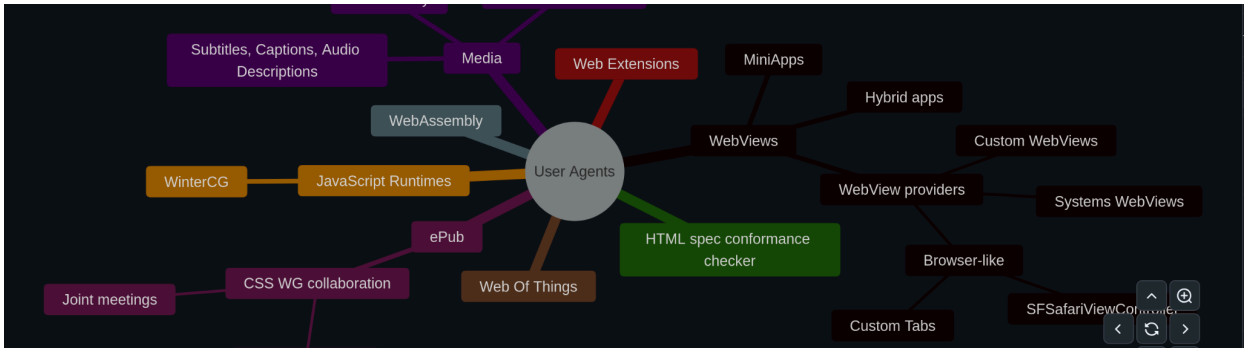
⁉️ Premise



We need to talk more

The web is much more than browsers. What works great in browser development is missing in related spaces like embedding.

Can we get together to collectively describe security and compatibility properties of user agents that are not browsers? Can we develop things that don't fit in browsers collectively with cross review just like it's happening for every browser facing web feature?

The Web Is More!

Previous Discussions

TPAC 2024 Web Standards Beyond The Browser

TPAC 2025 Modular Web Engines Breakout

WebView CG Meeting June 2026

Working Draft

The Web Embedding Group

Brainstorming by Niklas Merz with support from LLMs and real people

Editors: Martin Alvarez-Espinar

Mission. The [Web Embedding] Community Group exists to understand, document, and improve the use of web technologies outside the browser, such as WebViews, hybrid and packaged applications, including MiniApps, and embedded runtimes. Web engines are embedded in vastly more contexts than browsers, yet these are not always considered while developing new standards or improving the current web. **These contexts lack shared terminology, functionality, and security baselines, challenging developers' expectations of platform compatibility.** The group works to give the "embedded web" the kind of common foundation the main browser-based web already has, preserving its essential values and universality.

Scope:

- taxonomy and terminology for embedded web contexts;
- documented use cases and pain points across embedders;
- threat models and baseline security requirements for embedded engines
- compatibility expectations and gap analysis between embedded engines and the browser web platform;
- investigation of packaging convergence across MiniApps, Isolated Web Apps, and PWAs;
- investigation of a minimal embeddable web platform profile.

Out of scope: re-defining "the web", standardizing proprietary super-app APIs, producing W3C Recommendations (work that matures here transitions to an appropriate WG), browser-internal architecture.

Related Groups / Liaison: WebView CG, MiniApps CG & WG, High-Performance Baseline for Web Apps CG, Web Platform Incubator CG, TAG, WinterTC, WHATWG.

Workstreams

Editor's note: Is multiple workstreams the way to go? Is this too much & broad?

Track 1 — Landscape & Terminology

Description. Web engines are embedded in far more contexts than they are browsed in — WebViews, MiniApps, hybrid and packaged applications, super apps, kiosks and TV runtimes — yet there is no shared vocabulary for these contexts and no common understanding of what

they need from the platform. In these heterogeneous contexts, the user agents offer different capabilities—some scenarios extend the standard web features, and others are based on reduced web engines. This track establishes the factual and terminological foundation the other tracks build on: what "embedding the web" means, who does it, why, and where it hurts.

What do we call the two types of user agents we are using here? Embedded Web and Drive-by Web?

Editor's note: We as WebView CG spent much of our time analysing the status quo. It's useful to understand what we have today but we shouldn't get stuck in this field of work forever and try to get into actually working on new solutions as soon as possible.

Possible Deliverables:

- **A landscape report** defining a taxonomy of embedded web contexts and the characteristics that distinguish them (trust model, distribution, lifecycle, engine ownership, and additional features).
- **A use cases** and pain points report, gathered directly from embedders and developers, identifying which problems are shared across contexts and which are unique.
- A public information website serving as the canonical reference for the group's terminology and findings.
- Periodic engagement with the TAGs on user agent findings.

Scope: Documenting and classifying what exists today and what is emerging (including AI-generated embedded applications). **Out of scope:** defining "the web" as such, and advocating for any particular embedding technology over another.

Example activities: Interviewing and surveying embedders (app frameworks, super-app vendors, device manufacturers), cataloging existing embedding technologies and their constraints, analyzing where emerging patterns such as AI-generated apps fit the taxonomy, publishing explainers and blog posts that socialize the terminology.

Track 2 — Security

Description. Embedded contexts inherit the web's content model but not its security model: the embedder can intercept traffic, inject script, suppress UI signals, extend APIs and access to local resources, and bridge into native capabilities, while the user has none of the cues a browser provides. At the same time, regulation such as the EU Cyber Resilience Act is beginning to impose concrete security obligations on products that ship embedded web engines — without any shared baseline for what "secure" means in these contexts. This track documents how the browser security model changes under embedding and develops a common baseline embedders can implement and point to.

Editor's note: W3C has "The threat model for the web" but the embedding cases are likely not mapped yet. Similarly the ETSI standard for browser security and CRA compliance have WebViews and MiniApps out of scope for now.

Possible Deliverables:

- **A threat model report for embedded web contexts**, documenting how and where browser security guarantees (origin isolation, transport integrity, permission UX, process isolation) degrade or disappear when embedded.
- **A baseline security requirements** note for engine embedders and WebView providers, written with regulatory readiness (CRA and successors) as an explicit design goal.
- **Reviews and gap analyses** of existing embedding technologies against that baseline.
- **Requirements for content security policies** (CSP) applicable to embedded web contexts, aligned with the threat model for these contexts, while preserving the principles of the web.

Scope: Threat modeling, requirements, and best practices at the level of the embedding interface and runtime. Coordination with the WebView CG, the MiniApps CG, and relevant security groups rather than duplication of it. **Out of scope:** vulnerability research against specific shipping products, and certification or conformance enforcement.

Example activities: Mapping the JS-bridge attack surface across major WebView implementations, documenting how permission prompts and identity signals behave (or don't) across embedded contexts, tracking CRA implementation guidance and translating it into engineering requirements, workshops with embedders on hardening patterns.

Track 3 — Compatibility & Documentation

Description: Each WebView, MiniApp platform, and embedded engine supports a different slice of the web platform, and almost none of it is documented where developers look. MDN and browser-compat-data describe the browser web only. This track makes the embedded web measurable and documented, and feeds the results back into those shared resources instead of building a silo.

Possible Deliverables:

- **Compatibility data** across embedding contexts in WebViews, MiniApp runtimes, embedded engines. Something similar to caniwebview.com for other contexts could make sense.
- **Gap analyses** showing where embedded contexts diverge from the browser platform and from each other.
- **Developer-facing documentation** and tooling.
- Upstream contributions to MDN, BCD, Baseline and Interop

Scope. Measuring and documenting platform support across embedded contexts, including the test automation to do it. MiniApp work happens in coordination with the MiniApps CG. This track just supplies data and evidence. Spec work stays with the WG. Out of scope: Recommendations and proprietary embedder APIs.

Example activities. Automated test runs across WebView and MiniApp runtimes; per-context gap reports; contributing embedded support data to BCD and Baseline; developer surveys on which gaps hurt most.

– Brian's notes —

Windowing

I think there is a lot here that we can talk about in terms of windowing alone - for example, proper window-level things for apps:

* Styleable tooltips
* Custom context menus
* Sticky windows
* Window decorations

And we can look at a ton of already currently open standards positions as potential app/ui fodder too.. Just some examples of things that at least someone is proposing and almost certainly would be valuable at a higher level, but *maybe* not just for the "drive by web":

- Window Controls Overlay for Installed Desktop Web Apps #557
- Additional Windowing Controls #96
- window-drag #668
- Menu elements proposal #580
- CSS ::tooltip Pseudo Element #526
- Side Panel #320
- Tabbed Web Apps 195
- Isolated Web Apps #184
- Multi-Screen Window Placement API #117
- Web App Launch handler #90
- VirtualKeyboard API #16
- Web Background Synchronization (BackgroundSync)#14

💡 Finding The Thing

Success creates success

Let’s find one thing that many need and are interested in working on collectively

Draw out all ideas and find common things

🗣 Projects, Groups, People

Groups & Projects

- Mini Apps Working Group & Community Group
- WebView Community Group
- Embedded Web Engines & Native Web Runtimes Community Group
- Isolated Web Apps
- WPE WebKit
- Window control overlay

Supporters

-

Incentives

Unsuitable for the drive-by web but important for embedding the web

Some of these capabilities were judged unsuitable for the open web's trust model; others are native to embedding and have simply never had a venue. Both point at the same gap.

Capabilities exposed only to Isolated Web Apps in Chromium

Isolated Web Apps are a Chromium-specific packaged-app boundary shipped initially via enterprise policy on managed ChromeOS devices (Mozilla: negative; WebKit: no position). Capabilities gated behind it currently evolve in a single engine, without cross-vendor review:

- **Controlled Frame** — embedder-controlled top-level browsing context, IWA-only. WICG
- **Direct Sockets** — raw TCP/UDP; Gecko closed without a position, no WebKit signal. Google Groups
- **Borderless display mode** — IWA-only due to spoofing risk

Capabilities declined for the open web, implemented via proprietary bridges in embedded contexts

Some work might eventually happen in WHATWG but examples like this got initially declined by WebKit as fingerprinting vectors:

- Web Bluetooth
- Web MIDI
- Web NFC
- WebHID
- Web Serial
- WebUSB
- Magnetometer
- Ambient Light
- Proximity
- Battery Status
- Network Information
- Device Memory
- background Geolocation
- User Idle Detection
- Bluetooth Scanning
- HDCP policy check

Mozilla rates the device-access subset as harmful.

Also: File System Access beyond OPFS. All are available today in Electron, Cordova/Capacitor, MiniApp runtimes, and enterprise WebView apps through unstandardized native bridges.

WebView proposals without a destination group

- **WebView security model** A threat model for WebViews + MiniApps, CRA compliance
- **Locally Hosted WebView Content**; **Preloading WebView Pages** (WebView CG explainers). WebView Explainers Repo
- **JS bridge / native messaging** — three proprietary mechanisms (Android, WebKit, WebView2), no explainer yet (To be done in WebView CG)
- **Embedder configuration surface** — what an embedder may disable/override (JS, cookies, TLS errors, mixed content) and what the page can detect about it. Currently undefined everywhere.

MiniApps — standards without implementations

The inverse of the IWA case: here the specifications exist and the deployments don't.

- **MiniApp Manifest, Packaging, Lifecycle, Addressing, Widgets** — W3C specs near stable; real-world implementations and deployments remain limited, which is currently the biggest blocker. MiniApps WG · Packaging draft
- **Runtime fragmentation** — most MiniApp runtimes are built on WebViews that differ heavily in API surface, performance, and stability across platforms; the WG maintains gap-analysis docs comparing what MiniApps can do in major super apps versus what the web platform supports. MiniApp behavior in the wild is only as good as the WebView underneath — making this directly downstream of every WebView item above. W3C
- **Collaboration explicitly requested** — the WG plans to update its charter with a collaboration model spelling out how it works with IWA, the WebView CG, and others to avoid duplicated effort, and participants stressed MiniApps should not become a forked "subset Web" drifting from the main platform. TPAC 2025 notes

Web Bundles / Web Packaging — contested substrate

- **Web Packaging** — original format: Mozilla negative; Web Bundles at IETF: neutral. Positions GitHub
- **Underpins both IWA (signed bundles) and MiniApp Packaging** — two of the three packaging stories above rest on a format no second engine has endorsed. Signing, distribution, and update semantics remain unresolved.

🤯 Reasoning

Capabilities without a venue

Capability tiers. A growing set of capabilities for example raw sockets, device access (USB, HID, Serial, Bluetooth, NFC, MIDI), arbitrary content embedding have been declined by engines for the open web on fingerprinting and security grounds, yet is universally needed and ad-hoc implemented in embedded contexts via proprietary bridges. Some now ship in a single engine restricted to Isolated Web Apps, outside TAG review and standards-position processes. Some of these extended capabilities (e.g., Bluetooth, NFC) are also offered to MiniApps developers within SuperApp contexts. The result is that real platform capabilities evolve with no multi-vendor venue. This group provides that venue: not to relitigate whether these APIs belong on the drive-by web, but to define the trust tiers of embedded contexts — installation, packaging, signing, policy — under which such capabilities can be exposed with a shared, reviewable security model, and to document where each existing capability sits today.

Some work in WHATWG

Internal rationale — not for the public charter

An argument for the group's existence, buried in Chromium's own process docs.

Chromium's internal guidance for launching IWA-specific APIs literally instructs engineers: do not request a TAG design review. The TAG concluded it isn't the right venue for IWA-specific proposals; Mozilla's position on IWA is negative (API design, security, venue); WebKit has taken no position. Chromium's guidance concludes there is little value in requesting standards positions for IWA-specific capabilities. New platform capabilities — Direct Sockets, Controlled Frame, and whatever comes next — are now shipping with *no multi-vendor review venue at all*.

https://www.chromium.org/blink/launching-features/isolated-web-apps/#who-do-i-request-feedback-from-for-iwa-specific-apis

https://github.com/w3ctag/design-reviews/issues/842

https://github.com/mozilla/standards-positions/issues/799

The capability tier underneath it. This isn't just an IWA quirk. For example WebKit declined to implement 16 APIs including Web Bluetooth, Web MIDI, Web NFC, WebHID, Serial, WebUSB and various sensors, on the grounds that they increase fingerprintability with no safe way to protect the user. Mozilla flagged Bluetooth, MIDI, Serial and USB as harmful, arguing no permission prompt can explain the danger to a non-technical user. Both objections are about the *drive-by web's* trust model — a stranger's URL getting at your hardware. But every one of those capabilities is common in the embedded world: Electron apps, kiosks, MiniApps with native bridges, enterprise WebView apps all do raw sockets and device access today, each through a

proprietary bridge, with zero shared security model. The engines' answer was "not for the open web." Nobody ever answered "then for what, under which rules?"

https://webkit.org/tracking-prevention/

https://mozilla.github.io/standards-positions/