How to install this tool

Add visitor tools to your website, with clear steps and files you can host yourself.

Guide 1.0.1 · 31 October 2026
Schema-3 commercial packages

Your installation route

Prepare your site. Upload the JS/WASM pair. Add one block. Verify it before making it available to everyone.

1 Before you change your website
2 Upload the pair. Add one script.
3 Check it on a real page
4 Optional settings, kept small
5 Hosting and security notes
6 Your site still needs checking
7 Update, roll back or remove
8 Troubleshooting
9 More checks and developer references

For website owners and administrators. The first sections cover installation; later sections help developers and hosting administrators. Read the HTML offline or keep a printed PDF.

Before you change your website

You need the purchased ZIP, access to your website's files and a place to add a JavaScript module to its shared layout.

1. **Keep an original copy.** Save the ZIP, its license and the current website template. Back up the site with your normal backup tool and confirm how you would restore it.
2. **Check the domain.** Open `build-configuration.json` in a text editor. Check `schemaVersion: 3`, `licenseProfile: "commercial"` and `licensedBaseDomain` against the domain you purchased.
3. **Choose a safe test page.** Use a restricted test page or staging subdomain under that licensed base. Keep real customer information out of test fields.
4. **Check the host's capabilities.** It must let you serve both runtime files and run an ES module. If it strips scripts or blocks `.wasm` uploads, ask its administrator before installing.

What the purchased ZIP contains

`accesskit_widget_wasm.js`
The JavaScript module that loads the widget.

`accesskit_widget_wasm_bg.wasm`
The matching compiled widget. Keep this pair together.

`build-configuration.json`
Domain, profile, policy and source identity for this build.

`checksums.json`
SHA-256 fingerprints for checking package integrity.

`INSTALL.txt`
Short installation notes.

`LICENSE.txt` **and** `THIRD-PARTY-NOTICES.txt`
Keep these with your package; follow the supplied terms.

> **Your domain is part of the build**
>
> For an illustrative license for `example.com`, the apex, `www.example.com` and `staging.example.com` share the licensed base. `example.net`, `localhost`, IP addresses and a hosting-provider preview domain do not. Tenant bases on shared hosting can differ. Use the exact base in your package; do not guess it from the last two labels. Editing the JSON or script cannot change the compiled license.

Upload the pair. Add one script.

The example below uses a same-origin folder. The folder name is an example; use the path you actually upload.

1. **Extract the ZIP.** Upload the two runtime files together to a new versioned folder in your website's public file area. Preserve both filenames exactly. Keep licenses, metadata and the original ZIP in your release archive.
2. **Confirm the public path.** In this example the module is served at `/assets/charmingviews/v1/accesskit_widget_wasm.js`. A disk folder, admin file-browser location or private download URL is not the public path.
3. **Add this to the shared HTML layout, just before** `</body>`**.** In a CMS, use its supported site-wide custom-code location that preserves `type="module"`. Save and publish that layout through your normal process.

```
<script type="module">
  import init, { mount } from
    '/assets/charmingviews/v1/accesskit_widget_wasm.js';
  await init();
  mount();
</script>
```

Copy the complete block. Replace only the public folder path. The leading `/` starts at your site's root, so nested page addresses use the same files.

Same folder	Shared layout	Visitor browser
`accesskit_widget_wasm.js` `accesskit_widget_wasm_bg.wasm`	One module block per document	Loads the widget on a licensed host

Why both initialization calls appear

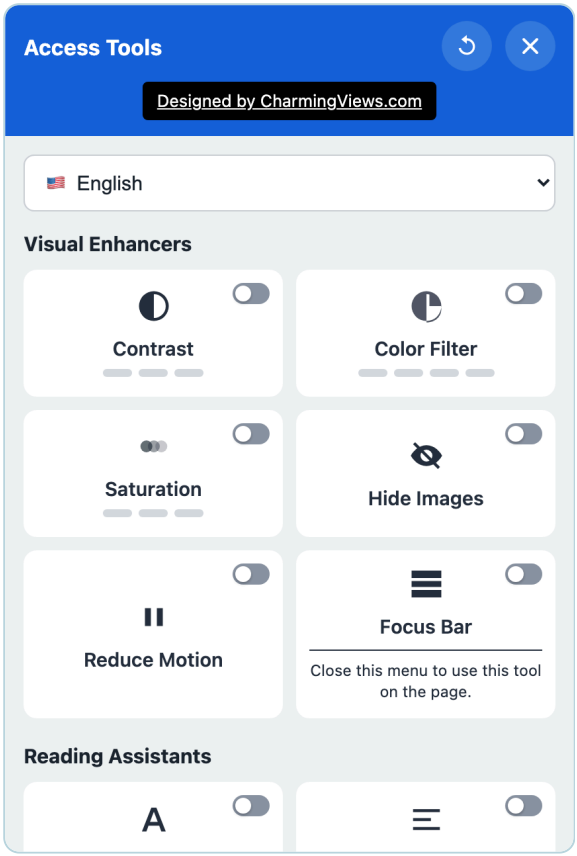
`await init()` loads the paired WASM and starts automatic mounting after the document is ready. `mount()` is the documented, idempotent explicit call for this same module instance. It does not justify loading multiple copies.

> **Keep the installation simple**
>
> Use one widget installation. Remove an older AccessKit embed or coordinate with an existing integration before adding this one. Do not paste the script into individual posts, every component, both the head and footer, or a tag manager in addition to the layout. No separate CSS, license key, Rust compiler or npm install is required for the supplied runtime pair.

Check it on a real page

Open the test page on the licensed domain in a fresh browser tab. Check the published page, not only the CMS editor.

- [ ] Find one accessibility-tools launcher. Open it with pointer and keyboard; check that it stays reachable above your cookie banner or chat control.
- [ ] Change Text size by one step. Confirm actual page text changes and important controls remain usable. Some pixel-fixed text may not scale.
- [ ] Close the panel with Escape. Reopen it, then use Reset to return tool settings to their defaults.
- [ ] Change a preference, reload the page and check it persists. Reset afterward. Storage restrictions or private mode can affect persistence.
- [ ] Visit a second page, a long page, a form and your mobile layout. Check navigation, focus, scrolling, overlays and any checkout you operate.
- [ ] Keep your site's existing keyboard and screen-reader checks. The widget does not fix missing labels, headings, alternatives or inaccessible content.



Actual schema-3 example build, default panel. Visible tools depend on build selection and browser capability.

If the launcher does not appear

Check the licensed hostname first. Then use the browser's Network and Console panels: both runtime requests should succeed, and Console should show no module, WASM or policy error. The message `AccessKit: disabled on this host` points to a hostname/license mismatch. See Troubleshooting.

> **What was checked for this guide**
>
> The exact main embed and configuration examples were run against a real schema-3 commercial fixture in Chromium on an isolated loopback server with the licensed hostname mapped locally. Mounting, a subdomain, preference/reset, repeated mount and blocked hosts were checked. This is not a compatibility guarantee for your site, production hosting or every browser.

Optional settings, kept small

The default installation works without a configuration object. If you need changes, set them *before* the first `await init()`.

This example moves the launcher to the left, disables the panel shortcut and hides Voice commands. **Use it instead of the earlier block.**

```
<script type="module">
  import init, { mount } from
    '/assets/charmingviews/v1/accesskit_widget_wasm.js';
  window.AccessKitConfig = {
    ui: { side: 'left', hotkey: null },
    features: { voiceControl: false }
  };
  await init();
  mount();
</script>
```

`ui.side`

`'left'` or `'right'`; default `'right'`. Check overlaps at narrow widths.

`ui.hotkey`

Default `'Alt+Shift+A'`; `null` disables it. Browser, OS or assistive-technology shortcuts can conflict.

`features`

Boolean tool keys, such as `voiceControl: false`, narrow available controls. They cannot add a tool omitted from the compiled build.

Interface language

The widget uses a saved widget language, then the browser's language, with fallback to English. Its language picker changes the widget interface, not your website content. To set Spanish from your own language control, run this after initialization:

```
if (window.AccessKit) {
  await window.AccessKit.ready;
  window.AccessKit.setLanguage('es');
}
```

Use `'en'` for English. Supported interface codes are `en`, `de`, `es`, `it`, `fr`, `ru`, `tr`, `ar`, `hi`, `zh-cn`, `jp`, `pt`. Preserve these exact codes. Speech language and voice availability are separate browser capabilities.

Set configuration once, before mounting, then fully reload. For other dimensions, accent colors or translation overrides, ask your developer to review the source contract and check usability and contrast.

Hosting and security notes

For the straightforward installation, serve both runtime files from the same HTTPS origin as the page.

HTTP responses and MIME types

Serve the module as JavaScript, for example `text/javascript`, and the binary as `application/wasm`. Both requests must return the actual files, not an HTML login, a "file not found" page or an SPA fallback. The loader has a non-streaming fallback for a wrong WASM MIME type, but fixing the response is preferable. See Browser references.

The purchased widget is not installed by double-clicking a file on your computer. `file://` introduces module-origin restrictions and has no licensed website hostname. This guide itself can be read offline.

Content Security Policy (CSP)

If your site uses CSP, ask its security owner to review the exact Console violation. The inline module in this guide needs an authorized hash or nonce, or an approved external module entrypoint. Module loading, the WASM fetch and WebAssembly compilation also need to satisfy the site's policy. Browsers can require `'wasm-unsafe-eval'` for compilation under `script-src`.

The widget inserts style elements and applies page styles. A strict `style-src` policy can block those operations; this package exposes no documented CSP nonce setting. A script nonce alone does not authorize the generated styles. A strict site may need a separately reviewed integration before this package works there.

> **Keep your existing protections**
>
> Do not disable CSP, CORS, mixed-content protections, authentication or a web firewall to make the widget appear. Avoid blanket `'unsafe-eval'` or wildcard allowances. Give your developer the specific error and failed request so they can choose the narrowest supported change.

CDN or other-origin hosting

This is an optional developer integration. JavaScript modules and WASM fetches must be permitted by the file host's CORS responses and the page's CSP. An explicit WASM URL can be supplied to the module initializer, but the paired files and version must still match. Use the same-origin example unless there is a tested reason to change it. A CDN's hostname does not expand the licensed *page* domain.

HTTPS and speech

Use HTTPS for a public site and avoid HTTP asset URLs inside HTTPS pages. Voice commands depend on browser recognition support, microphone permission and potentially a browser/vendor speech service. Check those services and your own privacy/consent requirements before enabling or testing speech.

Your site still needs checking

Universal installation describes the HTML integration pattern. It does not promise that every CMS, page design or browser behaves the same way.

Ordinary pages and single-page apps

For normal full-page navigation, use the shared layout so each document loads the same version once. For an SPA, put startup in its stable browser-only application shell after the DOM exists. Do not run a new module copy on every route, render, hydration or component mount.

The runtime has a per-module singleton and an idempotent `mount()`. It has no documented destroy/unmount API. Keep `#axs-host` outside a region the framework replaces. Different module URLs, including different query strings, can create separate instances; use one stable URL per release.

After a route change, check tool effects on the new content, focus/reading helpers, headings, forms and Reset. DOM replacement and host CSS can affect behavior. History navigation retaining one launcher was checked for this guide; full framework-specific route integration was not.

Frames, shadow DOM and third-party content

Do not assume page tools reach into embedded cross-origin frames, closed shadow roots or third-party checkout widgets. Each document has its own boundary. Check the actual accessible behavior of that content rather than installing extra copies to reach it.

Mobile and browser limits

- [ ] The visitor needs a browser that supports JavaScript ES modules and WebAssembly, with scripting allowed. No minimum browser-version matrix is promised by this guide.
- [ ] Check small screens, landscape, on-screen keyboards and overlap with fixed banners. Pointer-based magnification and a larger cursor are less relevant on touch-only devices.
- [ ] Read-aloud uses browser speech synthesis; Voice commands use recognition support and permission. They may be absent or unavailable. Focus-based read-aloud is not a replacement for a native screen reader.
- [ ] Hyphenation depends on the document language and browser support. Widget language does not translate content or supply missing document language.
- [ ] Preferences use the site origin's local storage when available. They do not synchronize across browsers, devices or subdomain origins.

> **Keep accessible content at the source**
>
> The widget offers visitor preferences and assistance. It does not automatically remediate your content, certify WCAG/ADA compliance, replace semantic HTML or guarantee that every accessibility barrier is removed.

Update, roll back or remove

Update without mixing runtime versions

1. **Keep the working release.** Archive its pair, metadata, licenses, embed and package checksums. Record the public path and the date.
2. **Check the new package.** Verify its licensed base and profile and compare its files with its `checksums.json` using your release tools. Checksums detect changes; they do not independently prove who supplied a package.
3. **Upload to a new folder.** For example, use `/assets/charmingviews/v2/`. Upload both files fully before changing the layout's import path. Here `v1`/`v2` are your release-folder names, not promised product versions.
4. **Test, then switch the shared embed.** Repeat the checks in Check it on a real page. Keep one installation. A full reload is needed; a mounted page does not turn into the new runtime automatically.
5. **Check caches.** Confirm the browser requests the new folder's JS and WASM. Review your CMS/CDN and service-worker caches if old content persists. Do not place a query string only on the JS and assume that versions the WASM URL too.

Roll back

Restore the shared layout's previous import path to the archived, matching pair. Refresh or invalidate the relevant HTML caches, fully reload, and repeat the verification checks. Retain the failed release for diagnosis; do not mix its JS with the old WASM.

Clean uninstall

1. Optionally use Reset in the widget before removal. This clears known tool preferences while retaining the saved widget language.
2. Remove the embed and any custom code that calls `window.AccessKit`. Remove an older tag-manager integration too if it loads the same widget.
3. Publish the layout, refresh relevant HTML/service-worker caches and fully reload. Confirm the launcher is gone and the browser no longer requests either runtime file.
4. Remove the hosted runtime folder when no pages depend on it. Keep your purchased ZIP and license in your records.

> **Do not clear all site storage**
>
> Removing the script stops future mounting after a full navigation/reload. It does not erase preferences already stored in a visitor's browser or immediately detach a running page. For a completely fresh local test, a developer may remove only the widget's known `axs.*` preference keys, including `axs.lang` if wanted. Do not use `localStorage.clear()`; that can remove unrelated site data.

A commercial schema-3 build has no time-based runtime expiry. That describes the binary's behavior; it does not promise future updates, support duration, transfer rights or any additional license terms.

Troubleshooting

Start with the symptom, then check the response or hostname before changing code.

No launcher; Console says “disabled on this host”

Compare the page hostname with `licensedBaseDomain` in `build-configuration.json`. Use a test subdomain under that base. Localhost, an IP and provider previews are not included. If the supplied base is wrong, report it with your purchase through the support/contact route supplied with your order; do not edit or patch the binary.

“Failed to load module script”, JavaScript MIME error, or 404

Check the exact module URL in Network. It must return the JS file and a JavaScript MIME type. Fix the public path, filename case, permissions or server response. Check whether the CMS removed `type="module"` or treated code as text.

WASM fetch, compile, magic-word or link error

Inspect the WASM response. A login page, HTML error, truncated transfer or mismatched pair can cause this. Re-upload both files from the same ZIP and check hashes. Serve `application/wasm`. Review CSP errors separately.

CSP, CORS or mixed-content message

Give your site/security administrator the exact blocked URL and directive. Prefer same-origin HTTPS. Follow the Hosting and security notes; do not turn off security controls.

Two launchers, repeated handlers or conflicting panels

Search the shared template, tags, plugins and application components for every widget loader. Keep one module path and one integration. Remove duplicates and fully reload. A repeat `mount()` in one module is idempotent; separately loaded copies are different.

Launcher appears but settings do not affect a part of the page

Check pixel-fixed CSS, replaced content, frames, shadow boundaries and existing widgets/overlays. Test one tool at a time, then Reset. The widget does not rewrite inaccessible source content.

> **Capture useful evidence**
>
> Record the guide/package version or source identity, affected page path, browser/version, device, steps and the first relevant Console/Network error. Redact tokens, private URLs, form values and personal details before sharing. Do not send a full network archive without reviewing it.

More checks and developer references

Preferences do not persist

Check private mode, storage restrictions and whether the page moved to another origin. `www` and apex domains may share a license while keeping separate browser storage. Test after a full reload.

Settings, language or tool list looks wrong

Set config before initialization and fully reload. A saved widget language can override browser detection. Tools can be omitted by the build, disabled by config or gated by browser capabilities. Do not assume a runtime flag adds an uncompiled tool.

Speech or a microphone feature does not work

Check browser support, permission, HTTPS and any required vendor service. Do not request microphone access just to verify the basic installation. Use the other controls when speech is unavailable.

The old release keeps loading

Check the actual request URLs, response/cache headers and any service worker. Switch both files through a new versioned folder and refresh the HTML cache. A hard refresh alone may not replace a service-worker cache.

Optional read-only diagnostic

A developer can run this in the Console on the affected published page. An absent API can mean startup failed or the host was denied; combine it with the error message and Network responses.

```
const kit = window.AccessKit;
console.table({
  hostname: location.hostname,
  apiPresent: Boolean(kit),
  enabled: kit?.enabled,
  language: kit?.lang
});
if (kit) console.log(kit.buildConfig);
```

Browser references

Core instructions and examples are included here for offline use. These optional online references explain browser rules, not a widget compatibility or support promise.

- MDN: JavaScript modules—module MIME, module scripts and local-file restrictions.
- MDN: WebAssembly.instantiateStreaming()—streaming MIME and policy requirements.
- MDN: CSP script-src—script authorization and WebAssembly compilation.
- MDN: CORS—requests between different origins.

If a tool doesn't work as expected on a particular page, please send us the page URL and describe the problem. We'll do our best to investigate and help, though we can't guarantee a fix for every website.

Email: support@buzzforus.com.

Keep this guide with the purchased package. The supplied `LICENSE.txt` governs use. Follow the support/contact instructions in your purchase materials; this guide sets no response time or refund policy.