from how the element looks.

Native HTML populates the tree without any ARIA at all. The W3C's current draft mapping, HTML Accessibility API Mappings 1.0, specifies that the `h1`–`h6` elements map to a "heading role, with the aria-level property set to the number in the element's tag name" [4]. An `<h2>` thus reaches software as a node carrying the heading role at level 2 — the mechanism by which a document's heading structure becomes machine-readable. HTML-AAM is a W3C Working Draft rather than a Recommendation, so its wording should be read as a draft mapping, not a settled requirement.

3 Who reads it, per their own documentation

The tree's first and defining consumers are assistive technologies. WAI-ARIA's stated purpose is to "allow assistive technologies to convey appropriate information to persons with disabilities" [1], and Chrome DevTools' documentation frames the tree as the representation of "how your web content is exposed to assistive technology" [3]. Assistive technology is not the only documented reader, however: Chromium's own documentation notes that accessibility APIs are "often used for automated testing scripts, and

UI automation software like password managers" [2] — software that reads the same tree for its own purposes.

Playwright, a browser-testing framework, is a second documented consumer — the only one of its class sampled here. Its documentation recommends role-based locators because a role locator "reflects how users and assistive technology perceive the page, for example whether some element is a button or a checkbox," calling it "the closest way to how users and assistive technology perceive the page" [5]. Playwright also exposes the tree directly: in its documentation, "aria snapshots provide a YAML representation of the accessibility tree of a page," a format that "describes the hierarchical structure of accessible elements on the page, detailing roles, attributes, values, and text content" [6].

Agent systems: a documented split

The newest documented readers are agent systems, several of them built atop existing browser-automation tooling, and here the documentation does not describe a single mechanism. It describes a split.

On one side are systems documented as operating on the accessibility tree or ARIA semantics. Microsoft's Playwright MCP server states that it "enables LLMs to interact with web pages through structured accessibility snapshots, bypassing the need for screenshots or visually-tuned models," using "Playwright's accessibility tree, not pixel-based input" and operating "purely on structured data" [7]; the same README documents an opt-in vision capability, so the server defaults away from pixels rather than excluding them. A separate Microsoft product, the Playwright CLI, is documented for coding agents as producing "accessibility snapshots with element refs," where "The snapshot file contains the accessibility tree with element refs for the next command" [8]; like the MCP server, the CLI also documents screenshot and PDF capture as a secondary capability, so it too defaults away from pixels rather than excluding them [8]. Browserbase's Stagehand documents a "DOM mode" whose tool table lists an `ariaTree` tool described as "Get accessibility tree of the page" [9]. And OpenAI's Publishers and Developers FAQ states, of its ChatGPT Atlas product, that "ChatGPT Atlas uses ARIA tags—the same labels and roles that support screen readers—to interpret page structure and interactive elements," advising site owners that "Making your website more accessible helps ChatGPT Agent in Atlas understand it better" [10]. That statement is now attached to a product with an announced end date: on 2026-07-09 OpenAI announced it is deprecating the standalone Atlas browser, scheduled to stop working on 2026-08-09, "moving browser-based agentic capabilities into ChatGPT and Codex" — without stating whether the ARIA-based interpretation carries into those successor surfaces [15].

On the other side are systems documented as working from screenshots. Anthropic's computer-use tool documentation, as archived on 2026-07-07, describes a tool that "provides screenshot capabilities and mouse/keyboard control," built on capturing screenshots and clicking coordinates, and does not describe consumption of the accessibility tree [11]. OpenAI's Computer Use API documents its built-in tool loop as one where "The model looks at the current UI through a screenshot, returns actions such as clicks, typing, or scrolling," after which "your harness sends back a new screenshot" [12]; the same guide also documents harness- and DOM-based options, so this describes one path among several rather than the product's only mechanism. Google's Gemini API documentation describes its Computer Use tool — built into Gemini 3.5 Flash, the documentation's recommended model for computer use, with the standalone Gemini 2.5 computer-use model listed as a legacy preview — as working from screenshots, "generating specific UI actions like mouse clicks and keyboard inputs," after which "your application captures a new screenshot and sends it back to the model" [13]. Amazon's Nova Act User Guide defines the model's "observation loop" as a cycle that "captures and processes the current state of the browser or environment, including

screenshot analysis" [14]. Stagehand's own "CUA mode" sits on this side of the split, alongside the DOM mode above — one framework documenting both mechanisms [9].

Two vendors document both approaches. OpenAI does so across two products: ChatGPT Atlas on the ARIA-consuming side, its built-in Computer Use tool loop on the screenshot side. Browserbase does so within one framework: Stagehand's DOM mode and its CUA mode. The documentation supports naming which systems are documented to read the tree and which are documented to read pixels; it does not support the broader claim that the accessibility tree is the interface through which agents, as a class, perceive the web.

4 What is absent from the tree

What the tree carries is bounded by its definition. Chrome DevTools' documentation states that "The accessibility tree is a subset of the DOM tree," one that "only contains elements from the DOM tree that are relevant and useful for displaying the page's contents in a screen reader" [3]. A tree of accessible objects, each carrying a role, states, properties, and a name [1], is a semantic and structural representation rather than a visual one. Information a sighted reader takes from appearance alone — position, color, proximity, typographic weight — is not, by this definition, part of the tree except insofar as it is also expressed through an element's role, name, or state. This is the boundary the split among agent systems runs along: the screenshot-based systems above read the pixels the tree omits, while the tree-based systems read the structure the pixels leave implicit. Playwright MCP's documentation draws the same line from the other direction, contrasting "structured accessibility snapshots" with "screenshots or visually-tuned models" [7].

5 What the documentation supports

Read together, the documentation supports a specific and bounded set of statements. The accessibility tree is a browser-built, standards-defined structure that exposes a page's semantics — roles, states, properties, names — to software that does not read pixels. Assistive technology, Playwright's testing tooling, and a named set of browser-automation and agent systems are documented as consuming it; other named agent systems are documented as working from screenshots instead. The documentation does not support a single, vendor-agnostic account of how software perceives a web page. These readings reflect the documentation as published at 2026-07-09 (the Atlas deprecation notice, announced that day, was read 2026-07-10); vendor documentation changes without notice, and the specifics should be assumed perishable.

REFERENCES

1. W3C (World Wide Web Consortium). *Accessible Rich Internet Applications (WAI-ARIA) 1.2 — W3C Recommendation* (2023). https://www.w3.org/TR/wai-aria-1.2/#dfn-accessibility-tree Accessed 2026-07-09. [archived]
2. The Chromium Project. *Accessibility Overview (docs/accessibility/overview.md) — undated living document* (2026). https://chromium.googlesource.com/chromium/src/+/main/docs/accessibility/overview.md Accessed 2026-07-09. [archived]
3. Chrome for Developers (Google). *Accessibility features reference | Chrome DevTools* (2026). https://developer.chrome.com/docs/devtools/accessibility/reference Accessed 2026-07-09. [archived]
4. W3C (World Wide Web Consortium). *HTML Accessibility API Mappings 1.0 — W3C Working Draft* (2026). https://www.w3.org/TR/html-aam-1.0/#el-h1-h6 Accessed 2026-07-09. [archived]
5. Microsoft. *Locators — Playwright documentation* (2026). https://playwright.dev/docs/locators Accessed 2026-07-09. [archived]

6. Microsoft. *Snapshot testing — Playwright documentation* (2026). https://playwright.dev/docs/aria-snapshots Accessed 2026-07-09. [archived]
7. Microsoft. *playwright-mcp — README (Playwright MCP server), GitHub* (2026). https://github.com/microsoft/playwright-mcp Accessed 2026-07-09. [archived]
8. Microsoft. *Introduction — Playwright CLI documentation* (2026). https://playwright.dev/agent-cli/introduction Accessed 2026-07-09. [archived]
9. Browserbase. *Agent — Stagehand documentation* (2026). https://docs.stagehand.dev/v3/basics/agent Accessed 2026-07-09. [archived]
10. OpenAI. *Publishers and Developers – FAQ, OpenAI Help Center* (2026). https://help.openai.com/en/articles/12627856-publishers-and-developers-faq Accessed 2026-07-09. [archived]
11. Anthropic. *Computer use tool — Claude Platform Docs* (2026). https://platform.claude.com/docs/en/agents-and-tools/tool-use/computer-use-tool Accessed 2026-07-09. [archived]
12. OpenAI. *Computer use — OpenAI API documentation* (2026). https://developers.openai.com/api/docs/guides/tools-computer-use Accessed 2026-07-09. [archived]
13. Google. *Computer Use — Gemini API documentation* (2026). https://ai.google.dev/gemini-api/docs/computer-use Accessed 2026-07-09. [archived]
14. AWS (Amazon Web Services). *Amazon Nova Act — User Guide (Glossary)* (2026). https://docs.aws.amazon.com/pdfs/nova-act/latest/userguide/nova-act-ug.pdf Accessed 2026-07-09. [archived]
15. OpenAI. *Evolving Atlas into ChatGPT for browser-based agentic work — OpenAI Help Center* (2026). https://help.openai.com/en/articles/20001371 Accessed 2026-07-10. [archived]

HOW TO CITE

Barkhausen AI (2026). Reading the accessibility tree: what platform documentation says software sees. https://barkhausen.ai/notes/reading-the-accessibility-tree/