DemandFlow Sense adapters

DEMANDFLOW®

Technical specification: how sensing sources connect, how adapters function and how they integrate

SPECIFICATION

Purpose and scope

This document describes the adapter layer of DemandFlow® Sense: the interface every sensing adapter implements, the runtime that drives adapters, the adapters available today, how results flow into the platform, and what an operator or vendor needs to provide to connect a new sensing source.

It is written for solution architects, integration engineers and vendor partners. The companion data sheet covers the commercial capabilities of Sense (catalogue, zones, feasibility, governance, assurance, charging and customer outputs); this document covers the connection between those capabilities and the sensing systems beneath them.

Design principles

- **One interface, many vendors.** Every adapter implements the same small set of operations and returns the same vendor-neutral shapes. The Sense core never sees a vendor's own field names, states or authentication.
- **Configuration before code.** For REST vendors, endpoints, paths, authentication and the field mapping are configuration on the adapter record, so a new vendor can be connected without a release.
- **Isolation.** Each adapter runs in its own invocation, for its own tenant. One vendor's slowness or outage cannot delay or fail another.
- **No secrets in the open.** Credentials live in the tenant's encrypted secrets vault and are decrypted only for the duration of a call. Nothing in the adapter layer logs a credential, a token or a raw vendor payload.
- **Data residency.** Results are written only to the tenant's own sensing store in the tenant's home region. A tenant whose home region is not a Sense region is skipped, never redirected.
- **Pure, testable mapping.** The mapping from native results to platform records holds no cloud, clock or global state, so every adapter is tested against a mock

At a glance

Item	Value
Required operations	`discover` , `activate` , `deactivate` , and one of `stream` or `poll`
Optional operation	`capabilities` (live KPI figures for feasibility)
Adapters today	Gateway vendor cloud (REST), 5G-Advanced sensing function via NEF, 6G sensing function (draft, trials), simulator
Defined, awaiting a vendor	Enterprise Wi-Fi controller, radar head end
Authentication	None, API key header, bearer token, OAuth 2.0 client credentials
Delivery modes	Pull on a cursor (poll), push to a verified callback (NEF notification), server-sent events (6G draft)
Runtime cadence	Fan-out every minute; one isolated run per adapter per tenant

endpoint and the same mapping code runs in the app and in the runtime.

Item	Value
Health	Up, Degraded, Down (after 3 consecutive failed cycles); adapter-down incident raised automatically
Result types	Track, event, aggregate, heartbeat
Event types	Detection, boundary alert, intrusion, presence, fall
Result store	Per-tenant table in the home region, keyed by zone and hour, TTL from the zone's retention
Vendor timeouts	10 s per request, 2 MB response cap (REST); 3 attempts with linear back-off (6G draft)

Architecture

The adapter runtime is a scheduled service with three entry points: a timed fan-out that drives every adapter, an HTTP route for push notifications from a network exposure function, and admin routes that let an administrator test a connection, import discovered sources and query live capabilities. Results leave the runtime on a queue and are written to the tenant's sensing store by a separate ingest service, which also keeps a per-minute summary in the tenant's main table and pushes the latest results to the open zone screen.

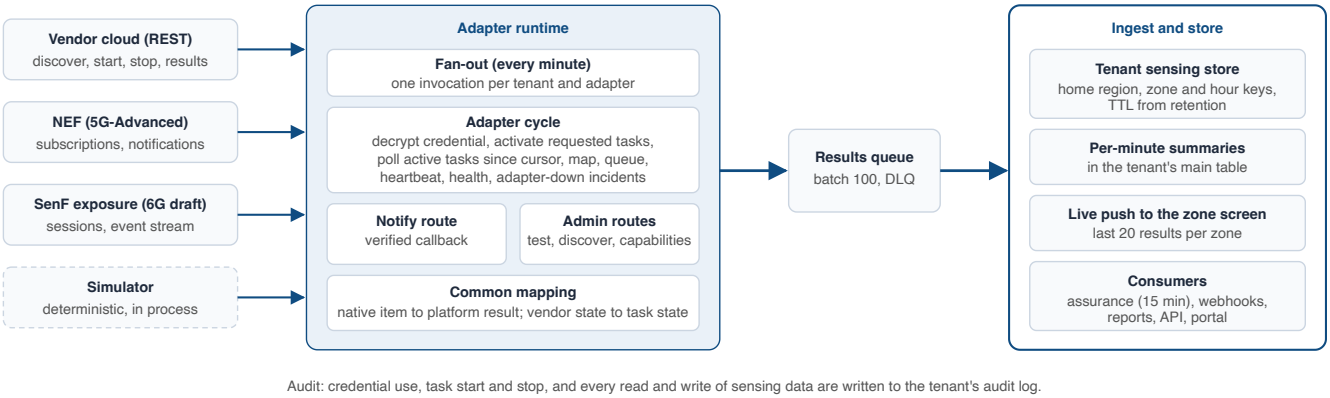


FIGURE 1. THE ADAPTER RUNTIME, THE RESULTS QUEUE AND THE INGEST PATH INTO THE TENANT'S SENSING STORE

The adapter interface

An adapter is a module that exports the operations below. `discover`, `activate` and `deactivate` are required, and an adapter must implement `stream` (push) or `poll` (pull) to be usable. `capabilities` is optional: when an adapter does not answer it, feasibility falls back to the stored KPI model for the source type. An implementation is checked

against this list before it is used.

Operation	Behaviour
`discover(ctx)` **returns** `NativeSource[]`	Lists the sources the adapter can currently see. Used by the admin "Discover" action to import sources, matched to existing source records by the vendor's own source id so a repeat import is idempotent.
`activate(ctx, taskRequest)` **returns** `{ externalTaskId, state }`	Places a sensing task with the vendor for one zone: the zone geometry (GeoJSON Polygon, LineString or Point), height band, acquisition mode, the vendor ids of the sources placed on the task, the update interval and the result types wanted. Returns the vendor's task id and its native state.
`deactivate(ctx, externalTaskId)` **returns** `{ state }`	Stops a task previously activated; returns the vendor's state after stopping.
`poll(ctx, externalTaskId, since)` **returns** `NativeResultItem[]`	Pull form. Returns the native items observed since an ISO timestamp (empty for everything available). The runtime keeps the cursor on the task record, so each cycle asks only for what is new.
`stream(ctx, externalTaskId, onNative)` **returns a stop function**	Push form. Calls `onNative(item)` as results arrive and resolves to a function that stops the stream.
`capabilities(ctx, { zone, sourceExternalIds })` **returns** `NativeCapability[]`	Optional. Live figures (range, position error, update interval, detection probability, false alarms, alert latency, availability) per source and acquisition mode, with a validity time. Used by feasibility in place of the KPI model while they are valid.

The context (`ctx`) carries everything an adapter needs and nothing it should keep: the adapter record (endpoint, paths, authentication type, configuration), the decrypted credential for this call only, the fetch implementation, the current time and, for a task, the zone's geometry and height band. Adapters hold no state between invocations: cursors live on the task record and an OAuth token cache lives only in the runtime's memory for the life of an invocation.

Vendor-neutral shapes

NativeSource (from `discover`)

Field	Meaning
`externalSourceId`	Required. The source's id in the vendor system; the idempotency key for discovery.
`sourceType`	Required. `gateway` , `ap` , `radar` , `gnb` or `senf` .

NativeResultItem (from `poll` or `stream`)

Field	Meaning
`externalSourceId`	Required. Must match a source placed on the task, or the item is rejected.
`observedAt`	Required. When the vendor observed it. An item without a parseable time is rejected.
`kind`	Required. `track` , `event` , `aggregate` or `heartbeat` .

Field	Meaning
`name`, `vendor`, `firmware`	Display name, vendor record (when one adapter serves several vendors), firmware.
`capabilities`	Any of `detect`, `track`, `classify`, `presence`, `fall`, `count`.
`modesSupported`	Any of `monostatic`, `bistatic`, `multistatic`.
`syncAccuracyNs`, `backhaulLatencyMs`	Timing accuracy and backhaul latency; feasibility uses them to decide whether sources can pair for joint sensing.
`lat`, `lon`, `rangeM`	Position and declared range, the basis of the coverage footprint.
`deviceId`	The inventory device this source corresponds to, when the vendor knows it.

Field	Meaning
`eventType`	For events: `detection`, `boundaryAlert`, `intrusion`, `presence`, `fall`.
`objectClass`, `trackId`	The classified object (unknown values map to `unknown`) and the vendor's track id.
`position`, `velocityMs`, `headingDeg`	Latitude, longitude and optional height; speed and heading.
`positionErrorM`, `confidence`	The source's own error estimate; confidence 0 to 1 (clamped).
`count`	For aggregates.
`rawRef`	A reference to the raw vendor record, carried for audit. The raw payload itself is never stored or logged.

Task states

A vendor's native task state is normalised to one of `requested`, `active`, `suspended`, `failed` or `ended`. Common vendor words are recognised (running, started and up become active; paused becomes suspended; stopped, completed, terminated and cancelled become ended; error becomes failed; pending and queued become requested), and the REST adapter's field map can declare any other mapping.

The runtime cycle

Every minute the fan-out lists the tenants with the Sensing module enabled and starts one isolated run per non-retired adapter. Each run is a single pass:

1. **Residency check.** Resolve the tenant's home region; skip the tenant if Sense is not deployed there. All reads and writes that follow use a client for that region.

2. **Readiness.** A non-simulator adapter needs an endpoint, a linked credential whenever the authentication type is not `none`, and the token URL and client id for OAuth 2.0. A missing setting is reported on the adapter's last error and the run ends.
3. **Credential.** Decrypt the linked secret for this run only; write an audit event recording that the credential was used.
4. **Activate.** For each task in state `requested`, build the vendor-neutral task request from the zone and its sources and call `activate`. Record the vendor's task id and normalised state, set the cursor to now, and audit the start.
5. **Pull.** For each `active` task of a polling adapter, call `poll` since the task's cursor. Each native item is validated and mapped, and every valid item is sent to the results queue as `{ subscriptionId, adapterId, taskId, zoneId, receivedAt, native }`. The cursor advances to the newest observation sent. Push adapters are not polled; their results arrive through the notify route.
6. **Probe.** An adapter with no tasks to run is probed with `discover`, so a down adapter is only marked up when the vendor answers again.
7. **Health.** A clean cycle sets status `up` and the heartbeat time and clears the failure count. A cycle with errors on some tasks sets `degraded`. A cycle that fails outright increments the failure count; after three consecutive failures the adapter is `down`.
8. **Incidents.** When an adapter is down, one incident ticket is raised, keyed on the adapter so a continuing outage does not raise another, listing the zones its active and requested tasks serve; those tasks are marked failed so the zone's KPI health and the operations screen reflect it.

Neither the fan-out nor the run ever throws: every error is logged and recorded on the adapter or task as `lastError`, and the run returns a summary (status, tasks activated, results sent, failed tasks, tickets raised).

The adapters

Gateway vendor cloud (REST)

A generic connector for Wi-Fi gateway sensing clouds. Because no two vendors expose the same API, everything vendor-specific is configuration on the adapter record:

Setting	Use
`endpointUrl`	Base URL of the vendor cloud. Relative paths resolve under it.
`authType` and credential	`none`; `apiKey` in a header named by `apiKeyHeader` (default `X-Api-Key`); `bearer`; or `oauth2ClientCredentials` with `tokenUrl`, `oauthClientId` and `oauthScope`, the token cached until 60 seconds before expiry. The secret itself is a linked secrets-vault record.
`discoverPath`	GET: lists the vendor's sources, mapped by the field map's `sources` section.
`activatePath`	POST: starts a task. Blank means the vendor streams every gateway continuously, so activation is synthetic and the task is active at once.
`deactivatePath`	DELETE, with `{task}` replaced by the vendor task id.
`pollPath`, `pollIntervalS`	GET, with `{since}` and `{task}` replaced and URL-encoded; later pages add a `cursor` query parameter from the field map's `results.cursor`.
`fieldMap`	JSON mapping the vendor's records to `NativeSource` and `NativeResultItem` (below).

Requests are JSON, with a 10 second timeout and a 2 MB read cap. A non-2xx response is reported as a plain sentence with the HTTP status; the credential and any access token never appear in an error. Errors from the vendor are surfaced on the adapter's test result and last error, not swallowed.

The field map

The field map turns a vendor's JSON into the common shapes without code. Each section names the path to the list of items, an optional cursor path, and a field spec per target property. A field spec is a dot path (`a.b[0].c`), a path with a time unit for epoch timestamps, a path with a lookup table for closed sets, or a constant. A lookup value the table does not know drops the item when the target is required and leaves the field unset when it is optional. Paths are read safely: only own properties are followed, and no code is evaluated.

```
{
  "sources": { "list": "data.devices", "fields": {
      "externalSourceId": "id", "name": "label",
      "lat": "location.lat", "lon": "location.lng", "firmware": "fw",
      "sourceType": { "const": "gateway" },
      "capabilities": { "const": ["presence", "fall"] } } },
  "results": { "list": "events", "cursor": "next", "fields": {
      "externalSourceId": "deviceId",
      "observedAt": { "path": "ts", "unit": "s" },
      "kind": { "path": "type", "values": { "motion": "event", "occupancy": "aggregate", "ping":
"heartbeat" } },
      "eventType": { "path": "type", "values": { "motion": "presence", "fall": "fall" } },
      "confidence": "score", "count": "people" } },
  "states": { "RUNNING": "active", "STOPPED": "ended" }
}
```

Required targets are `sources.fields.externalSourceId` , `results.fields.externalSourceId` , `observedAt` and `kind` . The map is validated when it is saved and every error is a plain sentence the settings screen shows, so an administrator can correct it without a developer.

5G-Advanced sensing function (via NEF)

DemandFlow acts as the application function (AF). The adapter subscribes to a sensing service for a zone through the operator's network exposure function and receives results by push notification, so it implements `activate` and `deactivate` only; `poll` returns nothing and there is no cursor.

- **Subscription pattern:** the NEF northbound pattern of TS 29.122 and TS 29.522: `{apiRoot}/<api>/v1/{afId}/subscriptions` , a notification destination, a self link, a test-notification request, 201 Created with a Location header, and errors as ProblemDetails (TS 29.571).
- **Area shapes:** the geographic area shapes of TS 23.032 in their JSON form (TS 29.572): point, point with uncertainty circle, polygon of 3 to 15 points. Larger polygons and lines are approximated by their bounding box and flagged as approximated.
- **Procedure:** per the Release 20 integrated sensing study (TR 23.700-14): the AF asks the NEF for a sensing service, the NEF authorises it and forwards it to the sensing function, and results return to the AF through the NEF.
- **Notification URL:** `<notify base>/sense/notify/{subscriptionId}/{adapterId}` , generated per tenant and adapter.
- **Settings:** endpoint, the application-function id (`afId`), the credential for the NEF, and a callback secret the NEF must present on every notification.

The sensing exposure API has no final stage 3 specification yet, so the API name, attribute names and the shape of the results report are held in one block and are confirmed against the operator's documents when an operator signs. A pilot needs an endpoint, an AF id and credentials.

The notify route (push delivery)

Push notifications arrive on an HTTP route owned by the runtime. It is treated as a third-party webhook: nothing is read or written before the sender is verified. The bearer or callback token in the request is compared with the adapter's callback secret in constant time, and an unknown tenant and a wrong secret look identical from outside (401), so the route cannot be used to probe which tenants or adapters exist. After verification, a retired adapter, an adapter of the wrong type, or an inactive subscription returns 404. Each result in the notification is mapped and queued exactly as a polled result is, and the matching task's state is updated from the notification's state.

6G sensing function (draft)

A draft adapter against the current 3GPP integrated sensing and communication exposure draft (TS 23.137), so an operator can trial a sensing-function API before Release 21 is final. It is switched on per adapter by the configuration flag `enableDraftSenf`; `activate` refuses before any endpoint call unless the flag is exactly true. Message flow:

Operation	Call	Message
`discover`	`GET /sensing-functions`	Lists sensing functions as sources.
`activate`	`POST /sensing-sessions`	SensingServiceRequest (zone as a sensing area, mode, update interval, result types) to SensingServiceResponse (session id and state).
`stream`	`GET /sensing-sessions/{id}/events`	SensingResultReport as server-sent events, each mapped to a native item.
`deactivate`	`DELETE /sensing-sessions/{id}`	SensingServiceStop.

Authentication is a bearer token from the linked credential. Transient failures are retried up to three attempts with a linear back-off. A Polygon of 3 to 15 points is sent as a polygon; larger shapes and lines become a bounding box marked approximated; a point becomes a point with an uncertainty circle. The message shapes are our reading of the draft and will be revised when the release is final.

Simulator

The simulator implements the full interface with no radio behind it. It holds no state: the vendor task id is derived from the task id, and the items for a time window are a pure function of the task and the tick, so the same window always yields the same results. Tracks and events are placed inside the zone's geometry and height band; a track arrives each tick, an event every third tick and an aggregate once a minute. It is the first source of every tenant, so the catalogue, zones, feasibility, live results, KPIs, incidents, charges, the portal, the API and webhooks all run end to end before any vendor is connected, and it is the fixture every other adapter is tested against.

Enterprise Wi-Fi controller and radar head end

Both are defined adapter types with their source types (enterprise access point, radar) already in the catalogue, the feasibility engine and the coverage model. A connector is built to the interface above when a vendor signs; for a vendor with a REST cloud the gateway vendor cloud adapter and a field map are usually enough.

Integration points

Admin actions

- **Test connection.** Checks the adapter's settings are complete, then makes a live call to the vendor (for a REST vendor, the raw source list and one page of results from the last hour, reported through the field map so mapping errors are visible). Writes the time and result on the adapter; never changes its health status, which the cycle owns.
- **Discover.** Imports the sources the vendor can see as source records on the chosen site, matched by the vendor's source id so re-running updates rather than duplicates. Sources can then be linked to the inventory device, antenna, RAN node or network function that carries them.

The results path

- **Queue.** Results leave the runtime on a regional queue (batches of 100, per-item failure reporting, dead-letter queue). A message that cannot be mapped or written is reported back for redelivery, never dropped silently.
- **Store.** Each result is written to the tenant's own sensing table in the tenant's home region: partition key zone and hour, sort key observation time and id, and a time to live set from the zone's retention (default 30 days). The id is a hash of the result, so a redelivered message overwrites the same item.
- **Summary.** One aggregate summary per zone and minute is kept in the tenant's main table for the zone screens; aggregate-only zones get no track or event rows anywhere outside the store.

- **Live capabilities.** From the feasibility screen, asks each source's adapter for current figures, with a 10 second limit per adapter so one slow vendor cannot hold the others up. Read only.

Sources in the inventory

A source record carries site, type, vendor, adapter, the vendor's source id, capabilities, modes, timing accuracy, backhaul latency, firmware, position, height, azimuth, beam width and range, status (planned, online, degraded, offline, decommissioned) and last seen. Linking a source to its device, antenna, RAN node or network function is what lets a network change show which sensing zones it affects.

- **Live.** The last 20 results per zone are pushed to the open zone screen.
- **Downstream.** Assurance reads the store every 15 minutes to measure KPIs; results that fall inside a zone's shape and height band raise webhook events (boundary alert, zone breach, fall); the API, portal and reports read the same store and samples.

Security

- Credentials are secrets-vault records encrypted with a tenant-scoped key and decrypted only inside a run or an admin call.
- The tenant for every admin call comes from the caller's own user record, never from the request body.
- Adapter and runtime code never log a credential, a token or a raw vendor payload; errors are plain sentences with an HTTP status.
- Every credential use, task start and stop, and every read and write of sensing data is written to the tenant's audit log.

Connecting a new sensing source

Step	Who	What is needed
1. Choose the adapter type	Operator, DemandFlow	REST cloud, NEF sensing function, 6G draft, or a new connector for a vendor without a REST API.
2. Provide access	Vendor or operator	Endpoint, authentication method and a credential for a test tenant; for NEF, an AF id and the notification endpoint to register; for 6G trials, the exposure API root.
3. Describe the data	Vendor	API documentation or sample JSON for sources, task start and stop, and results, including timestamps, ids and state words.
4. Configure	DemandFlow or operator admin	Create the adapter record, link the credential, enter paths and the field map, and run Test connection until the report maps cleanly.
5. Discover and link	Operator admin	Import sources to their sites and link them to the inventory equipment, with timing and backhaul figures where the vendor gives them.
6. Prove	Operator	Draw a zone, run feasibility, activate, watch live results, and let assurance measure a day of KPIs before selling against the tier.

www.demandflow.com
Integration and partner enquiries: www.demandflow.com/sensing-service-management/

DemandFlow® is a registered trademark of TOCLOCO LTD. Registered in England No. 09950859. 71-75 Shelton St, London, WC2H 9JQ. © TOCLOCO LTD. All rights reserved.