Tab 1

FedCM Identity Handler: Service Worker Interception — Design Doc

This Document is Public Set doc permissions to: "Anyone with the link", add contributors@chromium.org and googlers@chromium.org as either editors or commenters (uncheck "notify" when doing so).

Authors: brandm@microsoft.com, monicach@microsoft.com **July 2026**

One-page overview

Summary

Let an Identity Provider (IDP) **declare a FedCM-specific Service Worker in its `/.well-known/web-identity` file** so that the user agent can intercept credentialed FedCM requests (accounts, id-assertion, disconnect) and hand them to the worker before hitting the network. The worker receives a purpose-built `IdentityRequestEvent` (not a `FetchEvent`) and may `respondWith()` a synthesized response or fall through to the default credentialed fetch. This enables IDPs — e.g. Microsoft Entra — to augment sign-in requests (e.g. attach device-bound / proof-of-possession credentials obtained from a native broker) from a context that has no `window`.

The worker is **registered and managed by the user agent** under an **isolated, deterministic, FedCM-only storage partition** that page JavaScript cannot target, so FedCM registrations neither collide with the IDP's ordinary first-party Service Workers nor become reachable/unregisterable by arbitrary `Match Service Worker Registration` callers.

Provenance for the design decisions below is in Appendix A.

Platforms

All platforms that ship FedCM: **Mac, Windows, Linux, Chrome OS, Android**. Not Android WebView / WebLayer / iOS (no FedCM). Android carries binary-size sensitivity (see Speed).

Team

- Brandon Maslen
- Evelyn Chou
- Monica Chintala

Bug

- Chromium: crbug.com/526074797 (Identity Handler Service Worker interception).
- Explainer: https://github.com/w3c-fedid/identity-handler (and FedCM PR #822 explainer, PR #834 registration/lifecycle model).

Code affected

- **`content/browser/webid`** (browser-process FedCM): new `IdentityHandlerServiceWorkerManager`, `IdentityRequestEventDispatcher`; changes to `idp_network_request_manager`, `accounts_fetcher`, `metrics`, `flags`.
- **`content/browser/service_worker`**: `ServiceWorkerVersion` external-request plumbing, `FakeServiceWorker` test support.
- **Blink renderer `modules/credentialmanagement`**: new `IdentityRequestEvent`, `IdentityRequestRespondWithObserver`, `IdentityRequestServiceWorkerGlobalScope`.
- **Blink `modules/service_worker`**: `ServiceWorkerGlobalScope` event dispatch, `WaitUntilObserver` glue.
- **Mojom**: `third_party/blink/public/mojom/webid/fedcm_identity_request.mojom`, `service_worker.mojom`.
- **Flags/metrics**: `content_features`, `runtime_enabled_features.json5`, `about_flags.cc`, `flag_descriptions`, histograms.
- **WPT**: `external/wpt/fedcm/fedcm-identity-handler/*`.

Design

Background and motivation

FedCM today makes credentialed requests to IDP endpoints directly from the browser process. IDPs increasingly want to **participate in that request** — for example, to bind the session to device hardware, attach a proof-of-possession header, or apply conditional-access policy. A Service Worker is the natural interception point, but the standard `FetchEvent` is a poor fit for FedCM: FedCM requests originate in the browser process (often with **no client page open**), carry forbidden headers (`Sec-Fetch-Dest: webidentity`), and must not be observable by unrelated page code sharing the origin.

The Enterprise scenario driving this: an Entra Service Worker intercepts the FedCM `id-assertion`/`accounts` request, obtains a **device-bound cookie** from the OS authentication broker via the companion `self.platformAuthentication.executeGetCookies` API, and returns a response bound to the device — instead of an exfiltratable bearer token. This design doc covers the **interception mechanism**; the broker/`get-cookies` API is a separate, complementary work item (see Deltas doc).

Alternatives considered: the dispatch vehicle

FedCM sets client = null on every endpoint request — they are UA-initiated, often with no IDP page open — so any vehicle that resolves its target through an initiating client is unavailable.

Four vehicles were considered. Flipping service-workers mode to "all" and using standard Handle Fetch is impossible: HTTP fetch dispatches through the request's client, which is null . A browser-side layer firing FetchEvent directly has no client to name as the event source, and would implicitly enroll any IDP worker with a generic fetch handler. A dedicated FedCM worklet is feasible but rejected on cost (see below). The chosen vehicle, a dedicated identityrequest event, uses Fire Functional Event … on registration to target a registration's active worker without Handle Fetch or a client — the primitive Payment Handler uses — and requires an explicit listener to opt in.

Why not a worklet: Would have to define from scratch what Service Workers already specify: persistent registration, script update and install/activate, a clientless UA-driven wake, network egress (WorkletGlobalScope has no fetch), quota and site-data integration, and DevTools. Reuse buys all six for one new event type.

Discussion and provenance: #833 and Appendix A.

Registration model (browser process)

The IDP declares the handler in `/.well-known/web-identity`:

None

```
{
  "provider_urls": ["https://idp.example/fedcm/config.json"],
  "identity_handler": {
    "service_worker": "/fedcm/sw.js"
  }
}
```

Key properties (implemented by `IdentityHandlerServiceWorkerManager`):

- **UA-managed registration.** FedCM itself registers the declared script — the IDP page never calls `navigator.serviceWorker.register()`. This is essential for the long-lived-session scenario, where the IDP page may not be visited for weeks/months but FedCM still makes credentialed requests. Registration is triggered from the accounts-fetch path (`idp_network_request_manager` → `IdentityHandlerServiceWorkerManager::Register`).
- **Same-origin enforcement.** The `service_worker` script must be same-origin with the `.well-known` file; cross-origin declarations are rejected and never registered (endpoints fall back to network). Malformed (non-string) declarations are ignored.
- **Isolated, deterministic storage partition.** Registrations are stored under a FedCM-only `blink::StorageKey` created with a **nonce derived deterministically** from the handler origin (`SHA1("fedcm-identity-handler-v1:" + origin)` → 128-bit `UnguessableToken`). Isolation comes from the **browser being the sole minter** of this key (renderers cannot inject an arbitrary `StorageKey` nonce), not from secrecy. Determinism lets any later FedCM flow **recompute** the key to look up or unregister the worker with **no persisted side-state**. This directly answers the two spec-review concerns on PR #815: (a) pre-existing IDP Service Workers are *not* auto-enrolled, and (b) arbitrary `Match Service Worker Registration` callers cannot reach or unregister the FedCM worker.
- **Scope.** Currently derived from the script URL (`sw_url.GetWithoutFilename()`). The model's explicit `scope` field and `enabled` kill-switch are **not yet implemented** (gap; see Deltas doc).
- **Update via cache = all**, so the IDP controls update discovery through `Cache-Control`/`ETag` on the SW script (soft-update model, below).
- **Lifecycle.** Registered when `identity_handler` is present; unregistered when the IDP removes the declaration. Cleanup is stateless because the key is recomputable. If the manager is destroyed mid-registration (e.g. `AccountsFetcher` torn down), the destructor signals failure so parked credential requests are released rather than hanging.

Why nonce isolation rather than a shared service worker:

FedCM handler registrations live in a partition keyed by a browser-minted nonce derived from the IDP origin, not in the origin's ordinary service worker registration pool. A page-side getRegistrations() call does not see them, and navigator.serviceWorker.register() cannot create or replace one.

What this buys:

- No collision with page-registered workers. If FedCM shared the ordinary pool, a registration declared in .well-known and one created by register() could name the same scope with different scripts, and the behavior on conflict, on unregistration, and on removal of the well-known declaration would all have to be defined. These were raised on 10 Jun (#833); isolation sidesteps them rather than answering them.

- No implicit enrollment. A worker the IDP wrote for page traffic cannot be drawn into handling credentialed FedCM requests by accident.

- A stable identity for the registration that does not depend on what the IDP's pages do at runtime.

What is not yet settled. Whether isolation is required - rather than merely the simpler of two workable designs - has been asked in #833 and not answered: Farolino asked directly whether any security reason mandates a separate pool. The current endorsement of the nonce approach (Yanagisawa, 11 Jun) is on the grounds that it is the cleaner design, not that sharing is unsafe. An alternative that marks isolation with a flag on StorageKey rather than a nonce is also under consideration, and a security review of the implications of either was requested on 15 Jun and has not concluded. Note also that the nonce key determines more than registration visibility: it also fixes the handler's network and storage partition. Those two roles are coupled in the current implementation and need not be - see Appendix A.

Relationship to Payment Handler Service Workers

Same ExtendableEvent base, same respondWith() shape, same option of UA-initiated registration (Payment Handler's just-in-time install ≈ our .well-known declaration), and a UA-defined respondWith timeout (10s here).

- Identity Handlers use a browser-minted nonce StorageKey that renderers cannot address. Pre-existing IDP workers are not auto-enrolled, and arbitrary Match Service Worker Registration callers cannot reach or unregister the FedCM worker.
- Payment Handler allows page-initiated register() ; FedCM does not.
- Payment Handlers are user-chosen and may openWindow() for UI. Identity Handlers run with no client page, so isolation carries the security weight that Payment Handler places on explicit user choice.

- An unanswered `paymentrequest` cancels the payment; an unanswered `identityrequest` falls back to the default credentialed fetch

Soft-update model

Registration is separated from freshness. Rather than re-registering on every FedCM call, FedCM initiates a **Service Worker soft update in parallel** with the credentialed request that triggered the invocation; the currently active worker handles that request while a newer version installs/activates for a later request. `updateViaCache: all` + IDP `ETag` allow most update checks to be served from the HTTP cache. The CL contains soft-update markers on the functional-event path; full "soft-update-on-every-invocation-when-already-registered" parity with the model should be confirmed (see Deltas doc).

Bad rollout recovery

Handler failures are not uniform. If the handler never calls respondWith(), the request falls back to the network. But once respondWith() is called, the request fails without fallback - whether the promise rejects, the response is not JSON, or the body exceeds the 1 MiB limit. A handler that is broken after calling respondWith() therefore breaks sign-in for that IDP rather than degrading to the network path.

The reliable recovery is to remove the identity_handler declaration from the well-known file. The next FedCM flow that fetches the file unregisters the worker outright, and this path does not consult the script's HTTP cache.

Rolling forward to a fixed script is less reliable. Registrations use updateViaCache: 'all', so the update check honors the HTTP cache for the worker script. IDPs should serve both the well-known file and the handler script with a short max-age; otherwise a cached buggy script can survive a re-registration.

A browser-side circuit breaker - suspending a handler after N consecutive failures - is not implemented. The error taxonomy needed for one already exists; see Followup work.

Event dispatch data flow

None

```
Browser process                                Renderer (IDP Service
Worker)
```

----
idp_network_request_manager
| (accounts / id-assertion / disconnect, before network fetch)
▼
IdentityRequestEventDispatcher
| FindReadyRegistration → StartWorker (StartRequest
external-request slot)
| mojo: IdentityRequestEventData{ endpoint, request:
FetchAPIRequest }
▼
ServiceWorkerGlobalScope ——————————▶ dispatch "identityrequest"
IdentityRequestEvent
(ExtendableEvent)
.endpoint, .request
(destination=webidentity)
.respondWith(Promise<Response>)
|
IdentityRequestRespondWithObserver
-
isTrusted()/second-call guards
- type basic|default,
ok status, JSON MIME
- Fully read body, cap
1 MiB while reading
| mojo callback:
┌———————————————————————————————————┘ OnResponse(status,
body, headers)

```
  |
OnError(IdentityRequestError)
  ▼                                              OnNotHandled()
IdentityRequestResponseCallbackImpl
  | FinishRequest(external-request slot); apply 10s timeout
  ▼
idp_network_request_manager
  - OnNotHandled → default credentialed network fetch
  - OnResponse → parse as if it were the IDP's JSON response
  - OnError    → FAIL the request (no network fallback)
```

- **Single event type.** One `identityrequest` event carries an `endpoint` (`accounts` | `id-assertion` | `disconnect`) rather than three event types. The worker handles the endpoints it wants and simply doesn't call `respondWith()` for the rest. Rationale (recorded in the spec note): one handler, uniform request shape. Trade-off: the worker is woken even for endpoints it won't handle.
- **`request` is browser-constructed** with `destination = "webidentity"`, carrying method, headers, credentials mode, and body. The worker may call fetch(event.request) to perform the network call itself, or synthesize a Response . A worker-initiated fetch() does not carry the IDP's SameSite=Lax / Strict cookies: the handler runs under the nonce StorageKey , whose IsolationInfo propagates to the worker's URLLoaderFactory , giving a null SiteForCookies and a nonce-scoped cookie partition that is ordinarily empty. SameSite=None; Secure behavior is uncharacterised. Falling through without respondWith() yields the UA's default credentialed fetch, which is unaffected
-

Interaction with multi-IDP and active mode

Handlers are per-IDP and independent. The storage key is derived from the handler origin (GetFedCmStorageKey), and the dispatcher is installed per handler origin, so in a multi-IDP get() each IDP has its own registration, its own isolated storage partition, and its own dispatcher. An endpoint request is dispatched only to the handler for the IDP it is addressed to; IDPs that declare no handler fall through to the network as usual. Failures are likewise scoped: if one IDP's registration fails or its handler does not respond, only that IDP's parked requests are released to the network, and the other IDPs in the same call are unaffected.

Active mode does not currently support registered IDPs, so identity handlers do not apply there.

Threads / processes / storage / network

- **Browser UI thread** owns `IdentityHandlerServiceWorkerManager` and `IdentityRequestEventDispatcher`. SW registration/lookup go through `ServiceWorkerContextWrapper`.
- **Storage:** the isolated FedCM `StorageKey` partition in the Service Worker database.
- **Network:** the IDP SW may `fetch(event.request)` (its own network egress) or synthesize a `Response`; the UA validates response `type`/status/MIME before parsing.
- **Renderer:** the SW global scope, the `IdentityRequestEvent`, and the **Failure taxonomy** (`IdentityRequestError` mojom / `DispatchFailureReason`): `kNotHandled` → network fallback. `kPromiseRejected` / `kInvalidResponse` / `kBodyLoadFailed` / `kRespondWithTimedOut` → **hard fail without fallback** (a worker that claimed the request cannot be silently bypassed on failure — matches `FetchEvent` semantics).`IdentityRequestRespondWithObserver` (body reading, guards).

Major component responsibilities

Component	Responsibility
`IdentityHandlerServiceWorkerManager`	Register/unregister the declared SW under the isolated key; wire the dispatcher into the network manager; deterministic key derivation.
`IdentityRequestEventDispatcher`	Start the worker, dispatch `IdentityRequestEventData`, manage the external-request slot + 10s `respondWith()` timeout, map results to `DispatchFailureReason`.
`idp_network_request_manager`	Call the dispatcher before each credentialed fetch; consume SW response or fall back / fail.
`IdentityRequestEvent` (Blink)	Web-exposed event: `endpoint`, `request`, `respondWith()`.

`IdentityRequestRespondWithObserver` (Blink)	Validate the response, enforce the 1 MiB cap while reading, report success/typed error over mojo.

Metrics

Success metrics

- **New** `Blink.FedCm.IdentityHandler.Registered` (boolean): registration success/failure rate (recorded by `RecordIdentityHandlerRegistered`).
- **New** interception funnel: dispatched → responded (`respondWith` called) → fell back to network (`OnNotHandled`) → failed (typed by `IdentityRequestError`). This lets us distinguish rejected / invalid-response / body-load-failed / timed-out in the field (reviewer ask: label the "failed" histogram by reason).
- Existing FedCM outcome metrics (`Blink.FedCm.*`) should show no regression in success rate for non-handler IDPs.

Regression metrics

- FedCM end-to-end latency (existing FedCM timing histograms): the SW dispatch adds a worker start + `respondWith` round trip on the critical path for handler IDPs. Watch the accounts and id-assertion timing buckets.
- Service Worker external-request lifetime / stuck-worker counts (guard against the `respondWith`-never-settles slot leak; see Security/Followup).
- Android APK size (see Speed) — currently a launch blocker on the CL.

Devtrials

Finch feature: **`FedCmIdentityHandler`** (`content_features` / `runtime_enabled_features.json5`, `status: "test"`), also exposed as a `chrome://flags` entry for local testing. Standard experiment-controlled rollout; acceptance criteria: no regression in FedCM success/latency for the control population and clean interception funnel + registration success for enrolled IDP-origin test traffic. WPT virtual suite `virtual/fedcm-identity-handler` runs the feature on.

Rollout plan

Standard experiment-controlled rollout behind `FedCmIdentityHandler` (dev → beta → stable with Finch), gated on WPT + the security items in the Deltas doc being resolved and the spec (PR #815) being updated to the UA-managed model so we don't ship ahead of an unstable spec. No external milestone deadline; quality-driven.

Core principle considerations

Speed

- **Critical-path cost:** for handler IDPs, each credentialed endpoint now incurs a worker start + event dispatch + `respondWith` before the response is available. Mitigations: soft-update runs *in parallel*, `updateViaCache: all` avoids redundant script fetches, and a 10s `respondWith` timeout bounds worst case. Non-handler IDPs are unaffected (a single `Match`/registration check that returns null).
- **Binary size (Android):** the CL currently **fails** `android-binary-size` (~22–23 KB vs 16 KB budget). This must be resolved (feature-flag guarding, code sharing, or a size waiver) before launch. Recommend Finch-gating any plausibly speed-affecting behavior.

Simplicity

- **No new user-visible UI.** All behavior is IDP-facing. The IDP's mental model is "declare a worker in `.well-known`, the browser runs it for credentialed FedCM requests." Because the browser owns registration and the storage partition, IDP authors do **not** manage registration lifecycle, scope collisions, or unregistration from page code — a simpler contract than self-registered Service Workers, at the cost of a more complex browser-side implementation (isolated key, deterministic recompute, soft update).

Security

Threat model. Assume a compromised renderer and a potentially buggy/hostile IDP Service Worker on the IDP's own origin.

- **Isolation.** FedCM registrations live under a browser-minted nonce `StorageKey`. Renderers cannot mint this key, so page JS cannot enumerate, hijack, clobber, or unregister the FedCM worker, and pre-existing IDP Service Workers are not auto-enrolled. The worker only ever sees requests that already carry the IDP's own cookies (same trust boundary as the IDP server); no new (user, RP) correlation vector is created; uncredentialed endpoints (well-known, config, client_metadata) are never dispatched.
- **Forged events.** `respondWith()` checks `Event.isTrusted` and a respond-with-already-called flag, throwing `InvalidStateError` — so script-constructed events cannot provide responses or double-respond. (Implemented in the renderer; the spec algorithm must be updated to include these guards — see Spec Change Proposal.)
- **Response confinement.** The UA only accepts `type` `basic` (same-origin IDP) or `default` (SW-constructed) responses with an OK status and a JSON MIME type; `cors`/`opaque` are rejected, matching the same-origin, `redirect: error` posture of the normal credentialed fetch.
- **Resource exhaustion.** The response body is read with a **1 MiB cap enforced during the read** (`kMaxIdentityResponseSizeBytes`), matching the existing network path, so a hostile SW cannot allocate a multi-GiB Blink string.
- **Worker-lifetime DoS (watch item).** A `respondWith(new Promise(()=>{}))` that never settles must not keep the worker warm past the 10s budget. The dispatcher must promptly `FinishRequest` the external-request slot on timeout; a regression here lets an attacker keep a worker warm indefinitely and races unregister/soft-update. Needs an explicit unit test (raised in CL review).
- **URL/origin decisions** use origins (`url::Origin`), not string URLs.

Explicitly not Foreign Fetch

Foreign fetch let a service worker intercept cross-origin requests made to its origin from any client. It was removed from the spec (w3c/ServiceWorker#1188) over double-keying, unclear origin-trial results, and unclear use cases. Recorded here so it need not be re-litigated.

Does not apply:
- Implicit opt-in. The dedicated identityrequest event requires an explicit listener; pre-existing IDP workers are in a different partition and never enrolled.
- Arbitrary cross-origin interception. Three UA-initiated endpoints only, all same-origin to the IDP and already visible to its server.
- Double-keyed storage. One partition per handler origin; RP identity is not an input to the key.

Applies, mitigated:
- Persistent JS / botnet (crbug 40085867). Wake is UA-initiated, only during an active FedCM call, bounded by the 10s respondWith budget.

- Recursive dispatch. A handler's own fetch() is an ordinary same-origin request and cannot re-enter dispatch; asserted by handler-fetch-does-not-reenter.https.html.

What to point to: the request travels a trusted path inside the browser, unreachable from ordinary web content or script, so it offers no generic double-keyed-cache bypass.

Privacy considerations

No new cross-site information flow: the handler only intercepts already-credentialed same-origin IDP requests and cannot read cookie values. Standard FedCM privacy review applies; use the internal template for formal privacy review.

Testing plan

- **Unit:** `identity_handler_service_worker_manager_unittest.cc` (registration success/ failure, deterministic key, unregister-on-declaration-removal, mid-registration teardown); `identity_request_event_test.cc` (event surface, `isTrusted`/double-`respondWith` guards); `idp_network_request_manager_unittest.cc` (SW response consumption, fallback, typed errors). **Add** the `respondWith`-never-settles → external-request-count-returns-to-zero test.
- **WPT** (`external/wpt/fedcm/fedcm-identity-handler/`): same-origin interception, cross-origin rejection, respond-with-failure paths, IDL harness, constructor. **Add** register-rejects-on-malformed-well-known and the double-`respondWith` negative path (reviewer ask).
- **Special conditions:** Android binary-size gate; virtual suite `virtual/fedcm-identity-handler` to run with the feature enabled.

Followup work

- Split current CL into multiple layered CLs
- Update Explainer/Spec PRs to align with updated approach

Appendix A - Decision provenance

Which decisions carry Working Group alignment and which are ours. Sourced to w3c-fedid/FedCM#833, read in full (20 May - 22 Jun 2026).

WG-ALIGNED (agreed on a call or in-thread with editor assent)
- Intercept credentialed FedCM endpoints at all. 19 May call; PR #815.
- Sec-Fetch-Dest / request destination need not be preserved. Resolved, 19 May call.
- Dedicated IdentityRequestEvent rather than FetchEvent. 20 May.
- client_metadata excluded from dispatch. 27 May call.
- Response-type restriction can be relaxed. 27 May call. Note: the implementation still enforces it.
- Cookie-scope invariant can be relaxed - the IDP receives the full cookie jar, including Lax/Strict as well as None. 27 May call.
- Transparent fallback when the SW does not respond. 20 May.

IMPLEMENTER DIRECTION (SW owners' stated preference, not resolved)
- No destination: "webidentity" exception added to fetch(). 25 May.
- respondWith() takes Promise<Response>. Not formally closed; a use-case analysis was requested 22 May.
- opaqueredirect rejection retained as foolproofing. 25 May.
- Isolated nonce StorageKey. Endorsed 11 Jun as "a superior choice"; a StorageKey bit was proposed as an alternative; a security review of either was requested 15 Jun and has not concluded.

CHROMIUM IMPLEMENTATION DECISIONS (ours, not put to the WG)
- Single event plus an endpoint discriminator.
- Hard-fail rather than fallback on respondWith error.
- 10s respondWith timeout; 1 MiB response body cap.

OPEN
- UA-managed registration from .well-known. Two approaches still on the table; see PR #834.
- Registration matching by storage key and declared scope. 10 Jun.
- Soft-update model: the custom parallel flow vs. the spec's Soft Update algorithm. Implementer pushback, 10 Jun.
- The enabled kill-switch field. Its necessity was questioned 10 Jun.

Known gaps between this design and the WG record:
1. The implementation enforces the response-type restriction the WG agreed could be relaxed.
2. Soft update uses a custom parallel flow rather than the spec's.
3. .well-known is re-fetched on every request; a transient failure then breaks auth for an already-registered worker.

4. scope and enabled are declared but unimplemented.
5. Storage-pressure eviction clears the registration but not cookies, leaving a possible dangling registration id.
6. updateViaCache: "all" can pin a broken worker for the cache lifetime; a sw_version force-update knob is proposed, unanswered.

Provenance: rows citing the 19 May, 27 May, 2 Jun and 9 Jun calls come from participants' in-thread summaries, not the published minutes (w3c-fedid/meetings). Nothing after 22 Jun 2026 is reflected.