Engineer

Screen reader

software that reads a page aloud, or into braille

What it is

A screen reader is a program that presents everything on a computer or phone, from the text of a document to the buttons around it and the state each one is in, as synthetic speech or as a line on a refreshable braille display, and lets its user work that interface from the keyboard or with touch gestures. The word covers a whole family of screen readers rather than one product: NVDA, which is free and open source, and JAWS on Windows; VoiceOver, built into macOS and Apple's mobile systems; TalkBack on Android. Its users are mostly blind or have low vision, and some people who can see the screen use one because listening is easier for them than reading.

The part most developers miss is where the words come from. A screen reader does not look at the pixels. The browser turns the HTML into a second structure, the accessibility tree, in which every element is reduced to a role (button, link, heading, list), a name (the words that identify it) and a state (checked, expanded, disabled). The operating system's accessibility interface hands that tree to the screen reader, which speaks it. A button whose visible content is a drawing of a paper plane and whose code has no text is, in that tree, a button with no name, and the reader says exactly that.

Nor do its users listen to a page from top to bottom. NVDA's browse mode turns single letters into navigation keys: `H` jumps to the next heading, `D` to the next landmark, `K` to the next link, `F` to the next form field, and the digits 1 to 9 to a heading of that level. `NVDA+F7` opens a list of every link, heading or landmark on the page. VoiceOver has the rotor, opened with `VO-U`, which offers the same lists. A sighted reader skims by glancing at bold type and whitespace; a screen reader user skims by jumping between structural elements, and a page without that structure has nothing to jump between.

Use the concept in a decision

Put “What is a screen reader?” 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 a team actually faces is whether the interface it ships works for someone who receives it only as the markup describes it. A screen reader is the most literal consumer a page has. Layout, colour, position, an icon everyone recognises, a field that turns red: every cue that lives only in the visual design is lost, and only what the code says reaches the listener.

That makes it the one test of the markup that no amount of looking at the page can replace. A design review, a screenshot comparison and a careful click-through all happen in the visual channel, and the visual channel is where the defects a screen reader exposes are invisible by definition. Automated rule engines help, and they catch real problems in bulk: an image with no alternative text, a form control with no label, an attribute with an impossible value. What they check is that a name exists, not that it is the right name. Hearing a checkout button announced as “button” or a menu announced as “hamburger” takes a person and a few minutes.

The cost of that person and those minutes is lower than most teams assume. NVDA costs nothing to install, and VoiceOver is already on every Mac and iPhone in the office, so the first test needs no purchase and no specialist. It needs someone to close their eyes, or turn the screen off, and try to complete the page’s main task with the keyboard and their ears. The fix for what they find is usually cheaper still when it happens early: a native `button` or `a` element brings its role and keyboard behaviour with it, while a clickable `div` needs all of that built and maintained by hand.

The trade-off is that screen readers differ. Each one interprets the accessibility tree its own way, each browser builds the tree its own way, and a page tested in one pairing can read differently in another. Testing in two common combinations covers more than testing one exhaustively.

What it looks like in practice

On one application the defect was in plain hearing. The interface drew its controls with system symbols, and a symbol with no accessibility label is read out under its own internal name, so the send button announced itself as “arrow up circle, button”. Seventeen icon-only controls were renamed for what they do, each label naming the action rather than the shape, and following the control’s state where its meaning changes with it. A linter now requires an accessibility label within eight lines of any icon-only control, a deliberately crude distance chosen by measurement, so an eighteenth cannot be added without one. The page is plain about the limit: the rule proves that a label exists nearby, and whether that label is correct still needs someone to listen to it.

Documents raise the same question in another format. A PDF that looks right on screen says nothing about whether a screen reader can read it, so at the recorded validation stage, nine generated documents, the CV variants, reference sheet, portfolio and cover letter across languages, were run through an independent validator for the PDF/UA accessibility profile and passed all 106 configured accessibility-profile rules. Passing required tagged structure, a declared document language, alternative text on every graphic that is not decoration, a real title in the metadata and an explicit reading order instead of the one the layout happens to produce. Both numbers come from one validator, and the page says so.

This site treats its own automated audits the same way. Its credits page calls those audits the floor rather than the proof, and asks anyone who uses a screen reader on it, and hears something read wrongly, to say so, because that is the one report the gate has no way to generate.

Where it goes wrong

Most screen reader failures are invisible on screen, which is why they survive review. They tend to take a handful of shapes.

The control announces its artwork. An icon button with no text is named from whatever the code provides: a file name, a symbol name, or nothing, which leaves a bare “button”. Give every icon-only control a name that describes the action, and hide the drawing itself from the tree:

```
<button type="button" aria-label="Send message">
    <svg aria-hidden="true" focusable="false"><!-- icon --></svg>
</button>
```

The page reads as one long stream. Headings that are styled `div` elements, and a layout without landmarks, look fine and leave `H` and `D` with nothing to land on, so the listener has to arrow through every line to find the part they came for. Use real `h1` to `h6` elements in order, and put the page's regions in `header`, `nav`, `main` and `footer`.

Something changes and the listener hears silence. An error message appears under a field, a basket count goes up, a toast slides in and out. Sighted users see it; the screen reader says nothing, because its focus is elsewhere. Move focus to the message, or put dynamic updates in a region marked with `aria-live` so the reader announces them.

Hidden content is read, or visible content is silenced. Text pushed off-screen with CSS is still in the accessibility tree and is read aloud; `aria-hidden="true"` on a visible element removes it for the listener while everyone else still sees it. Both are useful on purpose and confusing by accident, so check what the tree contains, not only what the screen shows.

Every check passes and the page still reads badly. A label can exist and be wrong, an alt text can describe the file and not the point, and a reading order can be valid and absurd. Put a short listening pass on the release checklist for any page that changed its structure.

Further reading

- Tools and Techniques — the W3C's overview of how disabled people use the web, screen readers among the other assistive tools.
- NVDA User Guide — browse mode, single-letter navigation and the elements list, from the makers of a free screen reader.
- Use the VoiceOver rotor on Mac — Apple's guide to jumping by heading, link and landmark.
- Core Accessibility API Mappings 1.2 — how roles, names and states travel from the browser to the platform interface a screen reader reads.

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 a screen reader?” 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/screen-reader/