UZ801 LTE MiFi dongle — ADB auto-enable (host side)

Cheap ~$10 Qualcomm LTE MiFi dongles (`05c6:90b6`, "Android"/UZ801, msm8916) boot in **RNDIS mode** with a web UI at `http://192.168.100.1`. GETting `http://192.168.100.1/usbdebug.html` flips them into **ADB mode** (and they come up already-root). Ref: https://nickvsnetworking.com/adventures-with-a-10-lte-mifi-dongle/

The enable step has to run on the **USB host** (the Proxmox hypervisor), because only the host has the RNDIS link to `192.168.100.1` — the OmniWeb phone-agent runs in the unprivileged LXC (CT 102) and reaches the dongle only as a USB device once it is in ADB mode. So this automation is host-side udev + systemd, not agent code.

Files

- `mifi-adb-enable.sh` → `/usr/local/sbin/mifi-adb-enable.sh` (0755)
- `mifi-adb-enable.service` → `/etc/systemd/system/mifi-adb-enable.service`
- `99-mifi-adb-autoenable.rules` → `/etc/udev/rules.d/99-mifi-adb-autoenable.rules`

```
sudo install -m0755 mifi-adb-enable.sh /usr/local/sbin/mifi-adb-enable.sh
sudo install -m0644 mifi-adb-enable.service /etc/systemd/system/mifi-adb-enable.service
sudo install -m0644 99-mifi-adb-autoenable.rules /etc/udev/rules.d/99-mifi-adb-autoenable.rules
sudo systemctl daemon-reload && sudo udevadm control --reload
```

Flow: dongle plugged (RNDIS) → udev fires → `mifi-adb-enable.service` → script brings up the RNDIS iface, curls `usbdebug.html` → dongle reboots into ADB mode → re-enumerates with the ADB interface → the CT-102 udev rule (below) hands the node to the container → the agent's `adb` sees it. On the adb-mode re-enumeration the script runs again, detects the ADB interface (`ff/42/01`) and exits — no loop.

Companion CT-102 USB-passthrough udev change (REQUIRED)

The dongle's Qualcomm VID `05c6` must be handed to the container, and the group **must be plugdev (host gid 100046)**, not container-root, or the agent's adb (user `omniweb-agent`, a plugdev member) cannot open the node. The same applies to the Android phone rules: they were `GROUP="100000"` (container root) which the agent cannot access — devices only worked while adb held the fd from enumeration time and dropped on every re-enumeration. Fixed to `GROUP="100046"`.

In `/etc/udev/rules.d/99-ct102-android.rules` all phone rules use `OWNER="100000", GROUP="100046", MODE="0660"`, plus the dongle VID:

```
SUBSYSTEM=="tty", SUBSYSTEMS=="usb", ATTRS{idVendor}=="05c6",
OWNER="100000", GROUP="100046", MODE="0660"
SUBSYSTEM=="usb", ATTR{idVendor}=="05c6", OWNER="100000",
GROUP="100046", MODE="0660"
```

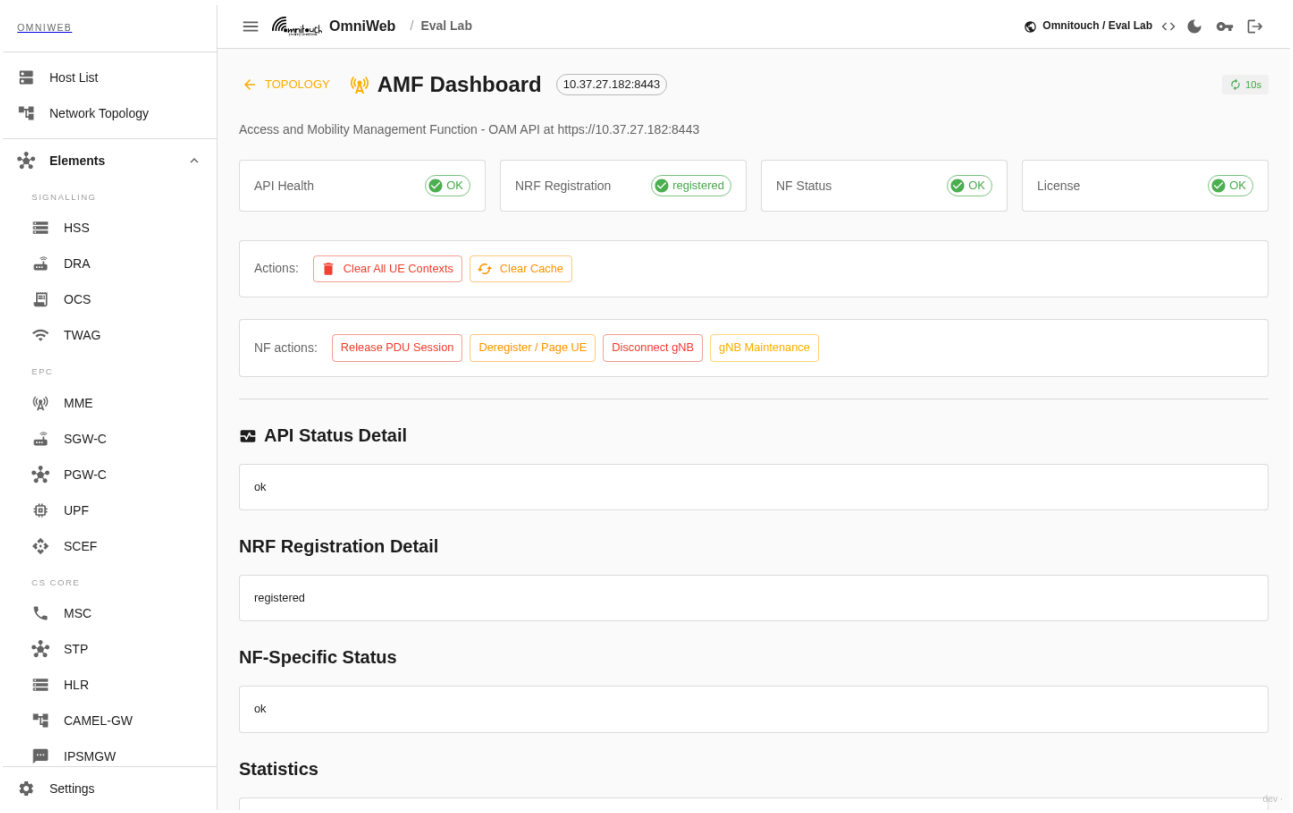
> These host changes are applied live in the lab but should be codified in the OmniCore ansible (monitoring/host role) so they survive a host rebuild.

AMF — Access and Mobility Management Function

The **AMF** is the control-plane anchor of the 5G core. It terminates the **N1** signalling to the UE and the **N2** (NGAP) association to each gNodeB, and handles registration and connection management — attaching UEs, keeping them reachable, and paging them when they are idle. It owns the **UE contexts** (one per registered subscriber, keyed by SUPI) and the **gNB associations** (one per connected radio, keyed by SCTP association), and it selects the SMF for each PDU session while relaying session requests between the UE and the SMF. When a subscriber is on the network, the AMF is the NF that knows where they are and how to reach them.

← 5G Core

In the dashboard



The AMF dashboard opens with the four standard health cards — **API Health**, **NRF Registration** (whether the AMF is currently registered with the NRF), **NF Status**, and **License** — each polling on an interval. Below them, it surfaces the data the AMF owns:

- **UE List** — every UE context the AMF holds, keyed by SUPI, with its registration and connection state. This is the "who is attached right now" view. Selecting a UE opens its detailed context, and the AMF can also list a UE's active PDU sessions by SUPI.
- **gNodeBs** — the connected radios, one row per SCTP association. The table decodes the raw NGAP values into readable **gNB-ID**, **TAC** (tracking area), **PLMN**, and **SST** (slice/service type), so an operator can confirm which cells are up and which tracking areas and slices they serve without decoding identifiers by hand. Selecting a row opens the detailed gNB by its SCTP association ID.
- **Statistics** — the AMF's live counters (registered UEs, connected gNBs, and signalling volumes), for a quick read on load and activity.

- **Maintenance Mode** — whether the AMF is in maintenance, used to drain or hold a node during planned work.
- **Configuration** — the runtime configuration the service is currently using.

Actions

The dashboard exposes actions to recover from stuck state. Each is confirmed before it runs, since they clear live state:

- **Clear All UE Contexts** (bulk) — removes every UE context from the AMF. Use as a last resort to recover a node holding stale or corrupt contexts; every affected UE must re-register.
- **Clear Cache** (bulk) — clears the AMF's NF-discovery cache and UE caches. Use when the AMF is acting on stale peer or lookup data — for example, after an NF has moved or re-registered.
- **Deregister** (per-UE) — deregisters a single UE and cleans up its state. Use to drop one subscriber's context that will not clear on its own, rather than resetting the whole node.

Beyond the dashboard buttons, the AMF's OAM API exposes finer-grained recovery controls an operator can reach through the backend proxy:

- **Force-disconnect a gNB** — tears down a single gNB by its SCTP association ID. Use to clear a wedged or misbehaving radio association without touching the others.
- **Force-release a PDU session** — releases one PDU session by its `supi:psi` identifier. Use to drop a session that is stuck on the AMF's side while leaving the UE registered.
- **Deregister or page a UE** — forces a UE to deregister, or triggers paging to bring an idle UE back to connected. Use paging to test reachability, and forced deregistration to evict a UE cleanly.

Operations

- **Runtime diagnostics** (`/api/oam/diagnostics`) — shows low-level diagnostics for the running service.

- **Resource usage** (`/api/oam/resources`) — reports the runtime's memory, process, and scheduler usage for a health check on the node.
- **Peers** (`/api/oam/peers`) — lists the configured and discovered NF peer URIs and the current SCTP associations, so an operator can see who the AMF is talking to.
- **NRF discovery cache** (`/api/oam/nrf_discovery`) — shows the contents of the discovery cache — which NF instances the AMF has learned from the NRF.
- **Set log level** (`/api/oam/log_level`) — changes the log level at runtime (one of `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, `debug`) to turn up detail while troubleshooting.
- **Force NRF re-registration** (`/api/oam/nrf/reregister`) — re-registers the AMF with the NRF, used to recover a dropped or stale registration.

Related

- SMF — Session Management Function — the AMF selects the SMF and relays PDU session requests to it.
- UDM — Unified Data Management — the subscriber data the AMF uses during registration.
- AUSF — Authentication Server Function — authenticates the UE during registration on the AMF's behalf.
- NSSF — Network Slice Selection Function — resolves slice selection for the AMF.
- ← 5G Core — the overview of all NF dashboards.

OmniWeb Android-control MCP server

An MCP server that wraps the OmniWeb "Android middleware" (the `omniweb-android` Flask service, `/api/android/*`) so an MCP client — Claude Code, Claude Desktop, or a custom client — can drive the lab's Android phones and USB modems.

It is a thin HTTP client over the existing REST API (it imports no Flask internals), so it works against any reachable `omniweb-android` instance.

Tools exposed

Tool	Signature	What it does
`list_devices`	`()`	All known devices (phones + modems), on
`device_info`	`(serial)`	Full details for one device.
`screenshot`	`(serial)` → PNG image	Current screen (universal: scrcpy phones A devices).
`tap`	`(serial, x, y)`	Tap at pixel coordinates.
`swipe`	`(serial, x1, y1, x2, y2, ms=200)`	Swipe/drag/fling.
`input_text`	`(serial, text)`	Type into the focused field.
`key`	`(serial, name)`	Press back/home/recent/enter/del/power/menu/v
`adb_shell`	`(serial, command, timeout=30)` → `{stdout, stderr, return_code}`	Arbitrary `adb shell` (power tool).
`list_scripts`	`()`	The device test-script library.
`run_script`	`(serials, file, params={})`	Fan a script across devices; returns run har

Tool	Signature	What it does
`get_run`	`(run_id)`	Result of a script run (status/stdout/return_
`at_command`	`(serial, command, read_ms=0, timeout_ms=0)`	Raw AT command to a USB modem.
`send_sms`	`(serial, number, text)`	Send an SMS from a modem.

Configuration (environment variables)

Backend target:

- `OMNIWEB_API_BASE` — base URL of the android service. Default `http://127.0.0.1:5002`. The `/api/android` prefix is added automatically.

Backend auth (pick ONE):

- `OMNIWEB_JWT` — a pre-minted backend access token (Bearer), OR
- `OMNIWEB_JWT_SECRET` — the backend's `JWT_SECRET_KEY`. The server then mints a short-lived admin JWT itself and auto-refreshes it. `OMNIWEB_JWT_TTL` (seconds, default 3600) controls its lifetime.

This MCP server's own front door:

- `MCP_TRANSPORT` — `http` (default, streamable HTTP for remote clients) or `stdio` (local `claude mcp add`).
- `MCP_HOST` / `MCP_PORT` — bind address for http transport. Default `127.0.0.1:5005`.
- `MCP_AUTH_TOKEN` — bearer token a remote MCP client must present. **Required to bind to anything other than localhost** — the server

refuses a non-loopback bind without it (fail closed). On `/mcp` the token is checked by middleware; `/` and `/health` are open for health checks.

Run it

venv (self-contained)

```
cd /opt/omniweb-mcp           # or wherever you put server.py + requirements.txt
python3 -m venv venv
venv/bin/pip install -r requirements.txt

export OMNIWEB_API_BASE=http://127.0.0.1:5002
export OMNIWEB_JWT_SECRET=<the backend JWT_SECRET_KEY>
export MCP_TRANSPORT=http
export MCP_HOST=0.0.0.0
export MCP_PORT=5005
export MCP_AUTH_TOKEN=<a long random token>
venv/bin/python server.py
```

systemd (production)

A unit is provided at `packaging/omniweb-android-mcp.service`. It reads `/etc/omniweb/omniweb.env` (for `JWT_SECRET_KEY` → it is re-exported as `OMNIWEB_JWT_SECRET`, see below) plus `/etc/omniweb/android-mcp.env` for the MCP-specific secrets. Create that drop-in (chmod 600):

```
# /etc/omniweb/android-mcp.env
OMNIWEB_JWT_SECRET=<same value as JWT_SECRET_KEY in omniweb.env>
MCP_AUTH_TOKEN=<a long random token>
```

Then:

```
sudo cp packaging/omniweb-android-mcp.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable --now omniweb-android-mcp
```

> The backend env file names the JWT secret `JWT_SECRET_KEY`; this server reads `OMNIWEB_JWT_SECRET`. Set `OMNIWEB_JWT_SECRET` explicitly in `android-mcp.env` (copy the value) so the names line up.

Connect from Claude Code

Remote (streamable HTTP), with auth:

```
claude mcp add --transport http android-control
http://<host>:5005/mcp \
  --header "Authorization: Bearer <MCP_AUTH_TOKEN>"
```

Local (stdio), pointing the server at a backend and minting its own JWT:

```
claude mcp add android-control \
  -e OMNIWEB_API_BASE=http://127.0.0.1:5002 \
  -e OMNIWEB_JWT_SECRET=<backend JWT_SECRET_KEY> \
  -e MCP_TRANSPORT=stdio \
  -- /opt/omniweb-mcp/venv/bin/python /opt/omniweb-mcp/server.py
```

Then in Claude Code: `/mcp` to confirm it connected, and try `list_devices`.

Connect from Claude Desktop

`claude_desktop_config.json` — stdio launch:

```
{
  "mcpServers": {
    "android-control": {
      "command": "/opt/omniweb-mcp/venv/bin/python",
      "args": ["/opt/omniweb-mcp/server.py"],
      "env": {
        "OMNIWEB_API_BASE": "http://127.0.0.1:5002",
        "OMNIWEB_JWT_SECRET": "<backend JWT_SECRET_KEY>",
        "MCP_TRANSPORT": "stdio"
      }
    }
  }
}
```

For a remote streamable-HTTP server from Claude Desktop, use an `mcp-remote` bridge or the HTTP server config supported by your Desktop version, pointing at `http://<host>:5005/mcp` with the `Authorization: Bearer <MCP_AUTH_TOKEN>` header.

Security notes

- Device control requires the configured `MCP_AUTH_TOKEN` on the HTTP transport. With no token the server only allows a loopback bind and prints a warning; a non-loopback bind without a token is refused.
- The server authenticates to the backend as an admin (minted JWT or supplied token), so anyone who can reach `/mcp` with the bearer token has full device control. Treat `MCP_AUTH_TOKEN` like a password and front it with TLS (e.g. via nginx) for off-host access.

AUSF — Authentication Server Function

The **AUSF** authenticates subscribers on behalf of the AMF. When a UE attaches, the AMF asks the AUSF to run 5G-AKA (or EAP-AKA') for that subscriber; the AUSF works with the **UDM** to obtain the key material, derives the anchor key returned to the AMF, and confirms the authentication result. It holds an active authentication session for each in-flight or established authentication so its progress and outcome can be tracked and, if needed, cleared.

← 5G Core

In the dashboard

The AUSF dashboard opens with the standard health cards across the top — **API Health**, **NRF Registration** (is the AUSF registered with the NRF?), **NF**

Status, and **License** — each polling on an interval. Beneath them are the AUSF's data sections, read live from its OAM API through the OmniWeb proxy:

- **Statistics** — the AUSF's own counters (`AUSF Statistics`), giving a running view of authentication activity.
- **Active AUSF authentication sessions** — the list of authentication sessions the AUSF currently holds (`List Active AUSF Authentication Sessions`). Each row is keyed by its session identifier, so an operator can see which authentications are in flight or established at a glance.

Actions

Both actions clear authentication state and ask for confirmation first, since they are destructive:

- **Clear a specific authentication session** (per-row) — removes a single authentication session by its identifier. Use this to drop one stuck or stale session without disturbing every other subscriber's authentication.
- **Clear All Auth Sessions** — removes every authentication session the AUSF holds. Use this to recover the AUSF from a widespread stuck state; each affected subscriber re-authenticates on its next attempt.

Operations

Standard OAM controls available on the AUSF:

- **Diagnostics** (`/api/oam/diagnostics`) — runtime diagnostics for the service.
- **Resources** (`/api/oam/resources`) — the service's resource usage.
- **Force NRF re-registration** (`/api/oam/nrf/reregister`) — makes the AUSF re-register with the NRF, to recover if its registration has gone stale.

Related

- AMF — Access & Mobility Management — requests authentication from the AUSF for each attaching UE.
- UDM — Unified Data Management — supplies the key material the AUSF uses to authenticate a subscriber.
- ← 5G Core — the 5GC dashboard overview and the other NFs.

BSF — Binding Support Function

The **BSF** registers and resolves the binding between a UE's PDU session and the PCF that serves it. When a session is established, the BSF records which PCF is handling policy for that session, keyed by the session's identifiers — the UE IP address, the DNN, and the S-NSSAI. Other NFs and application functions then query the BSF to discover the correct PCF for a given UE session, so that policy requests reach the PCF already holding that session's context rather than an arbitrary one. The BSF holds one binding per active PDU session and exposes them, along with summary statistics, for monitoring.

← 5G Core

In the dashboard

The BSF dashboard opens with the standard health cards across the top — **API Health**, **NRF Registration** (is the BSF registered with the NRF?), **NF Status**, and **License** — each polling on an interval. Beneath them are the BSF's data sections, read live from its OAM API through the OmniWeb proxy:

- **PCF Bindings** — the list of active bindings the BSF currently holds (`List all PCF bindings with binding ID, UE IP, DNN, S-NSSAI, PCF FQDN/IP`). Each row is keyed by its binding ID and shows the UE IP address, DNN, and S-NSSAI that identify the session, together with the FQDN or IP of the PCF serving it. This answers "which PCF is handling policy for this UE session?" at a glance. Selecting a binding opens its full detail (`Get detailed PCF binding by binding ID`).
- **Statistics** — the BSF's own counters (`BSF statistics: total bindings, bindings by DNN`), giving a running view of how many bindings are active in total and how they break down per DNN.

Actions

The BSF dashboard is read and monitoring oriented — it surfaces bindings and statistics for inspection and does not expose destructive per-row or NF-level clear actions.

Operations

Standard OAM controls available on the BSF:

- **Force NRF re-registration** (`/api/oam/nrf/reregister`) — makes the BSF re-register with the NRF, to recover if its registration has gone stale.

Related

- PCF — Policy Control — the NF each binding points to; the BSF is how consumers find the right PCF for a UE session.
- SMF — Session Management and application functions — Nbsf consumers that register and look up bindings as sessions come and go.

- ← 5G Core — the 5GC dashboard overview and the other NFs.

CHF — Charging Function

The **CHF** (Charging Function) provides converged online and offline charging for the 5G Core. The SMF consumes it over the N40 interface: as PDU sessions carry traffic, the CHF manages usage quotas, rates that usage through the rating/charging engine, and generates the CDRs (Charge Data Records) that feed billing and reconciliation. Its dashboard surfaces the live charging sessions, the health of the link to the rating engine, and the completed CDR history — so an operator can answer "is a subscriber being charged correctly, and is the rating engine reachable?" without leaving OmniWeb.

← 5G Core

In the dashboard

Across the top sit the standard **health cards** — **API Health**, **NRF Registration** (is the CHF registered with the NRF?), **NF Status**, and **License** — each polling on an interval. Below them the CHF-specific data sections auto-refresh:

- **Active charging sessions** — the live sessions the CHF is metering, each showing its charging reference (`chargingDataRef`), the subscriber (SUPI), the DNN, accumulated usage, and start/update timestamps. This is the working set of "who is being charged right now".
- **Statistics** — a summary view: total sessions, connectivity to the rating/charging engine, and total charged volume.
- **Rating-engine connectivity health** — a dedicated health check confirming the CHF can reach the rating/charging engine that performs the actual rating. If this is unhealthy, sessions cannot be rated and charging is at risk.
- **Completed CDR search** — a searchable view over the finalised CDRs, filterable by SUPI, DNN, and date range, for confirming what was recorded after a session closes.
- **Configuration** — the CHF's runtime configuration as the service currently sees it.

Selecting a session opens its full detail by charging data ref, showing the complete usage and metering state for that one session.

Actions

- **Abort a charging session** — terminates an in-progress charging session by its `chargingDataRef`. Use this to clear a stuck or orphaned session that the SMF is no longer maintaining, so it stops consuming quota and metering state.
- **Search CDRs** — queries the completed CDR history by SUPI, DNN, and date range. Use this after a session ends to confirm what usage was rated and recorded — for billing enquiries, dispute investigation, or spot-checking that charging is being written through correctly.

Operations

- **Set log level** — changes the CHF's log verbosity at runtime (emergency, alert, critical, error, warning, warn, notice, info, debug) to gather more detail while investigating a charging issue, then return to normal.
- **Force NRF re-registration** — makes the CHF re-register itself with the NRF, used to recover if it has fallen out of the registry and the SMF can no longer discover it.

Related

- SMF — the N40 consumer that drives charging as PDU sessions carry traffic.
- PCF — the policy side of policy-and-charging control that governs what the CHF meters against.
- The rating/charging engine (OCS) — the downstream system the CHF rates through; the rating-engine connectivity health card reflects the link to it.
- ← 5G Core — the overview of all NF dashboards.

NRF — Network Repository Function

The **NRF** is the registry the whole 5G core is built around. Every other NF registers its **NF profile** with the NRF on start-up — its type, addresses, PLMNs, the **services** it offers, and its capacity and priority — and then discovers its peers by asking the NRF who is available. The NRF tracks each registered instance's **status**, holds **status-change subscriptions** so an NF can be told when a peer it depends on comes or goes, and watches **heartbeats** to expire instances that stop checking in. When any NF needs to find another, the NRF is the source of truth for what is up and how to reach it.

← 5G Core

In the dashboard

The NRF has its own dedicated dashboard, richer than the standard NF page because it owns the registry the rest of the core discovers through. It opens with the standard health cards — **API Health**, **NRF Registration** (whether the NRF itself is registered), **NF Status**, and **License** — each polling on an interval.

Above the data it shows four **metric cards** summarising the registry: **NF types**, **total instances**, **registered** (with a percent-of-total bar), and **services** (with the subscription count). Below them are the NRF's core views:

- **Registered NF instances** — every registered instance, grouped by **NF type** (AMF, SMF, UDM, PCF, and so on). Each instance card is colour-coded by type and shows its registration status, heartbeat timer, service count, and PLMN. Selecting an instance opens its full **NF profile** — the detailed record the NRF holds, with an **Open Dashboard** link to that NF's own page and a **Deregister** button. This is the "which NFs are up and are they all registered?" view.
- **Registered services** — a flat table of every service each instance advertises, with its versions and status, so an operator can see what capabilities the core is currently offering across all NFs.
- **Subscriptions** — the NF status-change subscriptions currently held, each NF's standing request to be notified when a peer's status changes.
- **Statistics** — the NRF's live counters: total NFs by type, subscriptions, and discovery requests, for a quick read on registry size and lookup activity.

Actions

The dashboard exposes actions to manage the registry. Each is confirmed before it runs, since it changes live registration state:

- **Deregister an NF instance** (per-row) — removes one instance's registration from the NRF. Use to clear a stale or wedged registration for a single NF; that NF must then re-register before other NFs can discover it again.
- **Deregister All NFs** (bulk) — removes every registration from the NRF at once. Use as a last resort to reset a registry holding stale entries; every NF must then re-register.

- **Remove a subscription** — deletes one status-change subscription. Use to clear a stale or orphaned subscription left behind by an NF that no longer needs it.
- **Update an NF profile** — patches a registered instance's profile (for example its capacity, priority, or status) so operators can adjust how the NRF steers discovery towards or away from an instance without deregistering it.

Operations

The NRF's OAM API exposes further controls an operator can reach through the backend proxy:

- **Runtime diagnostics** (`/api/oam/diagnostics`) — shows low-level diagnostics for the running service.
- **Resource usage** (`/api/oam/resources`) — reports the runtime's memory, process, and scheduler usage for a health check on the node.
- **Manual heartbeat check** (`/api/oam/heartbeat`) — triggers an on-demand heartbeat check for an NF instance, to confirm whether it is still alive rather than waiting for its timer.
- **NF uptime** (`/api/oam/nf_uptime/{id}`) — shows an instance's registration time and how long it has been registered, for confirming how long an NF has been up.
- **Set log level** (`/api/oam/log_level`) — changes the log level at runtime (one of `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, `debug`) to turn up detail while troubleshooting.
- **Force NRF re-registration** (`/api/oam/nrf/reregister`) — re-registers the NRF, used to recover a dropped or stale self-registration.

Related

- AMF — Access and Mobility Management Function — registers with the NRF and discovers its peers through it, like every NF below.
- SMF — Session Management Function — discovered through the NRF for session management.

- UDM — Unified Data Management — one of the subscriber-data NFs the NRF holds registrations for.
- PCF — Policy Control Function — registered with the NRF for policy discovery.
- SCP — Service Communication Proxy — routes signalling between NFs using the registrations and discovery data the NRF holds.
- ← 5G Core — the overview of all NF dashboards.

NSSF — Network Slice Selection Function

The **NSSF** selects the network slice instance(s) and the allowed NSSAI for a UE. When the AMF handles a registration or a slice request, it asks the NSSF to resolve the UE's requested S-NSSAIs against the slices configured for that PLMN — mapping each to its network slice instance (NSI) and to the AMF set that serves it — and to return the allowed NSSAI the UE may use. It holds the configured slice definitions and tracks per-AMF NSSAI availability so slice selection can be reasoned about and monitored.

← 5G Core

In the dashboard

The NSSF dashboard opens with the standard health cards across the top — **API Health**, **NRF Registration** (is the NSSF registered with the NRF?), **NF**

Status, and **License** — each polling on an interval. Beneath them are the NSSF's data sections, read live from its OAM API through the OmniWeb proxy:

- **Slices** — every configured S-NSSAI, listed with its NSI mappings, the AMF sets that serve it, and the allowed NSSAI per PLMN (`List all configured S-NSSAIs with NSI mappings, AMF sets, and allowed NSSAI per PLMN`). This is the authoritative view of how requested slices resolve to slice instances and AMF sets for each PLMN.
- **NSSAI Availability** — the per-AMF NSSAI availability state (`Per-AMF NSSAI availability state`), showing which slices each AMF has reported it can currently serve.
- **Statistics** — the NSSF's own counters (`NSSF statistics: total slices, NS selection request counts`), giving a running view of the number of configured slices and of slice-selection request volume.

A separate **slice configuration** view (`Query Slice Configuration`) surfaces the NSSF's runtime slice configuration for reference.

Actions

The NSSF dashboard is read and monitoring oriented — it surfaces slices, NSSAI availability, and statistics without per-item control actions.

Operations

The OAM controls the NSSF exposes are:

- **View runtime configuration** (`/api/oam/config`) — the NSSF's live runtime configuration.
- **Force NRF re-registration** (`/api/oam/nrf/reregister`) — makes the NSSF re-register with the NRF, to recover if its registration has gone stale.

Related

- AMF — Access & Mobility Management — queries the NSSF to resolve requested S-NSSAIs into the allowed NSSAI and serving AMF set for a UE.
- NRF — Network Repository Function — where the NSSF registers so other NFs can discover it.
- ← 5G Core — the 5GC dashboard overview and the other NFs.

PCF — Policy Control Function

The **PCF** makes the policy decisions the rest of the 5G Core enforces. It answers the AMF over N15 for **access & mobility (AM) policy** — the mobility restrictions and RFSP a UE gets on registration — and the SMF over N7 for **session-management (SM) policy** — the PCC rules that set each session's QoS, gating, charging control and usage monitoring. The PCF binds a UE's SM policy associations back to itself through the BSF so any NF can find the right PCF for a session. The PCF dashboard is the place to answer "what policies are live, for which UEs and sessions, and are they what I expect?" without leaving OmniWeb. Every OAM response is wrapped `{status, response}`; the dashboard unwraps it for you.

← 5G Core

In the dashboard

Across the top sit the four standard health cards — **API Health**, **NRF Registration** (is the PCF registered with the NRF?), **NF Status**, and **License** — each polling on an interval. Use the polling indicator in the header to pause refresh or read the last-update time. Below the cards is the **Actions** bar (see below), then the PCF's data sections, each auto-refreshing.

Statistics — the PCF's live counters (`GET /api/statistics`): the policy-association totals and current counts. This is the quickest read on whether the PCF is actively serving AM and SM policy.

AM Policies — every active AM policy association (`GET /api/am_policy`), one row per UE the AMF has an AM policy for. This is the PCF's view of the mobility policy currently applied to each registered UE.

SM Policies — every active SM policy association (`GET /api/sm_policy`), one row per PDU session the SMF holds a policy for. Select a row to open its detail (`GET /api/sm_policy/{id}`), which returns the full PCC rules on that session — the QoS, gating, charging and usage-monitoring decisions the SMF is enforcing.

Policy Context — the active policy contexts view (`GET /api/policy_context`): the contexts the PCF holds across its AM and SM associations, useful for reconciling what the PCF believes is live against the AMF and SMF.

Configuration — the PCF's runtime configuration (`GET /api/oam/config`): the settings the service is currently running with.

Actions

Each per-row action asks for confirmation first, since removing a live association is disruptive to the UE or session it affects.

Remove an AM policy — on any AM Policies row, **Remove** deletes that AM policy association (`DELETE /api/am_policy/{id}`). Use it to clear a stuck or stale mobility policy for one UE.

Remove an SM policy — on any SM Policies row, **Remove** deletes that SM policy association (`DELETE /api/sm_policy/{id}`). Use it to drop a session's policy that will not clear on its own.

Update an SM policy — an SM policy association is also editable. Sending `PATCH` or `PUT /api/oam/sm_policy/{id}` updates the PCC rules on a live SM policy in place, so you can adjust a session's QoS or charging decisions without tearing the association down. Use it to re-drive policy onto an existing session rather than removing and re-creating it.

Operations

The Operations section carries the controls common to every NF:

- **Diagnostics** (`GET /api/oam/diagnostics`) — runtime diagnostics for the service.
- **Resource usage** (`GET /api/oam/resources`) — memory, processes and schedulers of the running service.
- **Peers** (`GET /api/oam/peers`) — the configured NF peer URIs the PCF talks to and their reachability status.
- **NRF discovery cache** (`GET /api/oam/nrf_discovery`) — the NRF discovery cache, or the configured NF URIs the PCF resolves peers through.
- **Log level** (`POST /api/oam/log_level`) — change the log level at runtime; one of emergency, alert, critical, error, warning, warn, notice, info, or debug.
- **NRF re-registration** (`POST /api/oam/nrf/reregister`) — force the PCF to re-register with the NRF, e.g. after an NRF restart left it stale.

Related

- SMF — requests SM policy from the PCF over N7 and enforces the PCC rules the PCF returns.
- AMF — requests AM policy from the PCF over N15.
- BSF — the binding function the PCF registers SM policy associations with so any NF can find the serving PCF.
- CHF — the charging function; the PCF's charging-control decisions drive what the SMF reports over N40.
- ← 5G Core overview

SCP — Service Communication Proxy

The **SCP** sits between the network functions of the 5G Core and carries their service-based traffic for them. Rather than every consumer NF connecting directly to every producer, requests can be routed through the SCP, which forwards each message to a healthy instance of the target NF type, load-balancing across instances and re-trying around failures. It can perform discovery on a consumer's behalf (delegated discovery) and caches NF reachability so that, once a target type is known, later requests are steered without a fresh lookup. The result is indirect communication: producers and consumers do not need to know or hold connections to one another, and the SCP handles routing, balancing, and cache freshness in between.

← 5G Core

In the dashboard

The SCP dashboard follows the same layout as every 5GC NF page. Four **health cards** run across the top — **API Health**, **NRF Registration** (is the SCP itself registered with the NRF?), **NF Status**, and **License** — each polling on an interval.

Below the cards, the SCP surfaces its own operational data:

- **NF Health** — the load-balancer's view of the NFs it proxies to: per-instance health and failure counts, so an operator can see which producer instances the SCP considers reachable and which are failing.
- **Discovery Cache** — the contents of the SCP's discovery cache: which NF types are cached, their TTLs, and the cache hit/miss rates. This is what lets the SCP steer repeat requests without re-discovering.
- **Statistics** — SCP throughput: total proxied requests, a breakdown of requests by target NF type, and the overall cache hit rate.
- **Configuration** — the SCP's current runtime configuration.

Each section auto-refreshes, and NF-type values cross-link to that NF's own dashboard.

Actions

The SCP dashboard does not expose destructive NF-level or per-row action buttons — it proxies traffic rather than owning session or subscriber state that an operator would clear. Its cache, however, is manageable: the OAM API lets an operator view the discovery cache and evict a single stale entry so the SCP re-discovers that target on the next request (see **Operations**). Clearing an entry is the way to recover from a producer that has moved or been replaced but is still being cached as reachable.

Operations

The SCP's OAM API exposes the following controls:

- **View / manage discovery cache** — read the current discovery cache, and delete an individual cache entry by id to force re-discovery of that

target.

- **View runtime configuration** — read the SCP's live configuration.
- **Set log level** — change the log verbosity at runtime, one of `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, or `debug`.
- **Force NRF re-registration** — make the SCP re-register itself with the NRF, useful after the NRF or the SCP has restarted and the registration has gone stale.

Related

- NRF — the registry the SCP discovers through and re-registers with.
- AMF, SMF, UDM, AUSF, PCF, CHF, NSSF, BSF — the NFs whose service-based traffic the SCP can proxy and whose reachability it caches.
- ← 5G Core — the 5GC section overview and the shared dashboard layout.

SMF — Session Management Function

The **SMF** owns the PDU session — it establishes, modifies and releases each UE's data session, allocates the UE its IP address from a local pool, and drives the user plane over N4/PFCP by programming the UPF with the packet-detection, forwarding and QoS rules for every session. It also talks to the PCF (N7) for session policy and the CHF (N40) for charging. The SMF dashboard is the place to answer "are sessions coming up, is the SMF still associated with its UPF, and is it running out of addresses?" without leaving OmniWeb. Every OAM response is wrapped `{status, response}`; the dashboard unwraps it for you.

← 5G Core

In the dashboard

Across the top sit the four standard health cards — **API Health**, **NRF Registration** (is the SMF registered with the NRF?), **NF Status**, and **License** — each polling on an interval. Use the polling indicator in the header to pause refresh or read the last-update time. Below the cards is the **Actions** bar (see below), then the SMF's data sections, each auto-refreshing.

Statistics — the SMF's live counters (`GET /api/statistics`): session establishment / modification / release totals and current session counts. This is the quickest read on whether sessions are actually being set up.

PDU Sessions — every active PDU session (`GET /api/pdu_session`), returned as `{ pdu_sessions: [...], total }`. Each row carries both a `sm_context_ref` (a UUID that uniquely identifies the SM context) and a `pdu_session_id` (the 3GPP PSI — a small integer that is **not** unique across UEs, since each UE numbers its own sessions from a low base). For this reason every per-session operation — detail lookup and release — is keyed on `sm_context_ref`, and the dashboard's per-row **Release** action passes that UUID rather than the PSI. Alongside the identifiers a row surfaces the session's DNN, allocated UE IP and state.

IP Pool — the address pool the SMF allocates UE IPs from (`GET /api/ip_pool/detail`): the pool summary (subnet, total, allocated and available counts) plus the list of currently allocated IPs. This is where you see whether the SMF is close to exhausting its address space.

UPF Association — the N4/PFCP link state to the user plane (`GET /api/upf`): whether the SMF is currently associated with its UPF. If sessions are failing to establish, this is the first thing to check.

PFCP Peers — the PFCP peer (UPF) association status (`GET /api/oam/pfcp_peers`): the SMF's view of each UPF it associates with over N4, including association state.

Actions

Each action asks for confirmation first — several are destructive to live sessions.

Release All PDU Sessions — tears down every PDU session and its PFCP state on the UPF. Use it to clear the SMF back to a clean slate, e.g. after a UPF swap or when draining a node for maintenance.

Clear Cache — clears the SMF's session caches without releasing sessions. Use it when the dashboard or the SMF's internal view looks stale or inconsistent with reality.

PFCP Reassociate — re-establishes the PFCP association with the UPF (SMF-initiated). Use it when the UPF association has dropped and new sessions cannot establish.

Re-associate UPF — forces a PFCP re-association with the **configured** UPF. The UPF address is config-driven — there is no add-UPF or inject-UPF action here — so this re-drives the association to the UPF already in the SMF's configuration. In practice it serves the same recovery purpose as PFCP Reassociate.

Release by DNN — type a DNN into the editable **DNN** field (e.g. `internet`) and this releases **all** sessions on that DNN. Use it to drain or reset a single data network — say to force UEs off one APN/DNN — without touching sessions on other DNNs.

Per-row Release — on any PDU Sessions row, **Release** tears down that one session and its PFCP context on the UPF. It releases by `sm_context_ref`, so it always targets exactly the session in that row even though several UEs may share the same `pdu_session_id`. Use it to drop a single stuck session.

Operations

The Operations section carries the controls common to every NF:

- **Diagnostics** (`GET /api/oam/diagnostics`) — runtime diagnostics for the service.
- **Resource usage** (`GET /api/oam/resources`) — memory, processes and schedulers of the running service.
- **Peers** (`GET /api/oam/peers`) — the configured NF peer URIs the SMF talks to (AMF, PCF, CHF, NRF, and its PFCP UPF peers via PFCP Peers above).

- **Log level** (`POST /api/oam/log_level`) — change the log level at runtime; one of emergency, alert, critical, error, warning, warn, notice, info, or debug.
- **NRF re-registration** (`POST /api/oam/nrf/reregister`) — force the SMF to re-register with the NRF, e.g. after an NRF restart left it stale.

Related

- AMF — requests session setup from the SMF over Nsmf.
- PCF — supplies session policy to the SMF over N7.
- CHF — the charging function the SMF reports usage to over N40.
- UPF — the SMF's N4/PFCP peer; the user plane the SMF programs and associates with.
- ← 5G Core overview

UDM — Unified Data Management

The **UDM** is the 5G core's subscription data authority. It manages the subscription profiles for every subscriber, tracks which serving NFs are handling each UE (registering the AMF and SMF against a subscriber), and provides the authentication data used to challenge the UE. It reads the underlying subscriber records from the **UDR**, its backing store, and caches them for fast lookup. The **AMF** consults it during registration, the **SMF** during session setup, and the **AUSF** draws its authentication material from it. When another NF needs to know a subscriber's profile or where they are being served, the UDM is the NF that answers.

← 5G Core

In the dashboard

The UDM dashboard opens with the four standard health cards — **API Health**, **NRF Registration** (whether the UDM is currently registered with the NRF), **NF Status**, and **License** — each polling on an interval. Below them, it surfaces the data the UDM owns:

- **UE List** — every UE context the UDM holds, keyed by SUPI. This is the "which subscribers the UDM is currently serving" view. The UDM can also return a detailed context for a single subscriber by SUPI, showing that subscriber's serving-NF registrations and cached subscription data.
- **Statistics** — the UDM's live counters, for a quick read on load and activity.
- **Configuration** — the runtime configuration the service is currently using.

Actions

The dashboard exposes actions to clear cached subscriber data. Each is confirmed before it runs, since it drops live state:

- **Clear Subscriber Cache** (bulk) — clears the entire subscriber data cache. Use when the UDM is serving stale profiles — for example, after subscription records have changed in the UDR and the UDM should re-read them rather than serve its cached copies.

Beyond the dashboard button, the UDM's OAM API exposes a finer-grained control an operator can reach through the backend proxy:

- **Purge one UE's cache** (`/api/oam/ue_cache/{id}`) — drops the cached subscriber data for a single subscriber by SUPI. Use to force a re-read for one subscriber whose profile has changed, rather than clearing the cache for everyone.

Operations

- **Runtime diagnostics** (`/api/oam/diagnostics`) — shows low-level diagnostics for the running service.
- **Resource usage** (`/api/oam/resources`) — reports the runtime's memory, process, and scheduler usage for a health check on the node.

- **Set log level** (`/api/oam/log_level`) — changes the log level at runtime (one of `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, `debug`) to turn up detail while troubleshooting.
- **Force NRF re-registration** (`/api/oam/nrf/reregister`) — re-registers the UDM with the NRF, used to recover a dropped or stale registration.

Related

- UDR — Unified Data Repository — the backing store the UDM reads subscriber records from.
- AUSF — Authentication Server Function — draws its authentication material from the UDM.
- AMF — Access and Mobility Management Function — consults the UDM during registration and registers as a serving NF.
- SMF — Session Management Function — consults the UDM during session setup and registers as a serving NF.
- ← 5G Core — the overview of all NF dashboards.

UDR — Unified Data Repository

The **UDR** is the persistent store of subscription, policy and application data that sits behind the UDM and PCF. Where the UDM and PCF are the front-end network functions that other NFs talk to, the UDR is the database they read and write — the single source of truth for who a subscriber is and what they are entitled to. In the Omnitouch core the UDR shares its subscriber store with the **HSS**, so 4G and 5G subscribers are held in one unified repository rather than two parallel databases.

← 5G Core

In the dashboard

The UDR dashboard opens with the standard NF health cards across the top — **API Health**, **NRF Registration** (is the UDR registered with the NRF?), **NF**

Status, and **License** — each polling on an interval.

Below the cards are the two UDR-specific data sections:

- **HSS Health** — a connectivity health check confirming the UDR can reach the shared HSS subscriber store it fronts. This answers "is the subscriber database reachable and healthy?" at a glance.
- **Subscribers** — the subscriber list drawn from the HSS, paginated and searchable by **IMSI**. Selecting a subscriber returns the full subscriber detail for that IMSI, so an operator can confirm a subscriber's provisioning without leaving OmniWeb.

Actions

The UDR dashboard exposes no destructive NF-level or per-row actions — it is a read and lookup view over the shared subscriber store. Its value is in interrogation: check that the store is reachable via the HSS health section, list subscribers, and drill into any one subscriber's full detail by IMSI.

Operations

Alongside the data sections, the UDR carries the operations common to every 5GC network function:

- **Set log level** — raise or lower the runtime log verbosity (`emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, `debug`) via `/api/oam/log_level`, for turning up detail while diagnosing an issue.
- **Force NRF re-registration** — push the UDR to re-register with the NRF (`/api/oam/nrf/reregister`) if its registration has gone stale.
- **View runtime configuration** — inspect the UDR's live configuration through the OAM config endpoint.

Related

- UDM — Unified Data Management — the front-end network function that serves subscription data out of this store.
- PCF — Policy Control Function — reads the policy data held in the UDR.
- HSS — the shared subscriber database the UDR unifies with, so 4G and 5G subscribers live in one repository.
- ← 5G Core — the full 5GC dashboard overview.

API Reference (OpenAPI / Swagger)

OmniWeb's controller (central backend) exposes a single REST API behind the same JWT login that governs the rest of the portal. That API is self-documenting:

- **Swagger UI:** `/api/docs` — browse every endpoint, expand a route to see its method, path/query parameters, whether it needs a token, and its summary. Click **Authorize**, paste a bearer JWT, and you can try calls live against the running instance.
- **OpenAPI 3 spec:** `/api/openapi.json` — the machine readable document, for importing into Postman/Insomnia or generating clients.

← Back to Operations Guide

What's documented

The spec is **generated at runtime** by introspecting the live route map, so it always matches the endpoints the running instance actually serves — there is

no hand-maintained file to drift. For every route it captures:

- the HTTP method and path (with path/query parameters),
- a summary and description taken from the code,
- whether a **bearer JWT** is required, and
- where applicable, the per-element **view/control** permission the route enforces.

Endpoints are grouped by area (Auth, Admin, SIM Bank, Remote SIM, APDU, Test Devices, IR.38, RAEX, TAP, and so on) so you can find a feature's routes quickly.

Getting a token

Most endpoints require a bearer token. Obtain one with `POST /api/auth/login` (the same credentials you use for the portal), then send it as `Authorization: Bearer <token>`. In Swagger UI, use the **Authorize** button so every "Try it out" call carries the token automatically.

Scope

This documents the **controller** API only. The device-agent that runs at remote sites is not part of this API: it dials *out* to the controller over a WebSocket and exposes no inbound REST endpoints, so there is nothing to call on it directly.

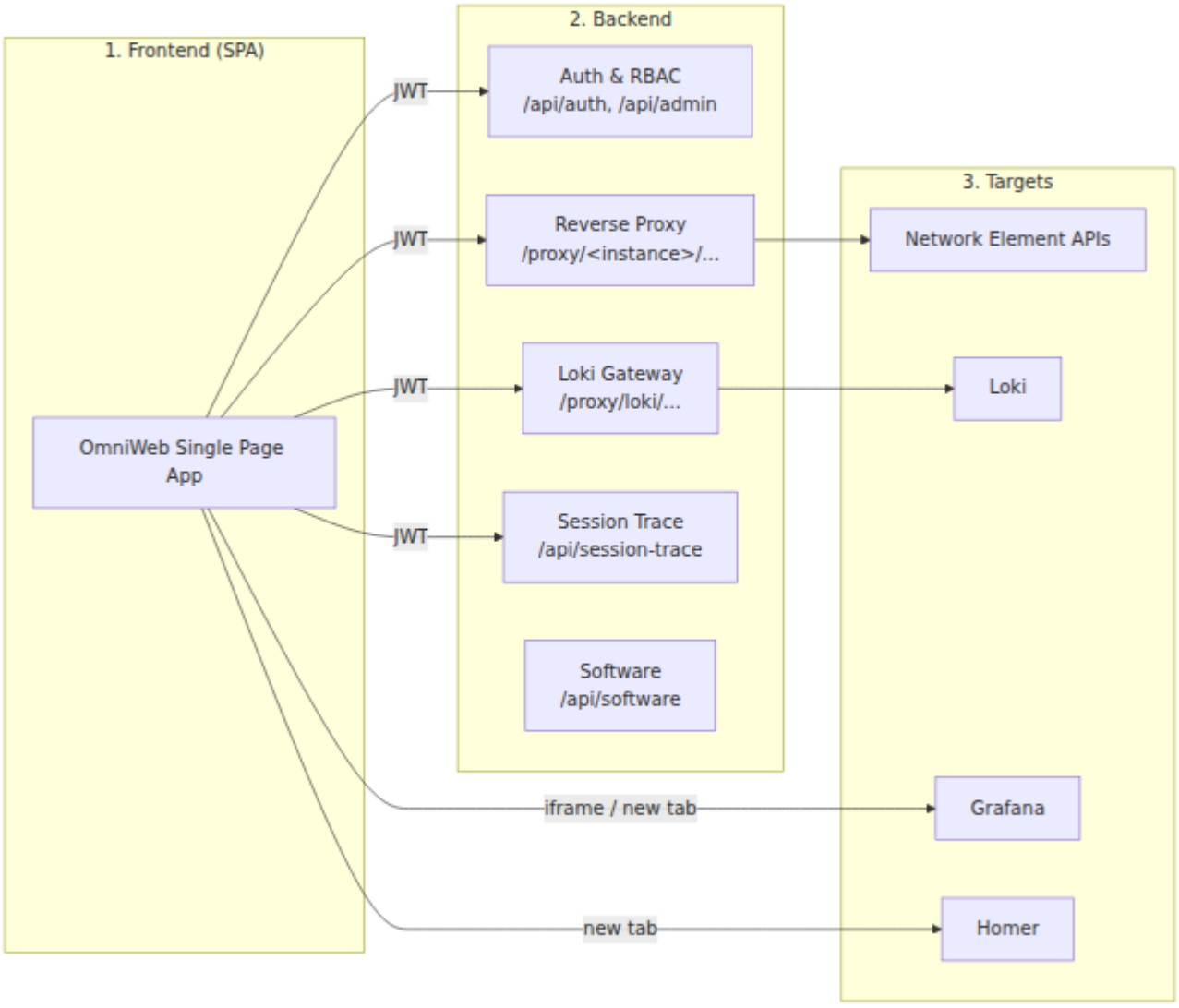
OmniWeb Architecture

This page explains how OmniWeb is put together and how a single browser session reaches every network function, Grafana, and Loki. If you only read one page to understand the system, read this one.

← Back to Operations Guide

Components

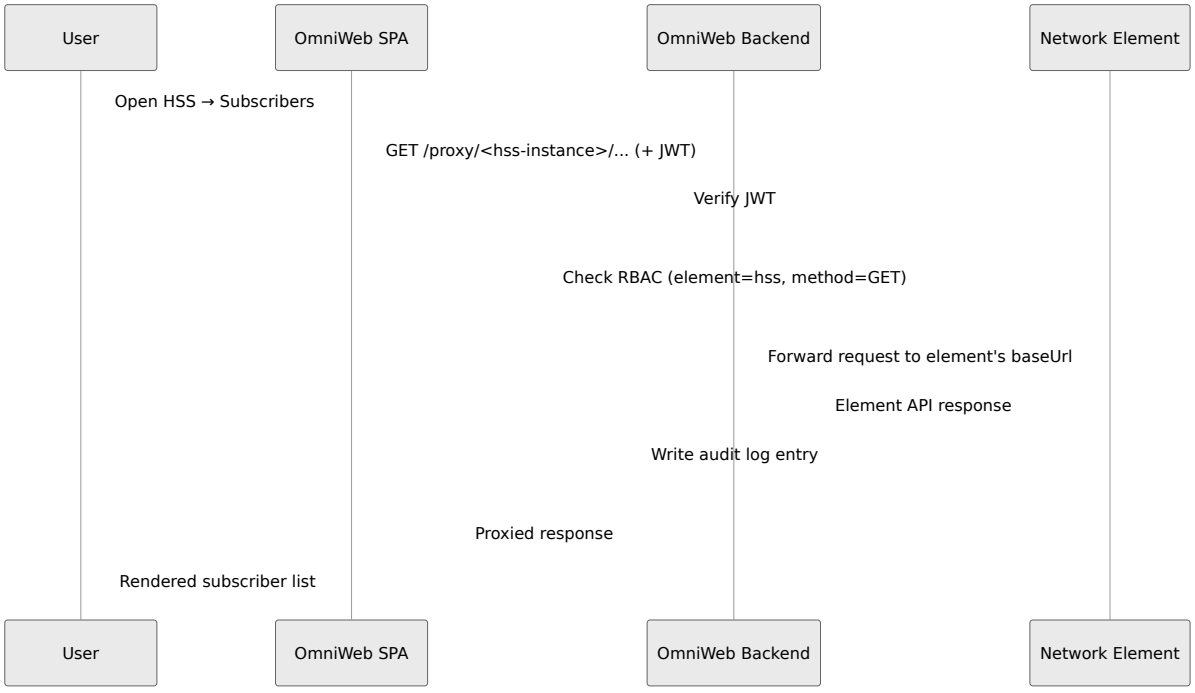
OmniWeb has three moving parts:



Component	Role
Frontend SPA	The user interface. A single-page web application that renders element pages, topology, logs, traces, and embedded dashboards.
Backend	The security and integration layer. It authenticates users, enforces RBAC, and reverse-proxies every request to the correct element API, Loki, or Grafana.
Targets	The things being managed and observed: the network elements' own management APIs, plus Grafana, Loki, and Homer.

Request Flow

Every operational request follows the same path. Nothing in the browser talks to an element directly — it always goes through the backend so that authentication, authorisation, and auditing are enforced centrally.



The proxy path encodes the **instance ID**, so OmniWeb can manage many instances of the same element type (e.g. several HSS nodes) and route each

request to the right backend.

The Configuration Model

OmniWeb is **configuration-driven**. The set of elements, their instances, and where their APIs live is declared in a single file: `public/config/omniweb.json`. The backend loads it at startup and the frontend uses it to build the sidebar, topology, and proxy targets.

```
{
  "elements": [
    {
      "type": "hss",
      "displayName": "HSS",
      "description": "Home Subscriber Server",
      "instances": [
        {
          "id": "my-hss-01",
          "name": "my-hss-01",
          "baseUrl": "https://10.0.0.1:8443",
          "tags": ["primary"]
        }
      ],
      "defaults": {
        "pollIntervalMs": 5000,
        "timeoutMs": 10000
      }
    }
  ]
}
```

Parameter	Type	Required	Default	Description
`type`	String	Yes	-	Element type. Determines which UI module renders the element. See supported types.
`displayName`	String	Yes	-	Human-readable name shown in the sidebar and topology.
`description`	String	No	-	Longer description shown in tooltips and overview pages.
`instances`	List	Yes	-	One entry per deployed node of this type. See instance parameters below.

Parameter	Type	Required	Default	Description
`defaults.pollIntervalMs`	Integer	No	5000	How often operational pages auto-refresh, in milliseconds.
`defaults.timeoutMs`	Integer	No	10000	Proxy timeout for this element type. Slow APIs (e.g. OCS) use a larger value.

Instance parameters:

Parameter	Type	Required	Default	Description
`id`	String	Yes	-	Unique instance identifier. Used in URLs (`/hss/<id>`) and in the proxy path (`/proxy/<id>/...`).
`name`	String	Yes	-	Display name for the instance.
`baseUrl`	String	Yes	-	The element's management API base URL. The proxy forwards requests here. Self-signed TLS is accepted.
`tags`	List	No	`[]`	Free-form labels (e.g. `primary`, `geo-redundant`) shown as chips.

Supported Element Types

`hss`, `mme`, `sgw-c`, `pgw-c`, `upf`, `pcscf`, `icscf`, `scscf`, `tas`, `epdg`, `entitlement`, `smsc`, `smsc-smpp`, `ipsmgw`, `msc`, `stp`, `hlr`, `camel-gw`, `dra`, `ocs`, `twag`, `ran-monitor`, `license`.

Each type maps to a UI module with its own set of tabs, but all are operated through the same generic patterns — see Common Operations.

How OmniWeb Talks to Each NF

The backend proxy is element-agnostic — it forwards whatever the frontend module asks for to the element's `baseUrl`. The *shape* of those calls depends on the element's native API:

API style	Elements	Notes
Element API	All network functions	The element's own management API, proxied verbatim under `/proxy/<instance-id>/...`. OmniWeb forwards `GET`/`POST`/`PUT`/`PATCH`/`DELETE` as the element expects; the request/response shape is whatever that element defines.
Log query (LogQL)	All (Logs tab)	Routed through `/proxy/loki/...` to Loki. See Logs & Tracing.

Because everything is proxied, adding an instance is purely a configuration change in `omniweb.json` — no code change is required to manage another node.

API reference (OpenAPI)

A consolidated OpenAPI/Swagger document (`swagger-doc.json`) ships with OmniWeb and describes the backend and element schemas (for example SMSC

frontend registrations and IMS subscriber locations). It is a reference artifact rather than a live API server.

Proxy Timeouts

Each element type has a configurable proxy timeout (`defaults.timeoutMs`). Most elements respond well within the 10-second default. Elements with heavier queries are given more headroom:

Element	Default Timeout	Why
Most elements	10 seconds	Standard API response time
OCS	180 seconds	Rating/CDR queries over large datasets

Monitoring Integration

OmniWeb does not reimplement Grafana, Loki, or Homer — it integrates them so a single login spans the whole stack:

- **Grafana** is served behind the authenticated gateway and embedded as an iframe. Each request is authorised against your OmniWeb login, so there is no second password. See Grafana.
- **Loki** is reached through the backend's `/proxy/loki/*` gateway, which runs LogQL queries on the operator's behalf. See Logs & Tracing.
- **Homer** (SIP capture) is auto-detected at startup (`/api/homer/status`) and, when present, opens in a new tab — also authorised against your OmniWeb login.

Environment Reference

Backend integration endpoints are configured via environment variables:

Variable	Default	Description
`GRAFANA_URL`	`http://localhost:3000`	Base URL of the Grafana instance OmniWeb proxies to.
`LOKI_URL`	`http://localhost:3100`	Base URL of the Loki instance used for logs and tracing.
`LDAP_URI`	`ldap://localhost:389`	OpenLDAP server for user/SSH-key sync (optional).
`LDAP_BASE_DN`	`dc=omniweb,dc=local`	LDAP base DN.
`LDAP_ADMIN_DN`	`cn=admin,dc=omniweb,dc=local`	Bind DN for LDAP writes.
`LDAP_ADMIN_PASSWORD`	`omniweb-admin`	LDAP bind password (change this).
`AUDIT_API_KEY`	`omniweb-audit-key`	Shared secret for SSH-login audit callbacks (change this).

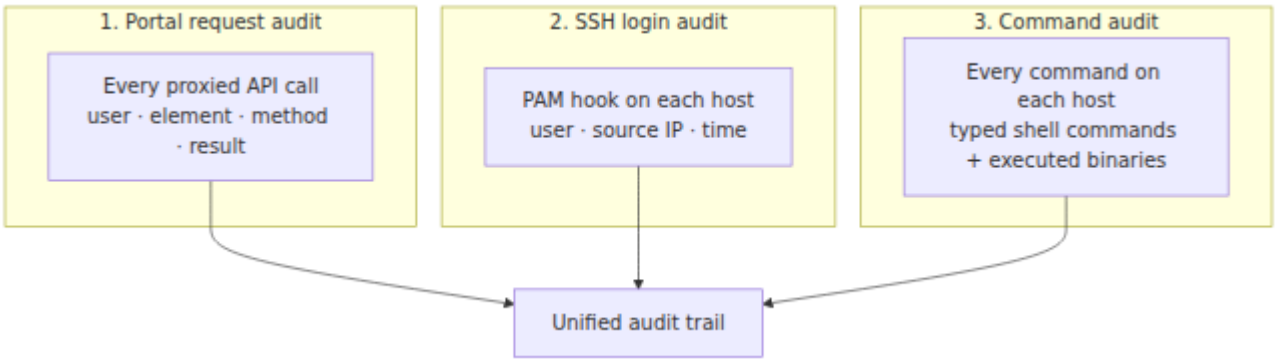
See Authentication & Access Control for the LDAP/SSH and audit details.

Audit Logging

Because OmniWeb is the single gateway to the network, it is also the single place where activity can be recorded. Auditing in OmniWeb has **three layers** that together answer "who did what, where" — across both the portal and the hosts it manages.

← Authentication & Access Control · ← Operations Guide

The Three Layers



Layer	Captures	Where it appears
Portal request audit	Every action taken *through* OmniWeb	Settings → Audit
SSH login audit	Every SSH login *to a managed host*	Settings → Audit (via host callback)
Command audit	Every command *run on a host* (typed + executed)	Settings → Commands

Portal Request Audit

Every request that passes through the OmniWeb proxy is recorded — regardless of whether it succeeded, failed, or was rejected by RBAC. Each entry

captures:

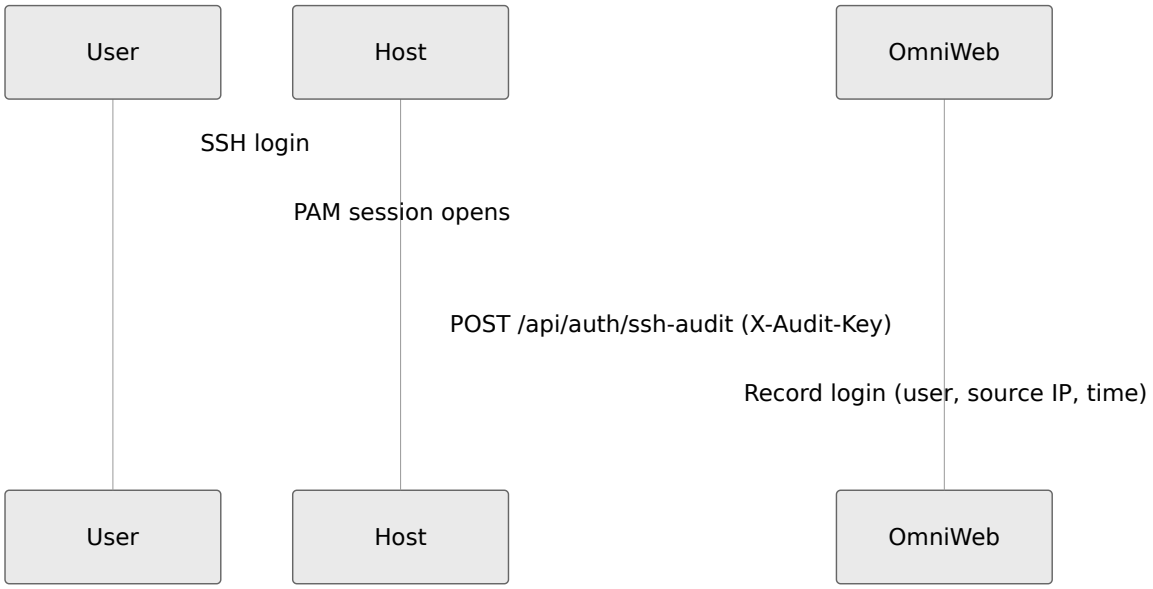
- The **user** who made the request.
- The **element type and instance** targeted.
- The **HTTP method** (which tells you whether it was a view, a change, or a delete).
- The **result** (status code).
- The **timestamp**.

This gives a complete record of who changed what, where, and when, inside the portal. It lives under **Settings → Audit**. Because reads (`GET`), writes (`POST`/`PUT`/`PATCH`), and deletes are all distinguishable by method, the audit log doubles as a change history for the network.

The Audit Trail (Settings → Audit) — every action with the user, event type, element/instance, HTTP method, result, and source IP. Note the denied `DELETE` (403), the `permission_changed`, and the `ssh_login` events alongside ordinary API requests. Filter by event type, user, element, or date range.

SSH Login Audit

Hosts can report SSH logins back to OmniWeb so that host access shows up in the same audit view as portal activity. A small PAM hook on each host calls OmniWeb when a session opens, reporting the username, source IP, and service detail.



The callback is authenticated with a shared secret so only trusted hosts can write audit entries:

Variable	Default	Description
`AUDIT_API_KEY`	`omniweb-audit-key`	Shared secret the host PAM hook presents (as `X-Audit-Key`) when reporting a login. **Change this** — and set the matching value on each host.

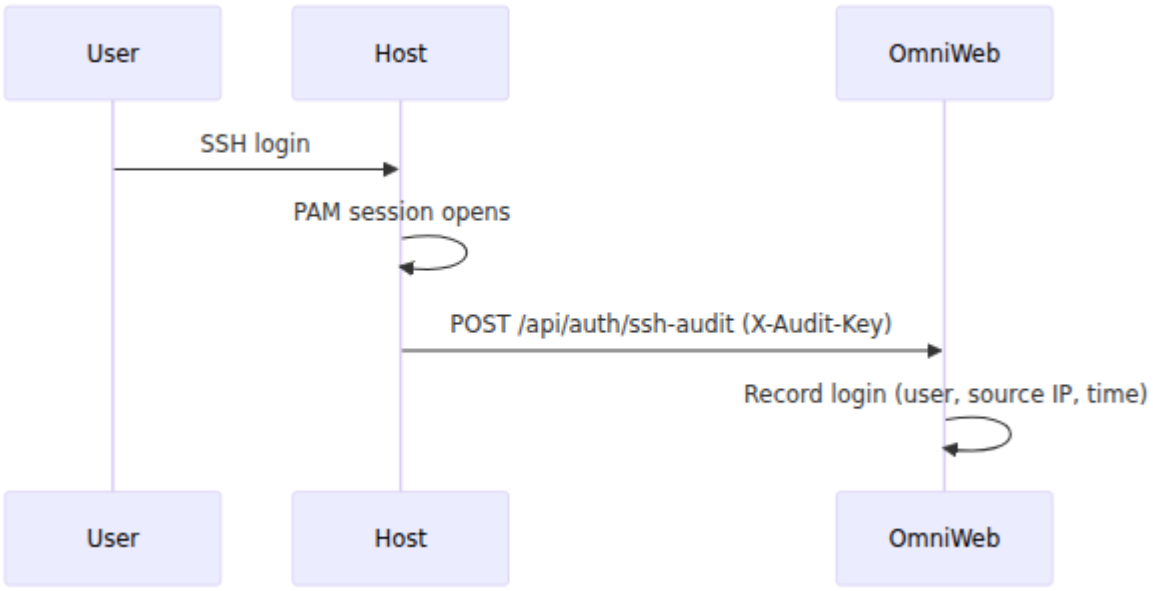
With this in place, every SSH login to a managed host appears under **Settings → Audit** alongside portal actions, tying together a person's portal and host activity.

Command Audit

For command-level visibility on the hosts, OmniWeb collects **two complementary streams**, both shipped to Loki and browsable under **Settings → Commands** with a source toggle:

Source	Captures	Loki stream
Typed commands	The exact command lines a user types in an interactive shell — **including shell built-ins** like `echo`, `cd`, `export`	`job="cmdaudit"`
Binary executions	Every binary that actually ran on the host (scripts, tools, automation)	`job="auditd"`

Neither replaces the other: built-ins never run a separate binary, so only the typed-commands stream sees them; and automation/scripts that never sit at an interactive prompt only appear in the binary-execution stream. Together they cover *what a person typed* and *what actually executed*.



Typed commands are captured by a lightweight interactive-shell hook that records each command a user runs (with user, working directory, and TTY) — the layer that catches built-ins a binary tracer structurally cannot see.

Binary executions come from a lightweight host agent that records every binary run on the host. Because it works in userspace it functions inside LXC containers without kernel audit support. Each entry captures:

Field	Meaning
`login`	The user who originally logged in — **survives** `sudo`/`su`, so privilege changes don't hide who is acting
`uid`	Effective UID the command actually ran as
`filename`	The binary that was executed
`cwd`	Working directory at the time of execution
(after `]:`)	The full command line

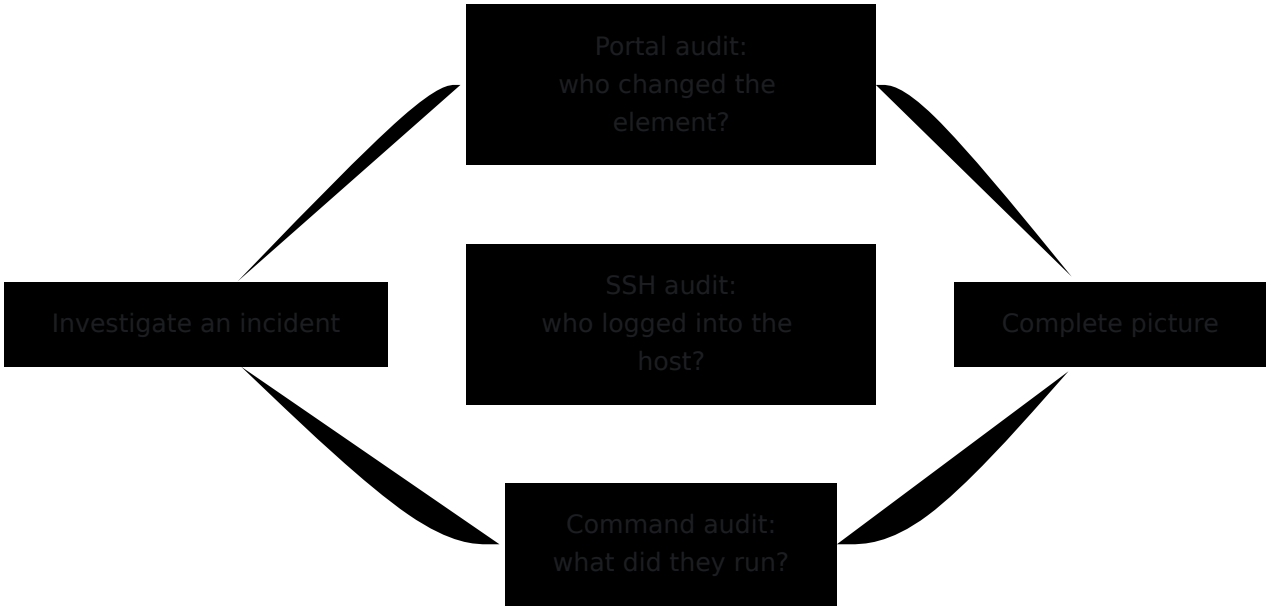
A representative execution entry:

```
2026-04-16T13:18:35 omni-stp01 [login:nickj uid:0 tty:(none)
cwd:/home/nickj filename:/usr/bin/sudo]: sudo tail -10
/var/log/auth
```

The **Settings → Commands** view defaults to the typed-command stream and lets you filter by host, search by user or command, and live-tail activity across the fleet — with the source toggle to switch to the raw binary-execution view when you want to see exactly what ran.

> **Note:** the command streams are an operator-visibility aid captured in userspace. A determined root user can bypass them; kernel-level auditing is the hardening step if you need non-bypassable capture. For the security-relevant, structured record (logins, permission changes, denials), rely on the **Portal** and **SSH login** audit above, which are written to OmniWeb's own database.

How the Layers Fit Together



- A configuration change made **through OmniWeb** → portal request audit.
- A change made **by hand on a host** → SSH login audit (who got in) + command audit (what they did).

Together they leave no blind spot between "changed it in the UI" and "changed it on the box".

Related

- Logs & Subscriber Tracing — the general log explorer behind the command streams.
- SSH Key Login — the access side of host activity.
- RBAC — the permissions whose use the portal audit records.

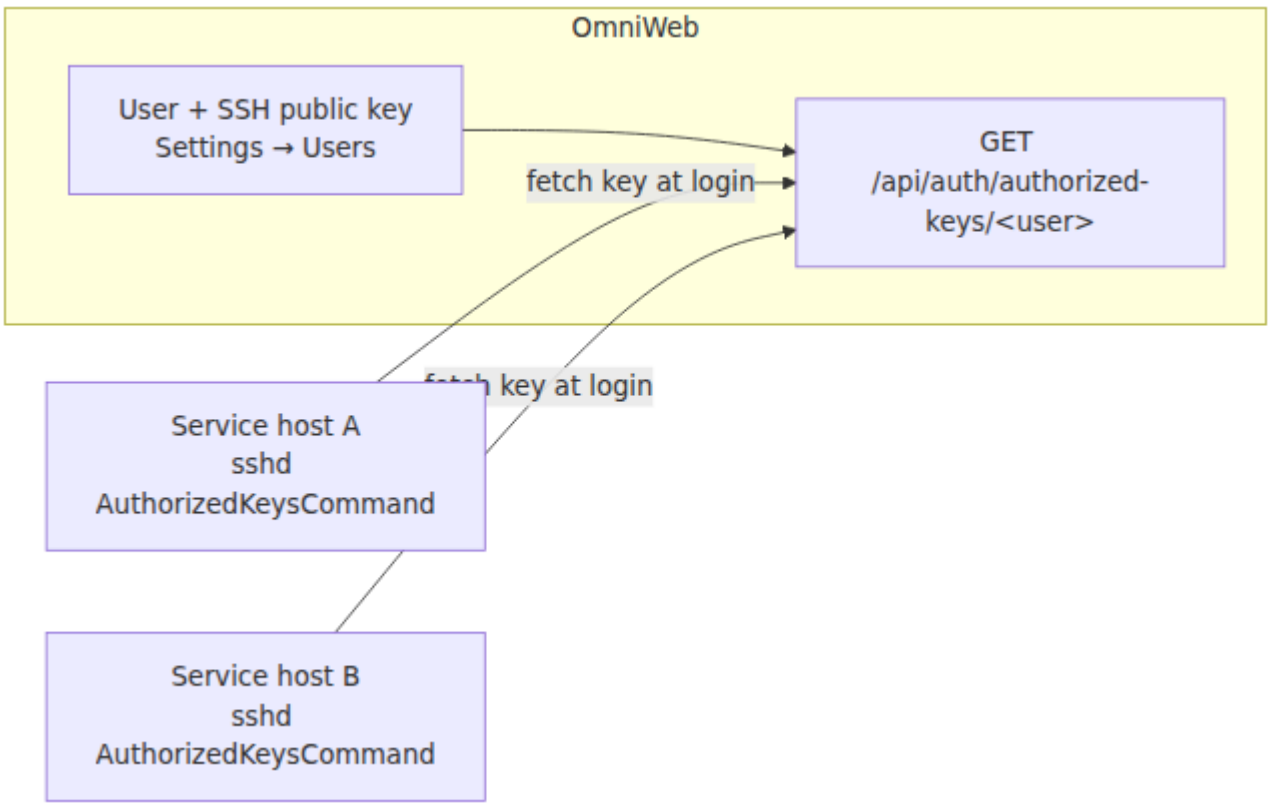
SSH Key Login (Host Access)

OmniWeb's access control extends beyond the portal to the **hosts** that run the network. Each user can carry an **SSH public key**, and the managed hosts fetch those keys **live from OmniWeb** at login time — so OmniWeb is the single place to manage *who can SSH where*, and, combined with audit logging, who actually did.

There is no directory server to run and no per-host `authorized_keys` files to maintain. (This replaces an earlier OpenLDAP-sync design; the directory is gone — keys are served on demand over HTTPS.)

← Authentication & Access Control · ← Operations Guide

What It Does



You give a user an SSH public key in OmniWeb once. Every managed host's `sshd` is configured to ask OmniWeb for that user's key **at the moment of login** (via OpenSSH's `AuthorizedKeysCommand`). Add a key in OmniWeb and the user can SSH to every enrolled host; disable the user and that access is gone everywhere on their **next** login attempt — no sync, no drift, no key files to clean up.

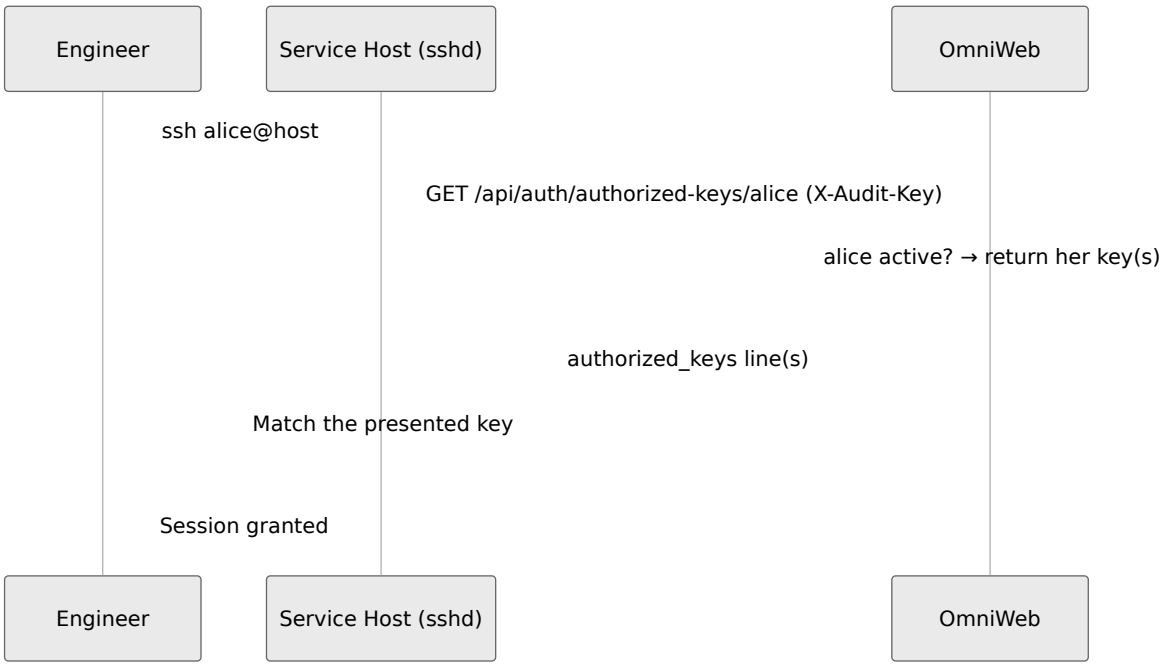
Setting a User's Key

Keys are managed on **Settings → Users** — the same screen that manages portal accounts, RBAC, and active/disabled state. Each user row shows whether an SSH key is configured; create or edit a user to paste one in.

Settings → Users — each user with whether an SSH public key is configured. The key you paste here is the key that grants host access; there is nowhere else to maintain it.

How Hosts Consume It

Each managed host runs a tiny helper wired into `sshd` as its `AuthorizedKeysCommand`. When someone connects, `sshd` runs the helper, which fetches that user's authorised keys from OmniWeb over HTTPS and hands them back for the match:



Key properties of this flow:

- **Live** — the key is fetched per login, so changes take effect immediately.
- **Active-users-only** — OmniWeb only returns keys for **active** users. Disabling (or deleting) a user in OmniWeb denies their next login on every host at once. This is the revoke button.
- **No enumeration** — an unknown/disabled user, or one with no key, returns an empty result (the login simply finds no key and is denied), so the endpoint doesn't leak which usernames exist.
- **Survives an OmniWeb outage** — the helper caches the last good result per user on the host, so a brief OmniWeb outage doesn't lock everyone out.

The account itself (the Unix user, home directory, shell) is provisioned by configuration management alongside the `sshd` helper — OmniWeb owns the **identity and keys**, the fleet automation owns **account existence** on each host.

Configuration

The key-serving endpoint is guarded by the same shared secret as the audit callback:

Variable	Default	Description
`AUDIT_API_KEY`	`omniweb-audit-key`	Shared secret each host's `AuthorizedKeysCommand` helper presents (as `X-Audit-Key`) when fetching keys. **Change this** — and set the matching value on each host.

Because OmniWeb is the only thing that should be queried and the helper runs on trusted hosts, the endpoint is served over HTTPS and the secret is a second gate on top of the network path.

Why This Matters

- **One place to grant/revoke SSH access** across the whole fleet — the user's active state *is* their host access.
- **No scattered key files** drifting out of sync on individual hosts, and no directory server to operate and back up.
- **A single identity** tying a person's portal access, their SSH access, and — through audit logging — their recorded activity.

Related

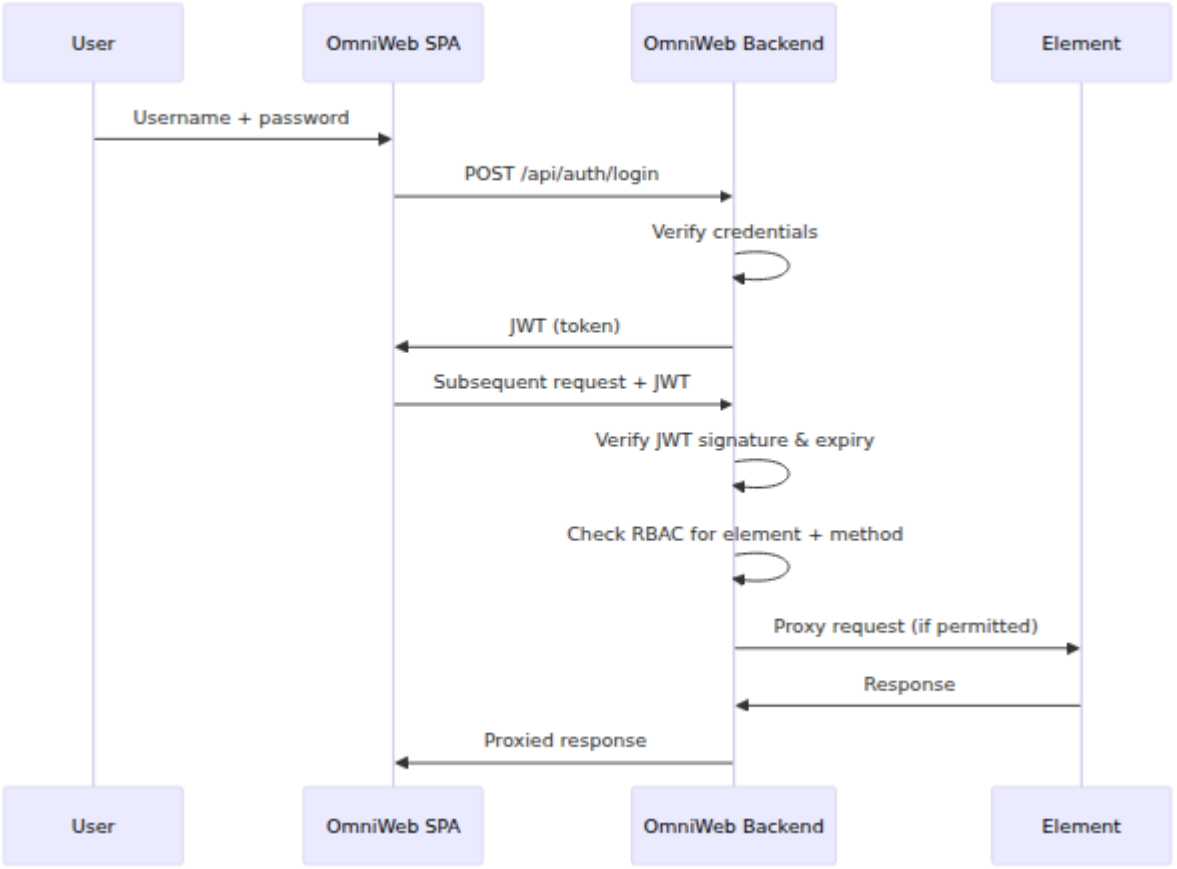
- Audit Logging — record SSH logins and every command run on the hosts.
- RBAC — control what the same users can do *inside* OmniWeb.

Login & JWT Sessions

OmniWeb authenticates operators with a username and password and issues a JSON Web Token (JWT). That token proves who you are on every subsequent request — and, crucially, it is the *same* credential that authorises the embedded Grafana, Loki, and Homer. One login spans the whole stack.

← Authentication & Access Control · ← Operations Guide

Login Flow



The token is issued by `POST /api/auth/login`. Once obtained, the SPA presents it on every API call, and the backend verifies the signature and expiry before doing anything else.

The login screen. Authenticating here yields the JWT that governs the entire stack — element APIs, Loki, Grafana, and Homer.

How the Token Is Carried

The backend accepts the JWT in three locations, which lets the same token work for XHR API calls, browser navigations, and embedded iframes alike:

Location	Used for
Authorization header	Standard API (XHR/fetch) requests from the SPA
Cookie (`access_token`, HttpOnly)	Browser navigations and embedded content (Grafana iframe, Homer tab)
Query string	Cases where neither a header nor a cookie can be set

The cookie is **HttpOnly** (not readable by JavaScript) and uses `SameSite=Lax`, which protects the token from theft via cross-site script access while still allowing top-level navigations to the embedded tools.

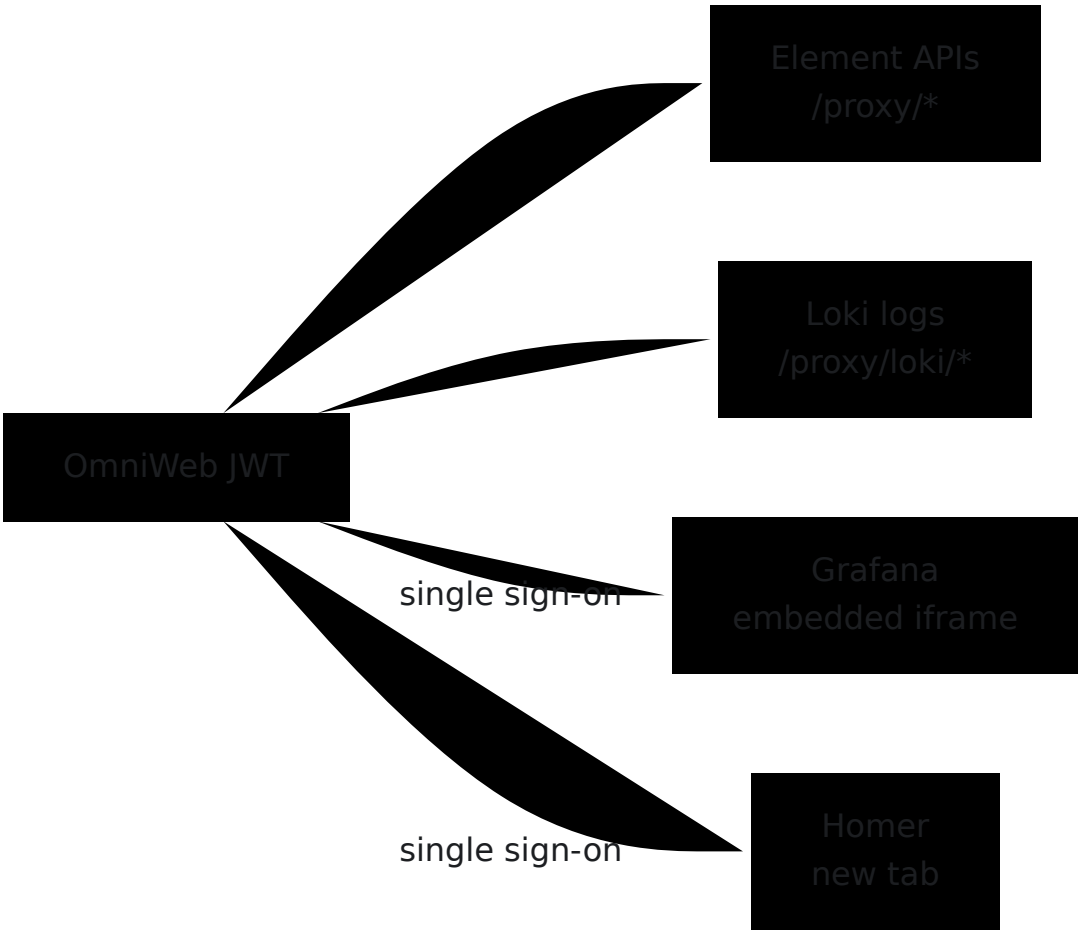
Token Lifetime

The access token has a configurable expiry — **24 hours by default** (`JWT_ACCESS_TOKEN_EXPIRES`, in seconds). When it expires, the operator is returned to the login screen and re-authenticates. There is no silent indefinite session; the lifetime is a deliberate security boundary.

Setting	Default	Description
`JWT_ACCESS_TOKEN_EXPIRES`	`86400` (24 h)	Access-token lifetime in seconds. Lower it for tighter sessions; raise it for fewer re-logins.
`JWT_SECRET_KEY`	*(generated)*	Secret used to sign and verify tokens. Set explicitly in production so tokens survive a backend restart and are consistent across instances.

One Login, Whole Stack

The same JWT authorises the integrated monitoring tools, so operators never log in twice:



- **Element APIs and Loki** are reached through the backend proxy, which checks the JWT directly.
- **Grafana and Homer** are reached through the authenticated gateway, which verifies your OmniWeb login before proxying — no second password. See Grafana and Homer.

This is why a single OmniWeb session governs the entire operational surface — dashboards, logs, traces, and element management all ride on the one token.

First Login & Changing Credentials

A fresh deployment ships with a default administrator account (`admin` / `admin`). **Change this immediately** after first login, via Settings. The default exists only to bootstrap the system and must not survive into production.

Logout & Expiry

Logging out discards the token. An expired or invalid token causes the next request to be rejected and the SPA to return to the login screen. Because element actions are only ever performed through an authenticated, RBAC-checked proxy call, an expired session can never perform a privileged action.

Related

- Role-Based Access Control — what a logged-in user is allowed to do.
- Audit Logging — every authenticated action is recorded.

Role-Based Access Control (RBAC)

Being authenticated tells OmniWeb *who* you are. RBAC decides *what you may do*. Access is granted at a fine grain — **per element type, per HTTP method** — which maps directly to the real operations an operator performs: view, modify, or delete.

← Authentication & Access Control · ← Operations Guide

Permission Levels

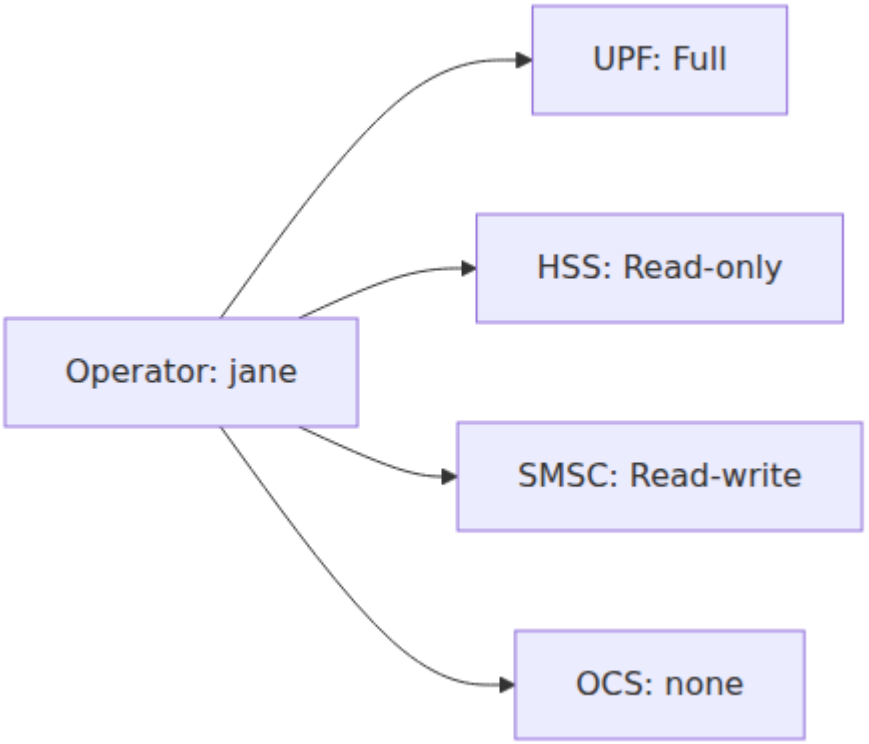
Each element type can be granted to a user at one of three levels. The level is expressed as the set of HTTP methods the user may use against that element through the proxy:

Permission	Methods allowed	What the operator can do
Read-only	`GET`	View data only — no changes
Read-write	`GET`, `POST`, `PUT`, `PATCH`	View and modify — e.g. add a subscriber, edit a route, change a QoS rule
Full	`GET`, `POST`, `PUT`, `PATCH`, `DELETE`	Everything, including deletion and clearing sessions

Because permissions are keyed to HTTP methods, they line up exactly with the generic editing model: an "Add" form is a `POST`, an "Edit" is a `PUT`/`PATCH`, and a "Delete"/"clear" is a `DELETE`.

Per-Element Granularity

Permissions are assigned **independently per element type**. A user can hold different levels on different elements:



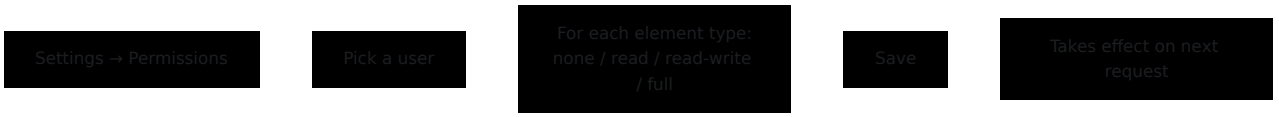
In this example Jane can clear UPF sessions, can view but not change the HSS, can add SMSC routes, and cannot see the OCS at all. This lets you give, say, a dataplane engineer full control of the UPF while keeping subscriber data read-only.

Administrators

Administrators have full access to all element types automatically. The admin role is also what grants access to user and permission management. Keep the number of administrators small, and prefer specific per-element grants for everyone else.

Managing Roles & Permissions

Permissions are managed under **Settings → Permissions**. For each user you choose, per element type, whether they have read-only, read-write, or full access (or none).

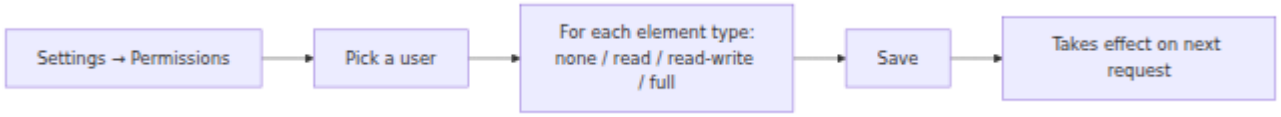


How Enforcement Works

RBAC is enforced in two complementary places:

1. **In the UI** — controls for actions you cannot perform are not shown. If you only have read access to the HSS, you will not see Add/Edit/Delete buttons there.
2. **In the backend** — every proxied request is re-checked against your permissions *server-side*. Even a request crafted to bypass the UI is rejected if your role does not permit that element/method.

This "defence in depth" means the UI is a convenience, not the security boundary — the backend proxy is.



Settings → Permissions — select a user, then grant a level (read-only, read-write, or full) per element type.

Good Practice

- **Least privilege** — grant the lowest level that lets someone do their job; widen only when needed.
- **Read-only by default** for elements an operator only needs to observe.
- **Reserve Full (DELETE)** for those who genuinely need to clear sessions or remove records.
- **Few admins** — use targeted per-element grants instead of handing out admin.

Related

- Login & JWT Sessions — authentication that RBAC builds on.
- Common Operations — how permissions shape the everyday UI.
- Audit Logging — both permitted and denied actions are recorded.

Authentication & Access Control

OmniWeb is the single front door to the entire network, so access control is one of its most important responsibilities. It uses JWT-based login, fine-grained role-based access control (RBAC) over every element, SSH-key login for host access, and a unified audit trail spanning the portal and the hosts it manages.

This area is broken into focused pages:

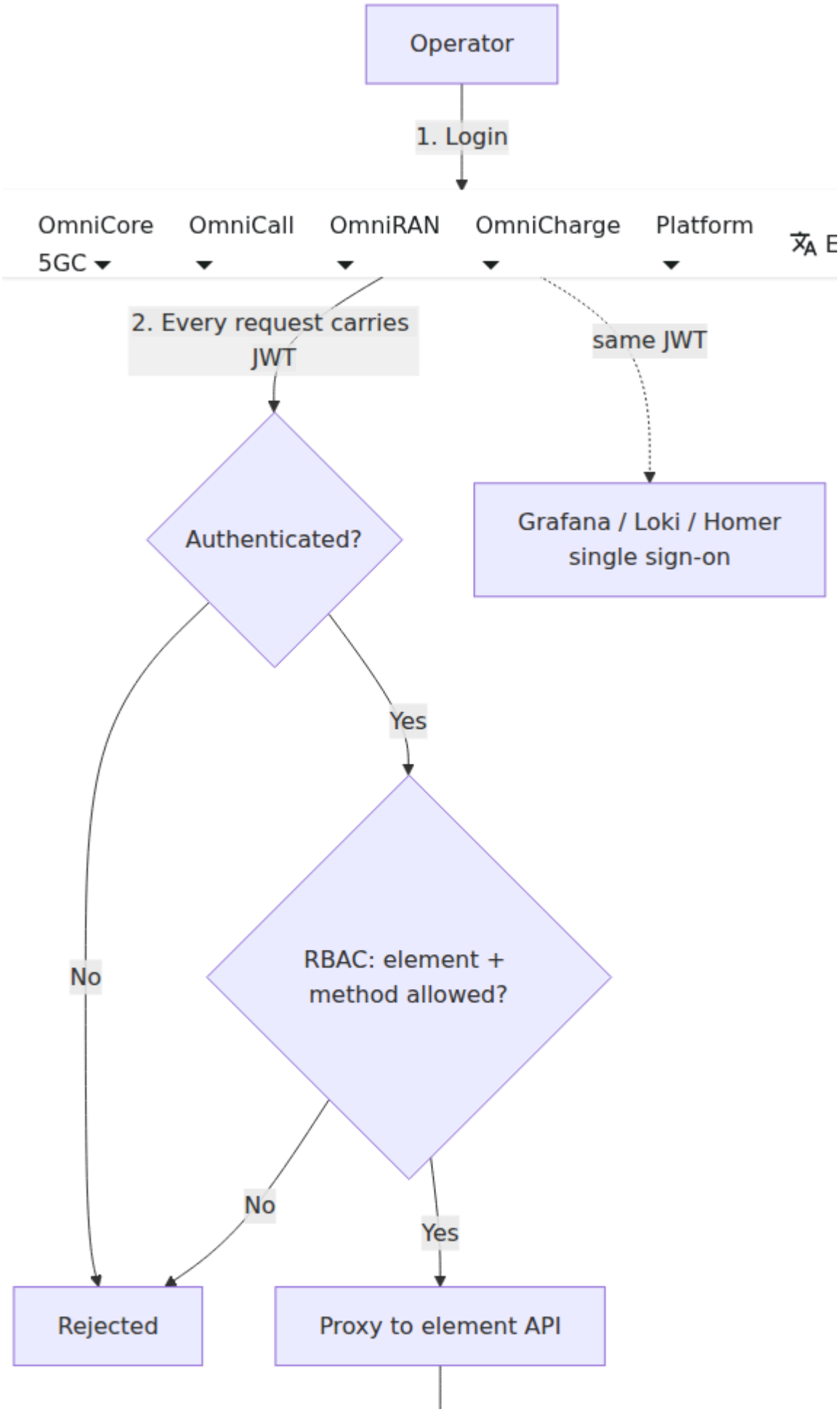
← Back to Operations Guide

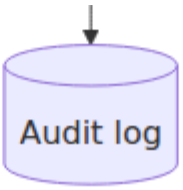
In This Section

- **Login & JWT Sessions** — How operators log in, how the JWT works, token lifetime, and how a single login spans Grafana, Loki, and Homer.
- **Role-Based Access Control (RBAC)** — Per-element, per-method permissions; how to grant them; and how they are enforced.
- **SSH Key Login** — Managed hosts fetch each user's SSH public key live from OmniWeb (AuthorizedKeysCommand), so host access is granted/revoked centrally.
- **Audit Logging** — The three layers of audit: portal request audit, SSH-login audit, and command audit.

The Access Control Model

OmniWeb layers four controls. A request must pass each before it reaches an element, and the whole chain is recorded.


Operator
1. Login
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
2. Every request carries JWT
same JWT
Authenticated?
Grafana / Loki / Homer single sign-on
Yes
No
RBAC: element + method allowed?
No
Yes
Rejected
Proxy to element API



Layer	What it controls	Page
Authentication	Who you are — proven by a signed JWT	Login & JWT
Authorisation (RBAC)	What you may do — per element type and HTTP method	RBAC
Host access	Who may SSH where — keys served live from OmniWeb	SSH Key Login
Audit	What actually happened — across portal and hosts	Audit Logging

Why It Matters

Because OmniWeb proxies every element, Grafana dashboard, Loki query, and Homer session, this single access-control chain governs the *entire* operational surface of the network. There is one place to grant access, one place to revoke it, and one trail that records who did what — on the portal and on the underlying hosts.

One trail for everything: API requests, logins, SSH logins, and permission changes — with user, element, method, result, and source IP. See Audit Logging.

Start with Login & JWT Sessions.

Common Operations

Every network element in OmniWeb is different on the inside, but the way you *work* with them is deliberately the same. This page documents the patterns shared by all elements — viewing data, editing and changing NF setups, reading configuration, running actions, watching logs, and understanding polling and permissions. Learn these once and they apply to the HSS, the UPF, the SMSC, the MSC, and every other NF.

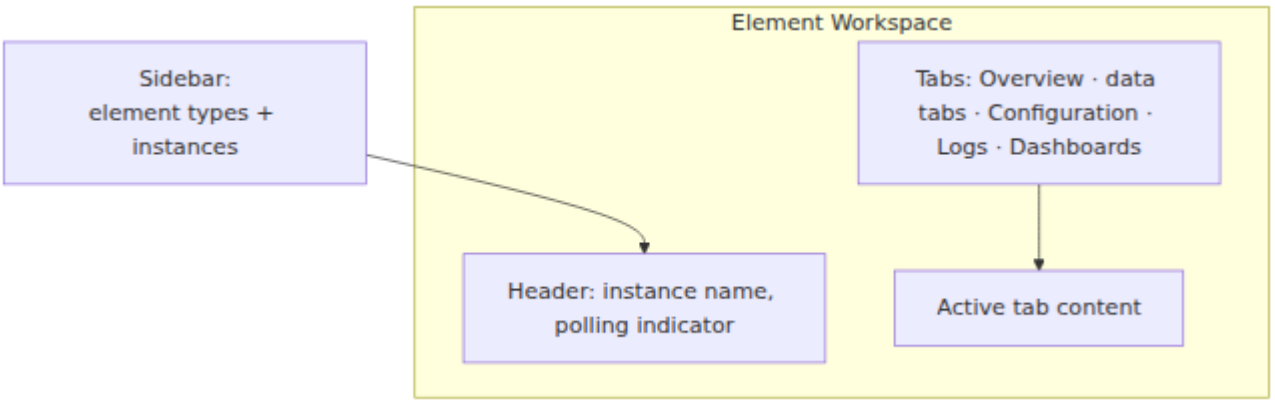
The screenshots below come from specific elements, but they illustrate the *generic* pattern — the same controls appear, with different fields, on every element.

← Back to Operations Guide

Table of Contents

Page Anatomy

Selecting an element instance in the sidebar opens its workspace. Every element workspace shares the same layout:



- **Overview** — the landing tab: health, key counts, and component status for that instance.
- **Data tabs** — the element-specific views (sessions, subscribers, peers, routing, etc.).
- **Configuration** — a structured view of the element's runtime configuration (where exposed).
- **Logs** — a live Loki log viewer scoped to that element.
- **Dashboards** — embedded Grafana dashboards for that element (when Grafana tags match).

Sidebar entries are ordinary links, so you can **Ctrl/Cmd-click** (or middle-click, or right-click → *Open in new tab*) any of them to open that workspace in a new browser tab — handy for watching two elements at once.

Viewing Data

Operational data is presented in a consistent, sortable table throughout OmniWeb. Tables provide:

- **Sorting** by column.
- **Pagination** for large result sets.
- **Search / filtering** where the element API supports it (e.g. search by IMSI/MSISDN or UE IP).
- **Skeleton loading** while data is fetched, and a clear **empty state** when there is nothing to show.
- **Row click-through** to a detail view where one exists.

Values are formatted for readability — timestamps as local time, durations as `h m s`, byte counts as KB/MB/GB, and large numbers with thousands separators.

The generic list pattern (here, a subscriber list): searchable, sortable, paginated, with click-through and inline editing. The same table drives sessions, peers, routes, and every other listing.

Tables can render richer cells — here, utilisation bars per row. The pattern is the same; only the columns differ.

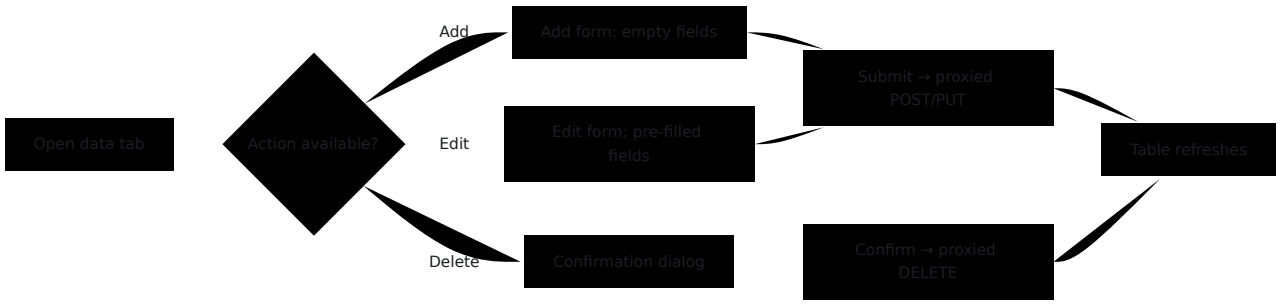
Detail Views

Where a row has more to show, clicking it opens a detail view. These reuse the same chrome (header, polling, actions) but present the single record in depth — for example an interactive relationship map for a complex object:

A detail view opened from a table row — here, an interactive map of one session's internal rules. Detail views are reached the same way everywhere: click the row.

Editing & Changing NF Setups

Wherever an element's API supports modification, OmniWeb exposes it through a consistent **action form**. The same form component drives "add a subscriber", "add a route", "edit a QoS rule", and "change a profile" — only the fields differ.



Action forms support these field types: **text**, **number**, **select** (dropdown), **boolean** (toggle), **password** (masked, with a show/hide toggle), and **multi-line text**. Each field can be marked required, carry a placeholder, and show

helper text. The form shows a loading state while submitting and reports success or failure via an on-screen notification.

The HTTP method behind each action maps to a permission level — see Permissions.

The same pattern, three elements

These illustrate the *same* mechanic applied to different NFs:

Task	Where	What happens
Add/edit a subscriber	HSS → Subscribers → Add	Fill the form (IMSI, MSISDN, profile, keys) → proxied `POST`/`PUT` → list refreshes
Add/change a route	SMSC → Routing → Add (or MSC → Routing)	Fill the rule (pattern, destination, priority) → proxied write → table refreshes
Clear/release a session	UPF → Sessions (or MSC → Calls)	Select the record → confirm → proxied `DELETE` → list refreshes

The mechanics — form or confirmation dialog, proxied write, table refresh — are identical. Only the element and fields change.

Changing routing is the same edit pattern: the routing table is a list you add to and edit; some elements additionally parse destinations into readable chips and offer a simulator to test a change before it goes live.

Running Actions

Beyond CRUD, many elements expose **operational actions** that don't fit a simple form. These are surfaced as buttons that trigger a proxied call and show the result inline. Examples of the same button-action pattern:

Example action	Element	Effect
Force release a call	MSC → Calls	Tears down an active call
OPTIONS ping a SIP peer	MSC → SIP Peers	Tests reachability of a SIP trunk
MAR test a Diameter peer	TWAG → Diameter	Sends a test Multimedia-Auth-Request
Activate/deactivate a redirect	UPF → Walled Garden	Redirects or restores a subscriber
Ping from the node	UPF → Diagnostics	Runs a ping from the element itself
Disconnect a session	TWAG → Sessions	Drops an authenticated WiFi session
Manual paging	MSC → Paging	Triggers a paging request

Destructive actions (release, delete, deactivate) always require explicit confirmation before they run.

Operational actions appear as buttons alongside the data they affect — here, activating/deactivating subscriber redirects and managing whitelisted IPs. Destructive actions prompt for confirmation first.

Viewing Configuration

Elements that expose their runtime configuration have a **Configuration** tab. OmniWeb renders it in a structured, readable layout with a **Raw JSON** toggle for the full underlying document.

Two behaviours are worth knowing:

- **Mixed config formats are normalised.** Some elements return configuration as Python-style dictionaries (`'key': True`, `None`) rather than strict JSON. OmniWeb parses these into a clean view automatically and falls back gracefully if a value can't be parsed.
- **Secrets are masked.** Any field whose name looks sensitive (matching `password`, `secret`, `token`, or `credential`) is displayed as `***` rather than its real value, both in configuration views and in form fields.

The generic configuration view: settings in a structured layout, with a Raw JSON toggle and sensitive values masked. Elements that expose config (UPF, TAS, ePDG, SMSC frontends, OCS, RAN Monitor) all use this pattern.

Logs on Every Element

Every element has a **Logs** tab backed by Loki. It queries that element's service logs by host/label, supports structured filtering, and streams new lines as they arrive. Because the log model is identical across elements, it is documented in full on its own page:

➡ **Logs & Subscriber Tracing**

Every element's Logs tab uses the same viewer — structured, colour-coded, live-streamed Loki logs with service and severity filtering.

Dashboards on Every Element

Elements tagged for Grafana show a **Dashboards** tab. OmniWeb discovers matching dashboards by tag and renders them inline, so you can see the statistics for an element without leaving its workspace. See:

➡ **Grafana Dashboards & Statistics**

Real-Time Polling

Operational pages auto-refresh. The **Polling Indicator** in each page header shows:

- **Active (green dot)** or **paused**.
- The **interval** (default 5 seconds, set per element via `pollIntervalMs`).
- A **spinner** while a fetch is in flight.

- The **timestamp** of the last successful update.

Polling can be toggled per page. Pages built around interactive or one-shot actions — consoles, diagnostics, JSON executors — do not poll by default and are refreshed manually. Pausing polling is useful when you want a stable view while reading a large table or while a trace is in progress.

Permissions & What You Can Change

What you can *do* on an element depends on your role. Access is granted **per element type, per HTTP method**:

Permission	Methods allowed	Typical use
Read-only	`GET`	View data, no changes
Read-write	`GET`, `POST`, `PUT`, `PATCH`	View and modify (add subscriber, edit route)
Full	`GET`, `POST`, `PUT`, `PATCH`, `DELETE`	Everything, including deletion / clearing

Permissions are assigned independently per element — you might have full access to the UPF but read-only on the HSS. If an action is not permitted for your role, its control is unavailable. Administrators have full access to all elements. See Authentication & Access Control for managing roles, and note that **every proxied request is audit-logged** regardless of role.

Multi-Instance & Fleet Views

Many element types run as multiple instances — several UPFs behind a PGW-C, a pair of SMSCs, redundant STPs. OmniWeb handles this at the element type level: a type with more than one configured instance gets two navigation modes.

Sidebar & Instance Selection

When a type has multiple instances, the sidebar entry expands to list each one by name. Clicking a named instance opens that instance's workspace directly. An instance dropdown also appears in the workspace header, letting you switch between instances without going back to the sidebar — useful when comparing behaviour across nodes without losing your position on a particular tab.

The instance selector in the top-right of the workspace header. Opening it shows every configured instance plus "All Instances" to jump back to the fleet view — without leaving your current tab.

Fleet View

Clicking the element type name in the sidebar (rather than a specific instance) — or selecting **All Instances** from the dropdown — opens the **fleet view** for that type. The fleet view shows a summary card per instance with health status and key metrics, so you can assess the whole fleet at a glance before drilling into a specific node.

UPF fleet view: one card per instance, all polled in parallel. Click any instance card or use the dropdown to drill in.

When to Use Each View

Situation	Use
Checking overall fleet health	Fleet view — all instances at a glance
Investigating a specific node	Instance dropdown or sidebar sub-item
Comparing two UPFs session counts	Open each in the instance dropdown in turn
Clearing a session on a specific UPF	Navigate to that instance, then Sessions

DRA — Diameter Routing Agent

The **DRA** is the Diameter routing hub of the core. Rather than every node holding a full mesh of Diameter connections, the EPC and IMS functions connect to the DRA and it relays and routes their signalling — **S6a** between the MME and HSS, **Gx/Gy** to the PCF/PCRF and OCS, **Cx/Dx** and **Sh** for the IMS, and so on. It chooses where each message goes from **routing rules** (matching on Diameter application, realm or AVP) and a peer-selection algorithm, and it can rewrite messages with **transform rules**. It is the Diameter counterpart of an SS7 STP. OmniWeb surfaces the DRA's peers, routing rules, live status and a connection map; throughput and trend charts are on the Grafana dashboards reached from the same page.

← Operations Guide

The DRA is a per-instance element page reached from the sidebar when one runs in the site's inventory, with a fleet view (the network map) where several are deployed. All traffic goes through the OmniWeb backend proxy to the DRA's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Network Map

A hub-and-spoke view of the DRA and every Diameter peer it knows, with the connection count at a glance (e.g. *5/6 connected*). Connected peers are linked to the DRA in green; an administratively **disabled** or **disconnected** peer is shown greyed and dashed. It is the fastest way to see, in one picture, which signalling neighbours are up.

Status

The **Status** tab reports the DRA's own identity and configuration: its **Origin-Host**, **realm**, the **listen IP and port** (Diameter is usually 3868), the reported **product name**, and the **peer-selection algorithm** in use (e.g. round-robin). This is where you confirm the DRA is running and configured as expected.

Peers

Every Diameter peer, each row showing the peer's **Origin-Host**, **IP and port**, reported **product**, **transport** (SCTP or TCP), **realm**, the negotiated **Diameter applications**, and its **connection state**. Peers can be filtered by name, application, realm or state. Each peer has an **enable/disable** toggle — disabling a peer administratively takes it out of routing without tearing down its configuration, which is how you drain a neighbour for maintenance.

Rules

The active **routing rules** and **transform rules**, and the **routing algorithm** they run under. Each routing rule matches on a filter (a Diameter application, an AVP, or all traffic) and resolves to a destination — either an explicit set of peers or a named strategy such as *destination-host* or *destination-realm*. This is where you confirm, for example, that S6a is routed to the HSS and Gx to the PCF.

Logs

The **Logs** tab tails the DRA's live log output for troubleshooting a single routing agent without leaving OmniWeb.

Monitoring dashboards

Message rates, per-interface throughput and other time-series for the DRA are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- **MME** and the **HSS** — the DRA relays **S6a** between them.
- The **PCF/PCRF** (**Gx**) and **OCS** (**Gy**) — policy and charging peers routed through the DRA.
- The **CSCF** and **HSS** — **Cx/Dx** IMS signalling.
- **STP** — the SS7 routing counterpart to the DRA in the signalling plane.

Entitlement Server — OmniSEP

The **Entitlement Server** (**OmniSEP**) decides which services a subscriber and their device are allowed to use. When a handset asks whether it may turn on **VoWiFi**, **VoLTE** or **Wi-Fi Calling** — or runs **ODSA** device provisioning to add or activate a device — it checks in with the entitlement server, which authorises the request over the **entitlement (TS.43)** interface, resolves subscriber data over **Diameter**, and applies the subscriber's **XCAP** service profiles. OmniWeb surfaces its live entitlements, activity, sessions, tokens, Diameter peers and XCAP profiles on one element page; volume and trend charts live on the Grafana dashboards reached from the same page.

← Operations Guide

OmniSEP is a per-instance element page reached from the sidebar when one runs in the site's inventory. All traffic goes through the OmniWeb backend proxy to the entitlement server's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

The landing tab is a single-screen health read for the entitlement server: its **system status**, the count of active **sessions** and **tokens**, subscribers carrying custom **entitlements**, and the health of its **Diameter** peers. It is the fastest way to confirm the server is up and authorising devices.

Entitlements

The subscribers that carry **custom entitlements** — overrides to the default service set, per subscriber. Open a subscriber to see which services are enabled for them. An operator can **delete all entitlements** for a subscriber to return them to the network default — useful for clearing a bad override during a fault.

Activity

The recent **activity** records — a feed of the entitlement requests the server has handled, each showing the subscriber or device, the service requested and the outcome. Open a record for its detail. This is the audit trail for confirming whether a handset actually asked for VoWiFi/VoLTE and what answer it was given.

Sessions

The active **EAP-AKA sessions** the server is holding for device authentication, each with its subscriber identity and state. An individual session can be **expired** from here to force a device to re-authenticate — the quickest way to clear a session that has stuck.

Tokens

The active **authentication tokens** the server has issued to devices after they authenticated. Each token can be **revoked** individually, which withdraws a device's authorisation and requires it to authenticate afresh — the entitlement equivalent of forcing a re-login.

Diameter

The **Diameter peers** the entitlement server connects to for subscriber data, with each peer's **connection state**. When entitlement decisions start failing for reasons that look like missing subscriber data, this tab is where you confirm the Diameter path is up.

XCAP Profiles

The **XCAP profiles** — the per-subscriber service settings (such as supplementary-service configuration) the entitlement server applies. Open a profile to read it. A profile can be **deleted** to clear a subscriber's stored settings and fall back to the default.

Logs

The **Logs** tab tails the entitlement server's live log output for troubleshooting a single server without leaving OmniWeb.

Monitoring dashboards

Request rates, per-service counters and other time-series for the entitlement server are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- The **HSS** holds the subscriber data the entitlement server's Diameter requests resolve against.
- The **CSCF** and **TAS** deliver the VoLTE/VoWiFi services the entitlement server authorises.
- The **ePDG** and **TWAG** carry the Wi-Fi Calling access the entitlement server enables.
- The **DRA** relays the entitlement server's Diameter signalling to its peers.

ePDG — Evolved Packet Data Gateway

The **ePDG** is the secure gateway for **untrusted Wi-Fi access** — the node behind Wi-Fi Calling (VoWiFi) and untrusted-WLAN data. A handset builds an IPsec/IKEv2 tunnel to the ePDG over any internet connection; the ePDG authenticates the subscriber against the AAA/HSS over Diameter (SWm/SWx), anchors the session to the packet core (S2b to the PGW), and enforces per-subscriber rate limits and geo-IP policy. OmniWeb surfaces the ePDG's live tunnels, its Diameter peers, rate-limiting and geo-IP configuration, and runtime settings; longer-run trends are on the Grafana dashboards reached from the same page.

← Operations Guide

The ePDG is a per-instance element page reached from the sidebar when one runs in the site's inventory. All traffic goes through the OmniWeb backend proxy, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

A single-screen health read: the ePDG's **status**, the number of **active sessions** (IPsec tunnels), and a summary of the gateway's operational state — the quickest confirmation that the ePDG is up and carrying Wi-Fi-calling tunnels.

Sessions

Every active session — the IPsec/IKEv2 tunnels currently established — keyed by subscriber, with the tunnel and bearer state. A session can be inspected to see its detail; this is where you confirm a specific subscriber's Wi-Fi-calling tunnel is up.

Diameter

The Diameter peer connections to the AAA/HSS that carry authentication (SWm/SWx), with each peer's connection state — the first place to look when Wi-Fi-calling authentications are failing.

Rate Limiter

The ePDG terminates untrusted-Wi-Fi IPsec/IKEv2 tunnels (SWu) and bridges them to the EPC over S2b, which makes it exposed to any handset — or any source — on the open internet. The **Rate Limiter** is the defence against a client hammering tunnel setup: it throttles IKE flooding so a single misbehaving handset or attacking source cannot exhaust the gateway. The tab shows the enforced **thresholds** — the **per-IP** and **per-IMSI** request limits, each expressed as a number of auth attempts within a rolling **window**, and the **block duration** applied once a limit is breached — alongside a live count of how many **IPs** and **IMSIs** are **currently blocked**.

Below the thresholds you can **look up** any IP address or IMSI to see whether it is blocked and how many failures it has accumulated, and **clear** that block so future auth attempts from it are allowed again; a single **Clear All** action drops every block and resets all failure counters at once. This is where an operator confirms why a subscriber's Wi-Fi-calling tunnel is being refused, and releases a block once the offending source has settled.

GeoIP

GeoIP is the ePDG's country-based admission control — how an operator restricts VoWiFi tunnel establishment to permitted countries by matching each incoming tunnel's source IP against a GeoIP database. The tab shows whether filtering is **enabled** and whether the database is **ready** (loaded and usable), the **mode** — **whitelist** (only the listed countries may connect) or **blacklist** (the listed countries are refused) — and the **country list** the mode applies to. It also shows the **GeoIP database in use** and whether **unknown-country IPs** — source addresses the database cannot resolve to a country — are allowed or blocked.

Once the GeoIP database file has been updated on disk, the tab can **reload** it so the new mapping takes effect without disturbing established tunnels. This is the control that keeps VoWiFi tunnel setup confined to the regions an operator serves.

Configuration

The ePDG's runtime **configuration**, **rate-limiting** rules (per-subscriber / per-source limits that protect the gateway), and **geo-IP** policy (which source regions are permitted to establish tunnels). These are the controls that shape who may connect and how aggressively.

Logs

The **Logs** tab tails the ePDG's live log output for troubleshooting a single gateway without leaving OmniWeb.

Monitoring dashboards

Tunnel rates, throughput and other time-series for the ePDG are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- The **PGW** anchors the ePDG's user plane over **S2b**.
- The **AAA/HSS** holds the subscriber authentication data the ePDG's Diameter requests resolve against.
- TWAG — the *trusted*-Wi-Fi counterpart.

5G Core

The **5G Core** section gives each 5GC network function (NF) its own live dashboard — health, registration, and the operational data that NF owns (UE contexts, PDU sessions, IP pools, NRF registrations, policies, charging sessions) — plus per-NF and per-item actions to clear state. It is the 5GC equivalent of the per-element pages for EPC/IMS functions, reading each NF's OAM API through the OmniWeb backend proxy.

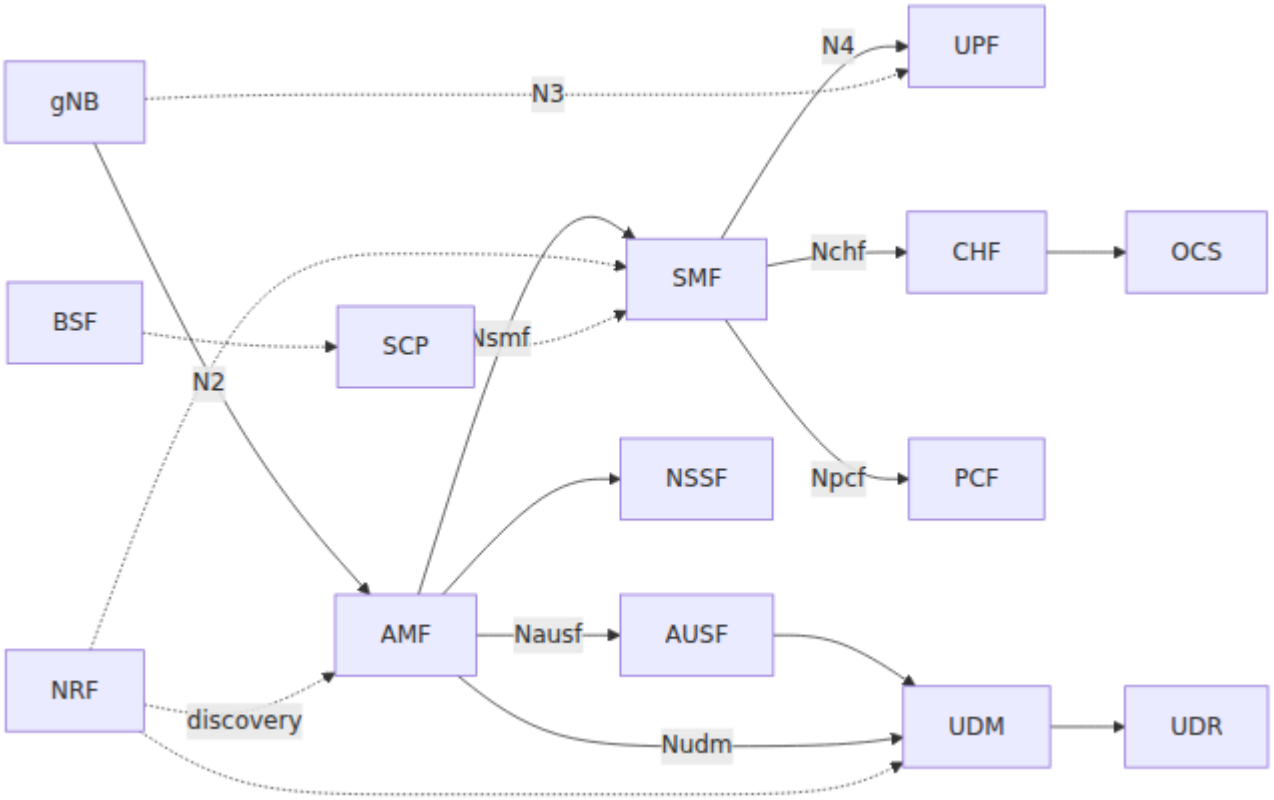
← Operations Guide

When it appears

The **5G Core** group only shows in the sidebar when the selected site actually runs a 5GC control plane — OmniWeb keys this off the presence of an **AMF and/or SMF** in the site's inventory. A lone UPF or CHF shared with the EPC is not a 5GC and does not surface this section.

Which NFs appear, and the address of each NF's OAM API, are derived **directly from the site's Ansible inventory** — the 5GC NF groups defined in the host file — so adding or moving an NF is an inventory change with nothing to hand-template. Each NF resolves to a label, description, IP, and port. OmniWeb serves the resolved registry via `/api/5gc/nf-registry`; the dashboards then query each NF's OAM API through the proxy (`/api/5gc/nf/<id>/…`), so all traffic stays behind the single authenticated gateway — see the configuration model. Deployments that run no inventory (e.g. the hosted service) fall back to a static `fivegc` block in the OmniWeb configuration.

The NFs



Each NF has its own dashboard:

NF	Function	Key data surfaced
NRF	Network Repository Function	Registered NF instances (grouped by type), registered services, subscriptions — see below
AMF	Access & Mobility Management	UE list, connected gNodeBs (with decoded gNB-ID / TAC / PLMN / SST), statistics, maintenance mode
SMF	Session Management	PDU sessions, **IP Pools**, UPF association, statistics
UDM	Unified Data Management	UE list, statistics
UDR	Unified Data Repository	HSS health, subscribers
AUSF	Authentication Server	AUSF (auth) sessions, statistics
PCF	Policy Control	AM policies, SM policies, statistics
CHF	Charging Function	Charging sessions, CGrateS health, CDR records
NSSF	Network Slice Selection	Slices, NSSAI availability, statistics
BSF	Binding Support	PCF bindings, statistics
SCP	Service Communication Proxy	NF health, discovery cache, statistics

Not every deployment runs every NF — only the NFs defined in the site's inventory appear. Each NF has its own reference page — follow the links above.

The SMF's IP Pools tab

As an example of the NF-owned data these dashboards surface: the SMF dashboard has an **IP Pools** tab showing the address pools the SMF allocates UE IPs from, alongside its live **PDU Sessions** and its **UPF Association** (the N4/PFCP link state to the user plane). Together these answer "is the SMF connected to its UPF, what sessions are up, and is it running out of addresses?" without leaving OmniWeb.

A dashboard at a glance

Every NF dashboard follows the same layout:

1. **Health cards** across the top — API Health, NRF Registration (is this NF registered with the NRF?), NF Status, and License. Each polls on an interval; use the **Polling Indicator** in the header to pause or see the last-update time (polling is a shared pattern — see Common Operations).
2. **Actions** — NF-level operations (see below).
3. **Data sections** — the API status detail, NRF registration detail, and the NF-specific tables from the table above, each auto-refreshing. Tables auto-decode what they can: PLMN/TAC/SST from Erlang-encoded S1AP terms, timestamps to relative time, statuses to coloured chips, and cross-links (an NF-type value links to that NF's dashboard).

Actions

Dashboards expose actions to clear NF state — each asks for confirmation first, since they are destructive:

NF	NF-level actions	Per-row actions
AMF	Clear all UE contexts · Clear cache	Deregister a UE
SMF	Release all PDU sessions · Clear cache · PFCP reassociate (re-establish the UPF association)	Release one PDU session
NRF	Deregister all NFs	Deregister one NF · Remove a subscription
AUSF	Clear all auth sessions	Clear one session
UDM	Clear subscriber cache	—
PCF	—	Remove an AM or SM policy

These let an operator recover from stuck state — e.g. force an SMF that lost its UPF to reassociate, drop a UE context that will not clear, or deregister a stale NF from the NRF.

The NRF dashboard

The NRF gets a richer view because it is the registry the whole 5GC discovers through:

- **Metric cards** — NF types, total instances, registered (with a percent-of-total bar), and services/subscriptions.
- **Registered NF instances** — grouped by NF type, each instance card colour-coded and showing registration status, heartbeat timer, service count, and PLMN. Click an instance for its full profile (with an **Open Dashboard** link to that NF and a **Deregister** button).
- **Registered services** — a flat table of every service each instance advertises, its versions, and status.

- **Deregister All NFs** — clears every registration (each NF must then re-register).

This makes the NRF the place to answer "which NFs are up, are they all registered, and what services are they offering?"

Related

- Network Topology — the topology views that link into these dashboards.
- Common Operations — the polling, configuration, and RBAC patterns shared with every element page.
- PGW-C / SMF and UPF — the EPC-side session/user-plane pages for a converged EPC+5GC.

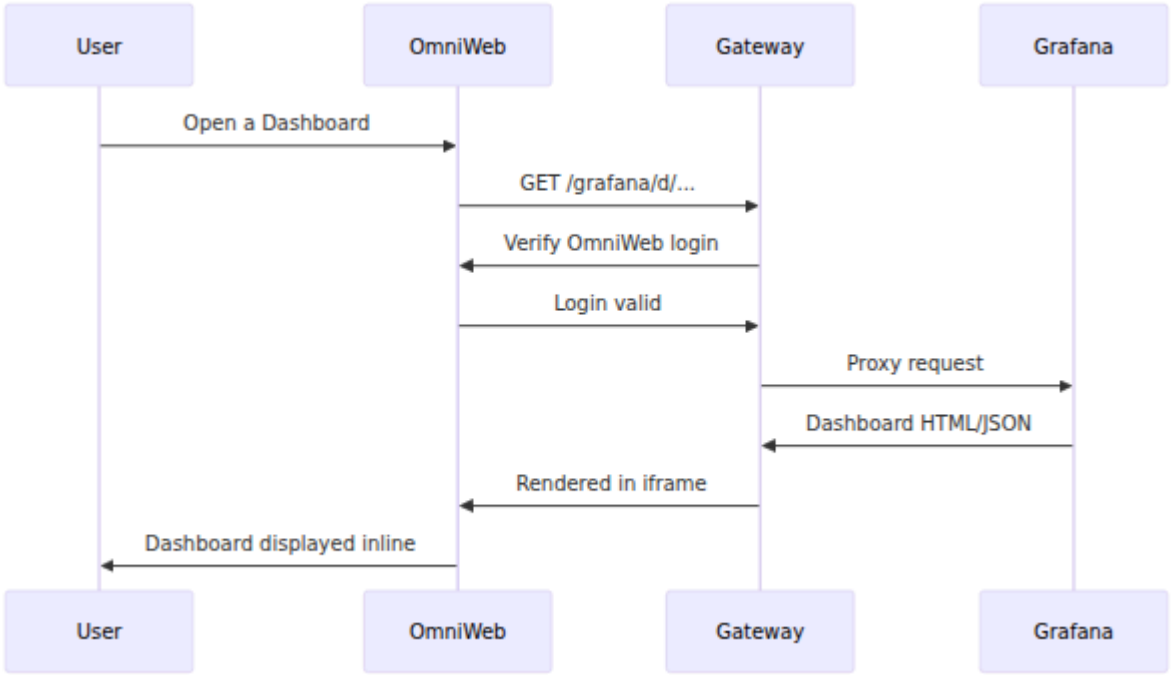
Grafana Dashboards & Statistics

OmniWeb embeds Grafana directly into the portal so operators can see dashboards and statistics for any element — and the network as a whole — without a separate login or browser tab.

← Back to Operations Guide

How It Works

Grafana is served behind the authenticated gateway at the `/grafana/` path and embedded as an iframe inside OmniWeb. Authentication is transparent: when you are logged into OmniWeb, Grafana access is authorised automatically against your OmniWeb login. There is no second password.



The Grafana instance OmniWeb proxies to is configured by the `GRAFANA_URL` environment variable (default `http://localhost:3000`). The gateway handles proxying and the single sign-on check transparently.

Two Ways to Reach Dashboards

1. Per-element Dashboards tab

Elements that are tagged for Grafana show a **Dashboards** tab in their workspace. OmniWeb queries Grafana for dashboards matching that element's tags and renders the relevant one inline. This keeps an element's stats next to its operational data.

Tag matching is case-insensitive. Representative tag sets:

Element	Grafana tags matched
UPF	`upf`
P-CSCF	`pcscf`, `cscf`
I-CSCF	`icscf`, `cscf`
S-CSCF	`scscf`, `cscf`
PGW-C, MME, SMSC, DRA, OCS, TAS	element-specific tags

To make a dashboard appear under an element, tag it in Grafana with that element's tag (for example, tag a UPF dashboard `upf`).

2. Grafana sidebar

The sidebar's **Grafana → Dashboards** entry lists all available dashboards. Selecting one opens it inline within OmniWeb, and the URL updates to reflect the dashboard path so you can deep-link to a specific dashboard.

Statistics

The embedded dashboards are where network-wide and per-element **statistics** live: session counts and rates, throughput, latency, drop rates, Diameter/PFCP

peer health over time, CDR volumes, registration counts, and capacity trends. OmniWeb's own element pages give you the *current* state (and let you act on it); Grafana gives you the *historical* picture and trends. They are designed to be used together — investigate a Grafana alert by jumping to the element's operational pages, logs, and traces.

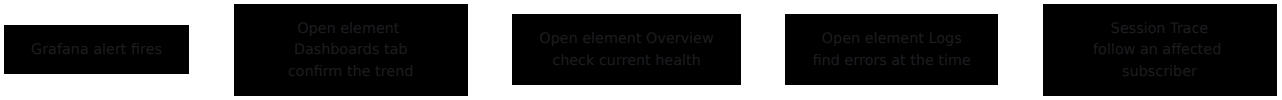
Alerting

Grafana's built-in alerting engine handles all threshold-based alerts. Alert rules are defined in Grafana and can notify via email, Slack, PagerDuty, or webhook. OmniWeb does not duplicate alerting — it provides the operational context (element pages, logs, traces) needed to investigate when an alert fires.

Common alert categories:

Category	Examples
Availability	Element unreachable, Diameter peer down, PFCP association lost
Capacity	IP pool exhaustion, eBPF map utilisation > 80%, VLR subscriber count threshold
Performance	Session setup latency, XDP drop rate, heartbeat failures
Traffic	Abnormal CDR volume, call drop rate, SMS delivery failures

Investigating an Alert — Recommended Flow



1. Confirm the trend in Grafana (Dashboards).
2. Check current state on the element's Overview.

3. Read the element's Logs around the alert time.
4. If subscriber-specific, run a Session Trace. See Logs & Subscriber Tracing.

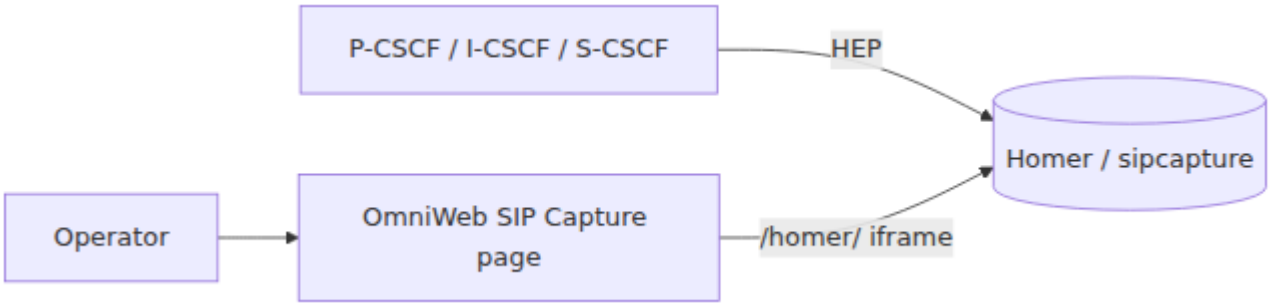
Homer — SIP Capture & Call-Flow Analysis

Homer is a SIP capture and search system (HEP / sipcapture). OmniWeb embeds the Homer web UI directly, so you can search captured SIP traffic and reconstruct a call's signalling ladder without leaving the portal or logging in separately.

← Operations Guide

What it is

Homer collects SIP messages from the IMS (via the HEP protocol) and stores them searchable by call-id, from/to number, time, and more. Its web UI lets you find a call and draw its full **flow diagram** — every SIP message between the endpoints and the CSCFs, with timings and any failure/hangup cause. OmniWeb does not reimplement any of this; it **embeds the Homer app** in an iframe under **SIP Capture** (nested under Session Trace in the sidebar).



Single sign-on

You do not log into Homer separately. OmniWeb proxies the Homer app at `/homer/` and injects the auth-proxy headers on the way through, so the embedded UI is **already authenticated** — no Homer login prompt. It is reached through the same authenticated OmniWeb gateway as Grafana and Loki, so one OmniWeb login governs it too.

The **SIP Capture** entry only appears in the sidebar when Homer is **configured and reachable** for the deployment — OmniWeb probes it and hides the entry if it is not present, so you never click into a dead page.

Homer vs Session Trace

OmniWeb has two complementary ways to look at signalling, and it helps to know which to reach for:

	Homer (SIP Capture)	Session Trace
Scope	SIP only, in depth	The whole network — Diameter, GTP-C, PFCP, SIP, MAP
Question it answers	"What did this *call*'s SIP flow do — every message, timing, and hangup cause?"	"What happened to this *subscriber* across every element?"
Best for	Diagnosing a VoIP/VoLTE call, a failed registration, a specific hangup	Following one IMSI/MSISDN end to end through the core

Reach for Homer when you have a call to dissect; reach for Session Trace when you are following a subscriber across the whole stack. Homer is also linked from the Logs & Subscriber Tracing guide.

Related

- Logs & Subscriber Tracing — cross-network Session Trace and where Homer fits.
- Grafana Dashboards & Statistics — the other embedded monitoring tool, reached the same authenticated way.

HSS — Home Subscriber Server

The **HSS** (Home Subscriber Server) is the master subscriber database for the mobile core. It holds every subscriber's identities (IMSI, MSISDN and SIM/ICCID), the authentication key material that proves who they are, and the EPC, IMS, PCRF and roaming service profiles that describe what each subscriber is allowed to do on the network. The MME reads it over S6a to authenticate and locate a device and to learn its EPC subscription; the CSCF reads it over Cx for IMS registration and iFC service triggers; and the PCRF draws its policy and charging rules from it. Its dashboard is where an operator provisions and searches subscribers, curates the shared service profiles, and checks the HSS's own health.

← Operations Guide

The dashboard is a per-instance element page reached from the sidebar when an HSS runs in the site's inventory. All traffic goes through the OmniWeb backend proxy to the HSS's API, so it stays behind the single authenticated gateway; responses come back in a `{status, response}` envelope. A documentation link in the page header gives in-context help. The page is organised into tabs.

Subscribers

The landing tab is the subscriber directory. Search **by IMSI, MSISDN or ICCID** to find a record — the search also resolves a subscriber **by IP address** when a live session is being traced back to its owner. Opening a record shows the subscriber overview: a relationship diagram tying the subscriber to its **identity** (IMSI), its **status** (enabled, and whether IMS is enabled), and the **EPC**, **IMS** and **roaming** profiles assigned to it, with the profile's APNs shown beneath the EPC profile. Sensitive key values are held but not displayed — they can be updated but read back as redacted.

Provisioning

Creating or editing a subscriber is done from the provisioning form, which brings the identity, credentials, SIM and profiles together in one place:

- **Subscriber identity** — IMSI, a subscriber name, and the **enabled** and **IMS-enabled** switches.
- **Authentication key set** — the authentication algorithm, **Ki** and **OPc** (or **OP**), **AMF**, **SQN**, and the SIP password used for IMS.
- **SIM card** — the **ICCID**, whether it is an eSIM, the PIN/PUK values and the batch and vendor details.
- **Phone numbers** — assign one or more **MSISDNs**, searching existing numbers as you type so a number is not created twice.
- **Assign profiles** — attach the **EPC**, **IMS** and **roaming** profiles that give the subscriber its service.

Existing subscribers can be updated the same way, and a subscriber can be deleted from its record.

SIMs

The SIM inventory — every **SIM** the HSS knows, searchable **by ICCID**. Each SIM carries its **ICCID**, eSIM flag, batch name and vendor, its transport-key values (**KIC**, **KID**) and eSIM **LPA**, and the full set of **ADM keys**. The authentication **key sets** that back the SIMs (Ki/OPc/AMF/SQN) are managed here as the credential store the subscriber records reference. SIMs can be created, edited and deleted.

MSISDNs

The MSISDN directory — the pool of provisioned phone numbers. Search **by digits** for a partial match to find or check a number before assigning it. Numbers can be created, edited and deleted, and are attached to subscribers during provisioning.

Service profiles

Service profiles are the reusable building blocks that describe what a subscriber can do; a subscriber references them rather than carrying its own copy, so a change to a profile applies to everyone using it.

EPC profiles

The packet-core subscription the MME enforces. An **EPC profile** sets the **network access mode**, the UE-AMBR **downlink/uplink** aggregate bandwidth, the **TAU interval**, and the list of **APN profiles** the subscriber may use, with one nominated as the **default APN**. Each **APN profile** ties an **APN identifier** to an **APN QoS profile** and a **PCRF profile**, and can be marked as the default. An EPC profile must reference at least one APN profile, and the default APN must be one of the assigned APNs.

Expanding an EPC profile lists its APN profiles. **Add APN** attaches an APN to the profile, and the unlink action on each row **removes it from this EPC profile only** — the APN profile itself is kept and can be re-added later or reused by other EPC profiles. Because a profile must keep at least one APN, the last one cannot be removed. The default APN is cleared automatically if the APN it pointed at is removed.

APN Profiles

The APN Profiles tab lists every APN profile independently of the EPC profiles that use it, and shows its **APN identifier**, **APN QoS profile** and **PCRF profile**. The **Used By** column names each EPC profile that references the APN, or marks it **Orphaned** when no EPC profile does — the state an APN reaches once it is removed from the last EPC profile that used it. The **Orphaned only** toggle filters the list to just those, so unused APNs can be found and cleaned up.

Profiles can be created here (a new profile starts unattached — link it to a subscriber's service by adding it to an EPC profile), edited, and deleted. An APN profile that is still linked to one or more EPC profiles cannot be deleted; remove it from those profiles first, then delete it from this tab.

APN QoS Profiles

The APN QoS Profiles tab manages the reusable APN-level QoS templates that EPC and APN profiles point at, so that every subscriber whose APN references a template gets the same quality of service on its default bearer. Each template carries a **name**, a **QCI** (the QoS Class Identifier that maps the bearer to a

standardised delay and loss behaviour), an **ARP** — the allocation/retention priority, with **pre-emption capability** and **pre-emption vulnerability** flags that decide whether the bearer may bump, or be bumped by, others when resources are scarce — the **APN-AMBR** uplink and downlink aggregate rates, and whether **online** and **offline charging** are enabled.

Templates can be created, edited, cloned to derive a new variant, and deleted. Because an APN profile references a QoS template rather than carrying its own copy, editing a template re-shapes the QoS of every APN that uses it, and a template that is still referenced by an APN profile cannot be deleted until those APNs are reassigned first.

APN QoS profiles

The per-APN quality-of-service settings an APN profile points at: the **QCI**, **ARP** and pre-emption **capability/vulnerability**, the APN-AMBR **downlink/uplink** bandwidth, and whether **online (Gy)** and **offline (Rf)** charging apply. Profiles can be cloned to derive a new variant from an existing one.

IMS profiles

The IMS subscription the CSCF reads over Cx. An **IMS profile** carries an **iFC template** — the initial-Filter-Criteria that decide which application servers a subscriber's SIP traffic is routed to. Profiles can be created, edited and deleted.

Static IPs

Fixed IP allocations for subscribers that must always receive the same address, managed alongside the EPC subscription data.

PCRF

The policy layer the PCRF applies. A relationship view links the three parts:

- **PCRF profiles** — the named policy a subscriber's APN references, made up of a set of charging rules.
- **Charging rules** — each with a **rule type** and **precedence**, a **QCI** and **ARP**, MBR/GBR **downlink/uplink** limits, and a **rating group** and **service identifier** for charging; a rule with an MBR or GBR is carried on a dedicated bearer.

- **Flow-information rules** — the traffic flow templates attached to a charging rule that define which packets it matches.

All three can be created, edited and deleted; editing a charging rule affects every profile that uses it.

Roaming

The roaming policy the HSS applies to inbound and outbound subscribers. A **roaming profile** sets a **default data action** and **default IMS action** and a list of **roaming rules**; each rule keys off an **MCC/MNC** and sets the **data** and **IMS action** for that network, so specific visited networks can be allowed or blocked while the profile default covers the rest. An operator lookup finds a network's MCC/MNC from an operator name (from IR.21) when adding a rule. Rules are shared — editing a rule affects every profile that uses it.

EIR

The Equipment Identity Register. Each **EIR rule** matches an **IMEI** by regular expression and sets an **action** — allow or block — so a device model or a specific handset can be barred from the network. A device-model search helps build the IMEI pattern. Rules can be created, edited and deleted.

Status

A single-screen health read of the HSS itself: the **API**, **database** and **license** health, and the **entity counts** — total **subscribers**, **EPC profiles**, **IMS profiles**, **SIMs**, **MSISDNs** and **roaming profiles**. It is the quickest confirmation that the HSS is up and how much it is carrying. Monitoring and statistics dashboards are provided as Grafana embeds.

The **Replication** tab shows the status of database replication to any peer HSS instances, and the **Logs** tab tails the HSS's live log output for troubleshooting a single instance without leaving OmniWeb.

Replication

The Replication tab reads the PostgreSQL replication health of the HSS node's subscriber database and draws it as a live topology. Each node shows whether its database is a **primary** or a **standby** — a standby being a node that is in recovery, replaying the primary's changes rather than accepting writes of its own. For a primary, every connected standby is drawn as a downstream link labelled with the standby's **client address**, the **replication slot** it streams through, its **streaming state**, and the **replication lag** measured against the primary's current WAL position — shown in bytes so a replica that is falling behind stands out at a glance. Whether each replication slot is currently **active** determines whether its link reads as healthy or broken.

This is how an operator confirms that the subscriber database is replicating across the HSS cluster — that a change provisioned on one node will reach the others, and that there is a standby to fail over to. A lagging replica shows a growing lag figure; a broken or disconnected one drops its link, and if two or more HSS instances are running but none of them replicate, the tab flags the whole cluster as a replication gap so it can be corrected before it becomes a data-loss risk.

Operations

Two subscriber-level operations act on a live session rather than on the stored record:

- **Cancel-Location (CLR)** — send an S6a Cancel-Location-Request to a subscriber's serving MME, detaching the subscriber so it re-attaches and picks up changed subscription data.
- **PCRF Re-Auth** — request that a subscriber be re-authorised for a given PDN session, pushing updated policy or charging rules onto an active session.

Related

- **MME** — reads subscriber authentication and EPC subscription data over **S6a**; the target of a Cancel-Location.
- **CSCF** — reads the IMS profile and iFC over **Cx** for IMS registration and service triggering.
- **PCRF / PCF** — applies the PCRF profiles, charging rules and flow rules held here; the target of a Re-Auth.
- **UDR** — the shared subscriber data store the HSS front-ends in a 5G core.
- Operations Guide — back to the element index.

LCS — Location Services (OmniLCS)

The **LCS** function provides **location services** for the core — it answers "where is this subscriber?" and drives cell-broadcast warnings. It resolves a UE's position from the serving cell (and configured cell geometry), services location requests from the network (immediate and deferred), and pushes **Commercial Mobile Alerts / cell-broadcast** warnings to an area. OmniWeb surfaces the LCS's cell inventory, live and recent location fixes, deferred-location requests, CAP alerts and broadcasts; longer-run trends are on the Grafana dashboards reached from the same page.

← Operations Guide

The LCS is a per-instance element page reached from the sidebar when one runs in the site's inventory. All traffic goes through the OmniWeb backend proxy, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

A single-screen health read: the LCS **status**, its **Diameter peers** (to the MME/serving network), **active** and **completed** location sessions, the number of **cells loaded**, and the **cell-sync** result — the quickest confirmation the LCS is up and has current cell geometry.

Cells

The **cell inventory** the LCS positions against: each cell's id, latitude/longitude, radius, height, TAC/LAC, PCI, EARFCN, azimuth and RAT. Cells can be added, edited and removed here — this geometry is what turns a serving-cell id into a position.

Location

Recent **location fixes**: each fix's subscriber, resolving cell, computed position, accuracy and method. A location can be **requested** for a subscriber from here, and opening a fix shows its authoritative last-known position. This is where an operator confirms a positioning result.

Deferred

The **outstanding deferred-location requests** the LCS is holding. Where the Location tab answers "where is this subscriber *now*", a **deferred MT-LR** (deferred mobile-terminated location request) asks the network to report a subscriber's position **later, when a trigger fires**, rather than immediately — the GMLC arms the request and the report comes back when the condition is met. The common triggers are **UE availability** (report the next time the device becomes reachable, having been idle or unreachable when the request was made), **periodic** location (report at a fixed **interval** for a set **number of reports**), and area/motion events (report when the UE enters, leaves or moves within a designated area). This is how an operator schedules positioning for a subscriber who cannot be found at the moment or who needs to be tracked over time.

Each row lists the deferred session's subscriber by **IMSI**, its **event/trigger type** (periodic or triggered), the reporting **interval** in seconds and the **count** of reports to send, and its **status** — pending while it waits for the trigger, active once reporting has begun. An operator can **create** a new deferred request from here — naming the target IMSI, the trigger type and, for periodic reporting, the interval and count — and **cancel** an outstanding session at any

time, which tears down the armed request so no further reports are produced. The list polls so a session's status updates as the network arms it and the trigger fires.

CAP Alerts

The **Common Alerting Protocol (CAP)** alerts the LCS knows about — each with its identifier, status, urgency, severity and target area. These are the emergency/warning messages eligible to be broadcast.

Broadcasts

The **Broadcasts** tab shows the active **cell-broadcast** messages (warning type, message identifier, serial, target cells and status) — the alerts currently being pushed to an area.

Logs

The **Logs** tab tails the LCS's live log output for troubleshooting without leaving OmniWeb.

Monitoring dashboards

Location-request rates and other time-series are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- The **MME** requests positioning over the location interfaces and is the LCS's Diameter peer.
- **CAP alerts** and **broadcasts** feed the cell-broadcast warning system.

License Server — Omnitouch Licensing

The **License Server** is the Omnitouch licensing authority the network functions check against. Every Omnitouch product — the MME, HSS, DRA, PGW-C, UPF, SGW-C, SMSC, CSCF and the rest — validates its entitlement against the License Server before it runs, asking whether a given **licensee** is licensed for that **product** and whether the licence is still within its validity window. The server holds cryptographically **signed licences**, each granting a licensee a set of products with a validity period and metadata, and it answers entitlement queries and records every request it is asked. Because a licence is only meaningful against a trustworthy clock, the server also tracks a **trusted time** source so a licence cannot be revived by rolling a node's clock back. OmniWeb surfaces what is licensed and its health, the log of entitlement requests network functions have made, an on-demand query tool, and the server's live logs.

← Operations Guide

The License Server is a per-instance element page reached from the sidebar when one runs in the site's inventory, with a fleet view where several are deployed. All traffic goes through the OmniWeb backend proxy to the License Server's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview / Status

A single-screen read of what the server is licensing and whether it is healthy. Summary cards give the count of **total products**, how many are **valid** and **invalid**, and the number of distinct **licensees**. A **trusted time** panel shows the state of the server's clock source — **valid**, **grace period** or **invalid** — because when trusted time is unavailable the server cannot safely validate licences at all, and that raises the page's health to a warning or error.

The **Licensed Products** table lists one row per licensed product, each showing the **product**, the **licensee** it is granted to, when it was **issued**, its **start** and **expiry** dates (with a "days remaining" or "expired N days ago" indicator that turns amber within 30 days of expiry and red once past it), and its **valid/invalid** status. Opening a row expands the full detail: the exact validity timestamps, whether the **cryptographic signature** is valid, any product metadata, and the signature itself. This is the quickest confirmation that the network's functions are all licensed and that nothing is about to lapse.

Request Logs

The audit trail of **entitlement requests** the network functions have made against the server. Each row records the **timestamp**, the **source IP** the request came from (which network-function host asked), the **licensee** and **product** it asked about, the **decision** — **allowed** or **rejected** — and, for a rejection, the **reason**. The list can be filtered by decision, by product, and by a partial **source IP** or **licensee** search, and it is paged. This is where you see, for example, that a newly deployed function is being rejected because its product was never licensed, or confirm that a function is successfully checking in.

License Query

An on-demand **entitlement check**. Enter a **licensee** and a **product** and the tool asks the server the same question a network function would — is this licensee entitled to this product right now — and returns the verdict: **valid** or **invalid**, the licence's **valid-to** date, the response **timestamp**, any **metadata**, and the **signature**. It is the direct way to confirm a specific entitlement without waiting for the owning function to check in, and it lists the known product identifiers (for example `omnihss`, `omniupf`, `omnimme`) so you can query the exact one.

Logs

The **Logs** tab tails the License Server's live log output for troubleshooting a single instance without leaving OmniWeb.

Related

- **Every Omnitouch network function** — the MME, HSS, DRA, PGW-C, UPF, SGW-C, SMSC, CSCF and the rest validate their entitlement here; each product's own **Overview / Status** tab shows a licence-health badge derived from this check.
- Operations Guide — back to the element index.

Logs & Subscriber Tracing

OmniWeb gives operators several complementary, real-time views into what the network is doing:

1. **Logs** — structured, live service logs for any single element, powered by Loki.
2. **Session Trace** — a cross-network view that follows one subscriber across every protocol and element (Diameter, GTP-C, PFCP, SIP, MAP).
3. **Homer** — deep SIP/VoIP packet capture and call-flow analysis.

← Back to Operations Guide

Jump to a Subscriber from an NF

Wherever a network function shows a subscriber, UE, or session by IMSI — the **MME** (UEs), **SGW-C**, **PGW-C** and **ePDG** (Sessions), **MSC** and **TAS** (Subscribers), and **SMSC** (Location) — a row of quick-link buttons lets you pivot straight to that subscriber elsewhere, without copying the IMSI around:

- **Go to HSS Record** — opens the subscriber's provisioning record on the HSS (resolved by IMSI across the configured HSS instances).
- **View Session Trace** — opens the subscriber's live signalling ladder in Session Trace, scoped to that IMSI.
- **View Logs** — opens *this* element's Logs tab pre-filtered to that IMSI, so you see only the log lines for that subscriber.

Each button only appears when it applies — the IMSI is known, and an HSS or Session Trace is configured for the site — so the set adapts to the deployment.

The subscriber quick-links (here on an MME UE) — one click to the HSS record, the live signalling ladder, or this element's logs filtered to the subscriber.

Real-Time Logs (Loki)

Every element has a **Logs** tab. It queries the element's service logs from Loki and streams new lines as they arrive, so you can watch an element live while you make a change or reproduce an issue.

The Logs tab (here on a UPF) — structured, colour-coded log lines streamed from Loki, with service, severity, protocol, and subscriber (IMSI) filters. Every element exposes the same viewer.

How log queries work

The frontend builds a LogQL query and sends it through the backend's Loki gateway. Nothing queries Loki directly from the browser — it is proxied (and therefore authenticated and audited) like every other request.

The backend's Loki target is set by the `LOKI_URL` environment variable (default `http://localhost:3100`). Queries use Loki's range-query API with a LogQL expression, a time window, a result limit, and a direction (newest-first or oldest-first).

Selectors vs. pipeline filters

OmniWeb builds structured queries from two kinds of filter:

Filter placement	LogQL position	Example	Use
Selector	Inside the stream selector `{ … }`	`{hostname="opt-dev-upf01"}`	Pick which streams (which host/service) to read. Indexed and fast.
Pipeline	After a `|`	`| level="error"`	Filter the matched lines by a structured field.

A typical element Logs tab pins the host/service as a selector and lets you add pipeline filters (severity, component, subscriber identifiers) on top. Because Omnitouch services emit **structured** log lines, you can filter on real fields rather than grepping free text.

Structured logging data

Hosts ship structured logs to Loki (configured by the Ansible common role). Two especially useful structured sources:

- **Service logs** — each NF's own structured output, labelled by `hostname`/service, available on that element's Logs tab.
- **Command audit** — every command executed on a host, including who logged in (surviving `sudo`/`su`), the effective UID, the binary, the working directory, and the full command line. Shipped to Loki. See Audit Logging for the field reference.

Using the Logs tab

1. Open any element and select **Logs**.
2. The viewer loads recent lines for that element and begins polling for new ones.
3. Narrow the view with pipeline filters (e.g. severity, or an IMSI/MSISDN if the service logs include it).

4. Adjust the time window to look back at a past event.
5. Pause polling (header indicator) to hold a stable view while reading.

> **Tip:** When investigating one subscriber, filtering element logs by their IMSI/MSISDN is the fast first step; to see the subscriber across *multiple* elements at once, use Session Trace below.

Session Trace — Follow a Subscriber Across the Network

The **Session Trace** engine correlates signalling for a single subscriber across multiple elements and protocols, giving one unified timeline from radio attach through authentication, session establishment, and call/SMS delivery. This is the tool for answering "what happened to this subscriber?" without hopping between element CLIs.

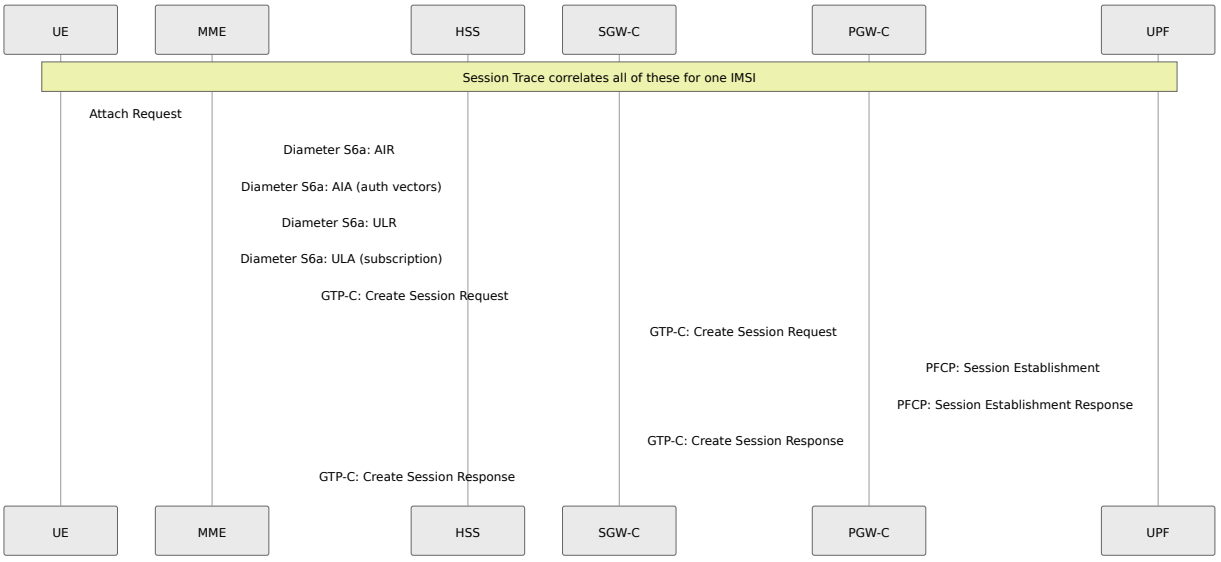
Session Trace — search by IMSI or MSISDN and correlate a subscriber's activity across Diameter, GTP-C, PFCP, SIP, and MAP, with per-protocol tabs across the top.

Supported protocols

Protocol	What it shows	Elements involved
Diameter	Authentication, policy, charging	HSS, DRA, PGW-C, PCRF, OCS, CSCF (Cx/Dx/Rx)
GTP-C	Session management (S5/S8, S11)	SGW-C, PGW-C, MME
PFCP	User-plane session rules (N4)	PGW-C/SMF ↔ UPF
SIP	IMS call control	P-CSCF, I-CSCF, S-CSCF, TAS
MAP	SS7 location, authentication, SMS	MSC, HLR, CAMEL GW, IP-SM-GW

User Trace

The **User Trace** view lets you search by **IMSI** or **MSISDN** and see every correlated protocol interaction for that subscriber across the network. A single attach, for example, surfaces the Diameter authentication, the GTP-C session creation, and the PFCP rule installation as one coherent sequence:



Protocol-specific tabs

Each protocol has its own tab — **Diameter, GTP-C, PFCP, SIP, MAP** — plus an **All Protocols** overview. The per-protocol tabs are filtered views, useful when you are chasing a protocol-specific problem (e.g. only the SIP exchange of a VoLTE call) rather than following a single subscriber across everything.

PCAP Ladder

The **PCAP Ladder** tab turns a packet capture into a ladder — a time-ordered sequence diagram of the signalling. Upload a PCAP or PCAPNG capture (for example from a network tap or an element-level trace) and OmniWeb parses the packets, identifies the signalling nodes by IP address and inferred role (MSC/VLR, HSS, S-CSCF, and so on), and correlates requests with their responses into sessions. The result is drawn as a message ladder across those nodes, ordered in time, so you can read a whole exchange top to bottom at a glance.

PCAP Ladder — an uploaded capture rendered as a ladder across the detected nodes. Click any message to see its decoded frame and hex.

Click a message on the ladder to open its full decode and raw hex, the same detail you would get from a protocol analyser. The tool recognises the signalling families you meet across a mobile core: MAP, CAMEL, TCAP, Diameter, SIP, GTP-C, PFCP, and GSM SMS. Every capture you upload is listed on the tab so it can be re-opened or shared with a colleague rather than passing the file around.

Live Flows

The **Live Flows** tab shows the same ladder view, but sourced live from the network's structured signalling logs (via Loki) rather than an uploaded file. As the network elements emit signalling telemetry, recent per-subscriber call flows appear here automatically — grouped by IMSI — with no capture to upload. Selecting a flow reconstructs it into the same ladder as a PCAP, so a live flow is as rich as one read from a file, down to per-message decode and hex.

Live Flows — recent call flows drawn straight from the network's signalling telemetry. Filter by IMSI, node, or protocol, then open a flow to render its live ladder.

Open a flow and it is reconstructed into the same ladder as a PCAP — but from the network's own telemetry, with nothing to capture or upload. The example

below is a single subscriber's LTE attach, correlated across every element that touched it: the S1AP/NAS exchange with the **eNodeB**, the Diameter S6a authentication and location update against the **HSS**, and the GTP-C session setup through the **SGW-C** and **PGW-C** — all on one time-ordered ladder, with per-message request/response round-trip times and full decode a click away.

A live flow opened into its ladder — one subscriber's LTE attach across the eNodeB, MME, HSS, SGW-C and PGW-C, correlating S1AP, Diameter S6a and GTP-C. Click any message for its decode and hex, or any node to focus its exchanges.

If no signalling telemetry is flowing yet, the tab shows an empty state; flows populate on their own as the network elements start emitting telemetry.

Homer — SIP Capture & Call-Flow Analysis

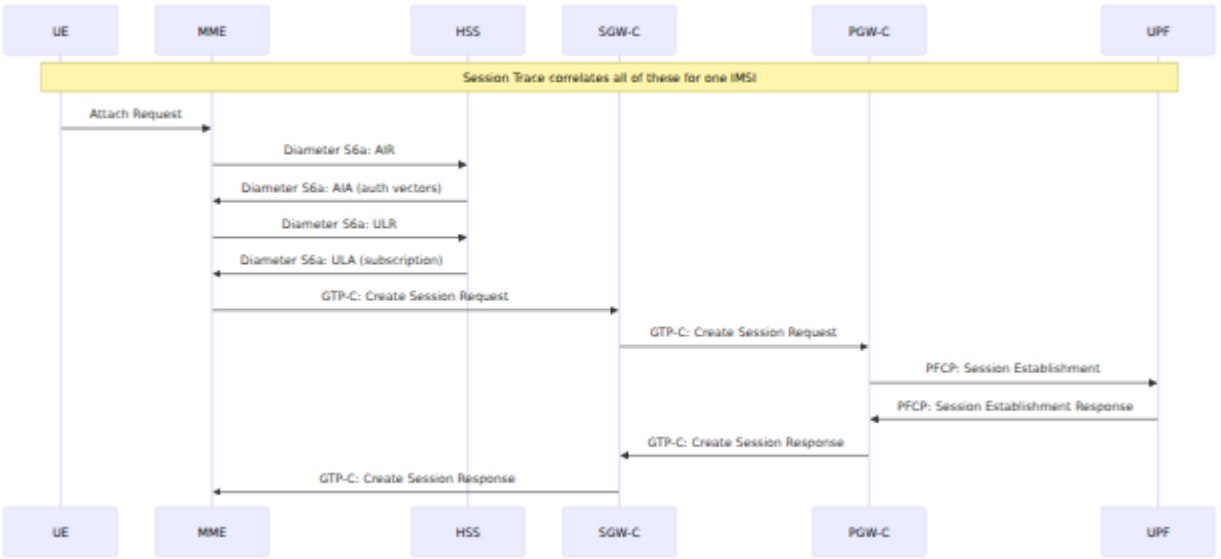
OmniWeb integrates with Homer, the open-source SIP capture and VoIP troubleshooting platform, for deep call-flow visualisation, protocol decoding,

and search across captured SIP signalling. Where Session Trace gives a cross-protocol *summary* for one subscriber, Homer gives the full, message-by-message SIP packet detail.

Availability & access

Homer is **optional** and auto-detected. On startup OmniWeb checks `/api/homer/status`:

- If Homer is configured (`HOMER_URL` set on the backend) and reachable, a **Homer** entry appears in the sidebar.
- If Homer is not deployed, the entry is simply hidden — nothing else changes.



Clicking **Homer** opens the Homer web UI in a new browser tab. Authentication is transparent: the gateway verifies your OmniWeb login via the same single sign-on used for Grafana, so there is no second login. The backend's `HOMER_URL` (and `HOMER_PROXY_SECRET`, when set) configure the integration.

Use cases

Homer is the tool to reach for when a problem is in the SIP layer itself:

- **VoLTE call-flow analysis** — trace SIP `INVITE`/`BYE` sequences through P-CSCF, I-CSCF, S-CSCF, and TAS.
- **VoWiFi troubleshooting** — analyse ePDG ↔ P-CSCF SIP interactions.

- **IMS registration debugging** — inspect `REGISTER` / `401` / `REGISTER` / `200 OK` exchanges.
- **Interconnect issues** — capture SIP-trunk signalling between the MSC and SIP peers.

Session Trace vs. Homer

Both help with troubleshooting, but they answer different questions — they are complementary, not alternatives:

Tool	Best for	Scope	Where it opens
Session Trace	"Follow this subscriber across the whole network"	Diameter, GTP-C, PFCP, SIP, MAP correlated by IMSI/MSISDN	Inline in OmniWeb
Homer	"Show me the exact SIP messages for this call"	SIP/VoIP signalling, full packet decode (VoLTE/VoWiFi/IMS)	New tab (auto-detected)

A common flow is to spot a SIP-related anomaly in Session Trace, then jump to Homer for the detailed call flow.

Troubleshooting

No logs appear on an element's Logs tab

Symptoms: The Logs tab is empty or shows "no results".

Possible causes:

- Loki is not reachable from the backend (`LOKI_URL` misconfigured).

- The host is not shipping logs to Loki (log shipping not running / misconfigured).
- The selected time window predates the available logs, or the host label does not match.

Resolution:

1. Confirm `LOKI_URL` points to a reachable Loki instance.
2. Verify the element's host is shipping logs (check the log-shipping agent on that host).
3. Widen the time window and remove pipeline filters to confirm any lines arrive.

Session Trace returns nothing for a subscriber

Symptoms: A User Trace by IMSI/MSISDN shows no events.

Possible causes:

- The subscriber was inactive in the selected window.
- The identifier is mistyped (IMSI vs. MSISDN).
- Trace sources for the relevant protocol are not feeding the engine.

Resolution:

1. Re-check the identifier and try the alternate one (IMSI ↔ MSISDN).
2. Widen the time window.
3. Confirm with the element's own Logs tab that the subscriber was active at all.

The Homer entry is missing from the sidebar

Symptoms: No **Homer** link appears.

Possible causes:

- Homer is not deployed, or `HOMER_URL` is not set on the backend.
- Homer is unreachable, so `/api/homer/status` reports it as unavailable.

Resolution:

1. Confirm Homer is deployed and `HOMER_URL` is set on the OmniWeb backend.
2. Verify the backend can reach Homer, then reload OmniWeb (detection runs at startup).

MME — Mobility Management Entity

The **MME** is the control-plane anchor of the EPC. It terminates **S1AP** to the eNodeBs, runs the attach, authentication and mobility procedures for every UE, and holds a **UE context** for each attached subscriber. It fetches subscription and authentication data from the HSS over **S6a** (Diameter), sets up bearers on the Serving Gateway over **S11**, and coordinates inter-MME and inter-system handovers over **S10/N26**. In short, it is the node that decides where and how each subscriber attaches, is paged and moves through the network. OmniWeb surfaces the MME's live eNodeBs, UE contexts, SGW and Diameter peers, interface states and handover status; longer-run trends are on the Grafana dashboards reached from the same page.

← Operations Guide

The MME is a per-instance element page reached from the sidebar when one runs in the site's inventory, and — where several are deployed — a fleet view for aggregate status. All traffic goes through the OmniWeb backend proxy to the MME's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

A single-screen health read: the **API status** of the MME and a summary of its live load — connected **eNodeBs**, attached **UEs**, **SGW peers** and the state of its **Diameter** (S6a) peers. It is the quickest way to confirm the MME is up and see how much of the access network it is serving.

eNodeBs

Every eNodeB the MME has an S1AP association with, each row showing the **global eNB ID**, name and connection **state**. Opening an eNodeB shows its context — the served cells and TACs and the UEs on it. An operator can **update the MME capacity** advertised to an eNodeB, which adjusts how the eNodeB load-balances UEs across the MME pool (used to drain or weight an MME for maintenance or rebalancing).

UEs

Every attached subscriber's **UE context**, each row keyed by **IMSI** and **GUTI** with its connection and mobility state and serving eNodeB. Opening a UE shows its full context. A range of per-UE actions is available:

- **Page** the UE.
- **Send to idle** — release the radio connection while keeping the context.
- **Offload** the UE to another MME.
- **Request re-attach** — force the UE to re-attach.
- **Send SMS** to the UE.
- **Delete** the UE context — clear a stale or stuck subscriber.

You can also **look up the operator / PLMN** for an IMSI to identify home versus roaming subscribers.

S-GW Peers

The Serving Gateways the MME talks to over **S11** (GTP-C), each row showing the peer's **IP**, its **liveness / reachability**, the GTP **restart (recovery) counter** — which increments when the S-GW restarts, telling the MME to re-establish the sessions it was holding — and the **session count** anchored on that gateway. Whether a peer was learned by **DNS** or is a **static** entry is shown alongside the S-GW **selection method** the MME applies for home and roaming subscribers. Opening a peer lists the **sessions** it carries, each identified by **IMSI** with its **S11 TEIDs** (the MME- and S-GW-side tunnel endpoints). It is where an operator confirms which S-GWs are up and sees how the UE sessions are distributed across them — useful when balancing load or diagnosing why bearers for a group of subscribers went down with a single gateway.

GUTI reallocation and S10 offload

Two mobility actions are driven by IMSI rather than per row:

- **GUTI reallocation** — reallocate a connected subscriber's **GUTI**, refreshing the temporary identity the UE presents (used when rotating identifiers or

clearing an identity mismatch).

- **S10 offload** — force-offload a subscriber to **another MME**. This moves a specific subscriber's context off this MME over S10, complementing the eNodeB-capacity control when draining an MME.

S10

The **S10/N26 inter-MME** interface — the path over which a UE's context is relocated when it moves between MMEs in a pool during a **TAU** or **handover** (and, over **N26**, between the MME and an AMF for EPC-to-5GC interworking). The tab has two parts. **Active handovers** lists the relocations in flight, each showing the **source** and **target** MME and the **IMSI** being moved, so an operator can watch a mobility event proceed or spot one that is stuck. **Peer MME discovery** shows how the MME finds its pool peers — the **DNS / NAPTR** lookups it runs and any **static** peers configured — with each peer's FQDN and address. It is where you confirm inter-MME mobility is healthy and that the MME can actually reach the neighbours it needs to hand contexts to, complementing the per-subscriber **S10 offload** action driven from the UEs tab.

Diameter

The MME's Diameter peers — principally the HSS over **S6a** — each row showing the peer **host**, **realm** and connection **state**. Opening a peer shows its detail. This is where you confirm the authentication and subscription path to the HSS is up when attaches are failing.

Interfaces

The **Interfaces** view reports the state of every MME interface in one place: **S1AP** to the eNodeBs, **GTP-C** (S11) to the SGW, **Diameter** for **S6a** (HSS) and **SLg** (location services), **SGsAP** (to the MSC for CS fallback and SMS), **LCS-AP / SLs** (location services), and **S10/N26** (inter-MME and EPC-to-5GC handover). It is the single screen for confirming which of the MME's signalling paths are up and which are down.

Logs

The **Logs** tab tails the MME's live log output for troubleshooting a single MME without leaving OmniWeb.

Monitoring dashboards

Attach rates, UE counts, handover rates and other time-series for the MME are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- The **HSS** holds the subscription and authentication data the MME fetches over **S6a**.
- The **SGW-C** is the control peer over **S11** — the MME drives session and bearer setup on it.
- The **MSC** is reached over **SGsAP** for CS fallback and SMS.
- Other **MMEs** are reached over **S10** for inter-MME mobility, and the **AMF** over **N26** for EPC-to-5GC handover.

Quectel EP06 AT Command Reference (cellular network testing)

Reference for the Test Devices modem console. Targets the **Quectel EP06** (firmware `EP06ELAR04A05M4G`, IMEI `868186042000040`) on the lab container, but applies to the EC25 / EG06 / EG06 / EM06 family. Every command below was tested against the live EP06; example responses are the **real responses captured from that modem** (it was in a `NO SERVICE / SEARCH` state at the time, so signal fields read empty — the parsers must handle that).

The relay is request/response: `sendModemAt(serial, command, readMs, timeoutMs)` → backend `modem_at` → agent `modem.run_at(serial, command, min_read, timeout)`. `min_read` keeps reading the port for at least that long so async URCs are captured before the call returns. There is **no persistent URC stream** — see "Event logging" below.

Device / SIM info

Command	Purpose	Live response
`ATI`	Manufacturer / model / firmware	`Quectel` / `EP06` / `Revision: EP06ELAR04A05M4G`
`AT+CGMM`	Model	`EP06`
`AT+CGMR`	Firmware revision	`EP06ELAR04A05M4G`
`AT+CGSN`	IMEI	`868186042000040`
`AT+CIMI`	IMSI	(SIM IMSI)
`AT+QCCID`	ICCID	(SIM ICCID)
`AT+CPIN?`	SIM PIN state	`+CPIN: READY`

Signal / measurement

Command	Purpose	Live response
`AT+CSQ`	RSSI/BER (legacy)	`+CSQ: 99,99` (99 = no signal). dBm = `-113 + 2*rssi`.
`AT+QCSQ`	Per-RAT signal	`+QCSQ: "NOSERVICE"`. When attached: `+QCSQ: "LTE",<rssi>,<rsrp>,<sinr>,<rsrq>`.
`AT+QNWINFO`	Current RAT / band / channel	`+QNWINFO: No Service`. When attached: `+QNWINFO: "FDD LTE","<mccmnc>","LTE BAND 3",1850`.
`AT+QENG="servingcell"`	Full serving-cell measurements	`+QENG: "servingcell","SEARCH"` (idle). See format below.
`AT+QENG="neighbourcell"`	Neighbour cells	`OK` (none when not camped).

`AT+QENG="servingcell"` LTE format

```
+QENG: "servingcell",<state>,"LTE",<is_tdd>,<MCC>,<MNC>,<cellID>,
<PCI>,
       <EARFCN>,<band>,<UL_BW>,<DL_BW>,<TAC>,<RSRP>,<RSRQ>,<RSSI>,
<SINR>,<srxlev>
```

- `RSRP` dBm, `RSRQ` dB, `RSSI` dBm (all signed integers).

- `SINR` is a **converted** value: actual dB = `(1/5)*SINR - 20` (range 0..250 → -20..+30 dB).
- `cellID`/`PCI`/`EARFCN` are usually hex/decimal per field; `band` is the LTE band indicator.

`AT+QENG="neighbourcell"` LTE format

```
+QENG: "neighbourcell intra","LTE",<EARFCN>,<PCI>,<RSRQ>,<RSRP>,
<RSSI>,<SINR>,<srxlev>,...
+QENG: "neighbourcell inter","LTE",<EARFCN>,...
```

Registration / attach

Command	Purpose	Live response
`AT+CFUN?`	Radio functionality	`+CFUN: 1` (full). `0` = minimum/airplane.
`AT+CEREG?`	EPS registration	`+CEREG: 2,2` (2 = searching).
`AT+CEREG=2`	Enable registration URC **with location** (TAC/cellID)	`OK`
`AT+CGATT?` / `AT+CGATT=1` / `=0`	PS attach state / attach / detach	`+CGATT: 0`
`AT+COPS?`	Current operator selection	`+COPS: 0` (auto, not registered).
`AT+COPS=?`	**Network scan** (operator search, slow)	list of `(stat,"long","short","numeric",AcT)` tuples.
`AT+COPS=0`	Automatic operator selection	`OK`
`AT+COPS=1,2,"<mccmnc>"[,<AcT>]`	Manual select by numeric ID	`OK`

Cell / band / RAT locking

Command	Purpose	Live response
`AT+QNWLOCK=?`	Capabilities	`+QNWLOCK: "common/4g",<num of cells>,[[<freq>,<pci>],...]` and `+QNWLOCK: "common/lte"[,<action>[,&lt;EARFCN>,&lt;PCI>[,<status>]]]`
`AT+QNWLOCK="common/4g"`	Query cell lock	`+QNWLOCK: "common/4g",0` (0 cells = no lock); after lock `+QNWLOCK: "common/4g",1,1850,0`.
`AT+QNWLOCK="common/4g",<n>,&lt;EARFCN>,&lt;PCI>[,&lt;EARFCN>,&lt;PCI>...]`	**Lock** to n cells (works on this firmware)	`OK`
`AT+QNWLOCK="common/4g",0`	**Unlock**	`OK` (clears the lock).
`AT+QNWLOCK="common/lte"`	Query single-cell LTE lock (read only)	`+QNWLOCK: "common/lte",0,0,0,0`.

> **Firmware behavior (verified live on `EP06ELAR04A05M4G`):** the **`common/4g`** form is the one that accepts a lock *action* on this unit — `AT+QNWLOCK="common/4g",1,&lt;EARFCN>,&lt;PCI>` locks (returns `OK`, lock persists) and `AT+QNWLOCK="common/4g",0` clears it. The `common/lte` *set* forms (`...,1,&lt;EARFCN>,&lt;PCI>` and `...,0`) all return **`ERROR`** on this firmware — `common/lte` is query-only here. The UI therefore uses `common/4g` for both lock and unlock. | `AT+QCFG="band"` | Query band mask | `+QCFG: "band",0x8d0,0x1a0880800d5,0x0` = (GSM mask, LTE mask, TDS

> mask). | | `AT+QCFG="band",0,<lte_hex_mask>,0` | **Band lock** (LTE). `0x0` LTE mask = all bands. | `OK` | | `AT+QCFG="nwscanmode"` | RAT scan mode | `+QCFG: "nwscanmode",0` (0 = auto, 1 = GSM, 2 = WCDMA, 3 = LTE). | | `AT+QCFG="nwscanmode",<m>[,1]` | Set RAT preference (`,1` = effective immediately) | `OK` | | `AT+QCAINFO` | Carrier-aggregation info | `OK` (none aggregated when idle). |

LTE band mask: bit `(band-1)`, i.e. band N → `1 << (N-1)`. e.g. B3 = `0x4`, B7 = `0x40`, B28 = `0x8000000`. `0x0` (or all-ones) = all bands enabled.

> **Firmware note:** `AT+QCFG="nwscanseq"` returns `ERROR` on this EP06 firmware (`EP06ELAR04A05M4G`) — it is **not** surfaced in the UI. `AT+QSCAN` is also not supported on this EP06 firmware (it is an RG/RM 5G-series command); the network scan uses `AT+COPS=?` plus `AT+QENG` instead.

Voice calls (VoLTE / IMS)

EP06 voice is **VoLTE-only** — it needs IMS registered over LTE. `AT+QCFG="ims"` on this unit returns `+QCFG: "ims",1,0` (IMS enabled by config, current status 0).

Command	Purpose
`AT+QCFG="ims"`	Query/enable IMS. `AT+QCFG="ims",1` to force-enable.
`ATD<number>;`	Dial a voice call (trailing `;` = voice, not data).
`ATA`	Answer an incoming call.
`ATH` / `AT+CHUP`	Hang up.
`AT+CLCC`	List current calls: `+CLCC: <id>,<dir>,<state>,<mode>,<mpty>,"<number>",<type>`. Empty `OK` = no calls.
`AT+CLIP=1`	Enable calling-line ID URC (`+CLIP:`) on `RING`.

`AT+CLCC` call states: 0 active, 1 held, 2 dialing, 3 alerting, 4 incoming, 5 waiting.

SMS (text mode)

Command	Purpose	Live response
`AT+CMGF=1`	Text mode (vs PDU)	`OK`
`AT+CPMS?`	Storage areas / counts	`+CPMS: "ME",0,255,"ME",0,255,"ME",0,255`
`AT+CMGL="ALL"`	List messages (text mode)	per message: `+CMGL: <index>,"<stat>","<sender>",,"<timestamp>"` then body line.
`AT+CMGR=<index>`	Read one message	`+CMGR: "<stat>","<sender>",,"<timestamp>"` + body.
`AT+CMGS="<number>"` then body + Ctrl-Z	Send (handled by agent `send_sms`).	`+CMGS: <mr>`
`AT+CMGD=<index>[,<delflag>]`	Delete message (`delflag` 4 = all).	`OK`
`AT+CNMI?` / `AT+CNMI=2,1,0,0,0`	New-message indication config	`+CNMI: 2,1,0,0,0`

PDP / data

Command	Purpose	Live response
`AT+CGDCONT?`	List PDP contexts (APNs)	`+CGDCONT: 1,"IPV4V6","ims",...`
`AT+CGDCONT=<cid>,"<type>","<apn>"`	Set context. types: IP, IPV6, IPV4V6, Non-IP.	`OK`
`AT+CGACT=1,<cid>` / `=0,<cid>`	Activate / deactivate context.	`OK`
`AT+CGPADDR[=<cid>]`	Show assigned IP.	`+CGPADDR: 1,"<ip>"`
`AT+CGCONTRDP=<cid>`	DNS / gateway / PCO.	`+CGCONTRDP: ...`
`AT+QPING=1,"<host>"`	Ping from the modem (URC replies; needs `min_read`).	`+QPING: ...`

Event logging (URCs) — and the relay limitation

The agent relay is **request/response with a per-call port open** — it opens the serial port, sends one command, reads until OK/ERROR (or `min_read` elapses), then closes. It does **not** hold the port open to stream unsolicited result codes (URCs) back to the browser. So a true live event feed is not possible through this path without agent changes.

URCs worth enabling (each is a one-shot `OK` command):

Command	Enables URC
`AT+CEREG=2`	`+CEREG:` registration changes (with TAC/cellID).
`AT+CGEREP=2,1`	`+CGEV:` PS/bearer events.
`AT+CLIP=1`	`+CLIP:` caller ID on incoming `RING`.
`AT+CNMI=2,1,0,0,0`	`+CMTI:` new SMS indication.
`AT+QINDCFG="all",1`	`+QIND:` Quectel indications.

Pragmatic approach used in the UI: the Events tab enables the URCs above, then **polls** the modem on an interval (`AT+CEREG?`, `AT+CLCC`, `AT+QENG`, `AT+CSQ`, `AT+CMGL`) and renders state changes into a timestamped log. Each poll also uses a short `min_read` window so any URC that happens to arrive during the read is captured opportunistically. This catches state transitions (registration gained/lost, call started/ended, new SMS, signal change) at the polling cadence, which is sufficient for lab testing. The limitation is documented in the UI.

MSC — Mobile Switching Centre

The **MSC** is the call-control heart of the circuit-switched core. It holds the visitor location register (VLR) for the subscribers currently in its service area, sets up and tears down **voice calls**, carries **SMS**, and manages **mobility** — location updates, paging and handovers — for those subscribers. It reaches the radio side over **RAN/BSC** connections (A-interface, driven over M3UA/SS7), bridges voice to the IMS and interconnect through **SIP peers**, and steers the actual media through one or more **media gateways**. OmniWeb surfaces the MSC's live subscribers, calls, SMS transactions, RAN and SIP peers, routing table and pending pages; longer-run trends are on the Grafana dashboards reached from the same page.

← Operations Guide

The MSC is a per-instance element page reached from the sidebar when one runs in the site's inventory, and — where several are deployed — a fleet view for aggregate status. All traffic goes through the OmniWeb backend proxy to the MSC's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

A single-screen health read: the **API status** of the MSC and a summary of what it is currently handling — registered **subscribers** in the VLR, **active calls**, in-flight **SMS** transactions, live **RAN connections** and the state of the **SIP peers** and **STP** signalling links. It is the quickest way to confirm the MSC is up and see, at a glance, its live load.

Subscribers

Every subscriber the VLR currently holds, each row keyed by **IMSI** and **MSISDN** with its registration and location state. Opening a subscriber shows its full VLR context. From here an operator can:

- **Purge** a subscriber from the VLR — clear a stale or stuck registration so the subscriber re-registers cleanly.
- Run a **subscriber action** against a specific subscriber.
- Manage **supplementary services** — inspect and change the call-forwarding, barring and related services active for the subscriber.
- Send **Advice of Charge (AoC)** to the subscriber.
- Send a **silent SMS (Type 0)** or initiate a **silent call** — a location/reachability probe that does not alert the handset.

Calls

Every active call, each row showing the **calling** and **called** parties, the call **state** and its **duration**. Opening a call shows its full detail — the legs, the serving RAN connection and the media path. An operator can **force-release** a call to clear a stuck or unwanted call.

SMS

The SMS transactions the MSC is currently handling, each showing the **originator**, **recipient** and **delivery state**, so you can confirm messages are being carried and spot ones that are stuck.

RAN

The radio side of the MSC. It lists the active **RAN connections** — the live A-interface associations carrying subscriber signalling — and the known **BSCs**, learned from the RESET exchanges on the SS7/M3UA links. This is where you confirm the base-station controllers are connected and see which subscribers are on which connection.

SIP Peers

The SIP peers the MSC bridges voice through — the IMS, softswitches and interconnect trunks — each row showing the peer's **address** and its reachability **status**. Opening a peer shows its detail. An operator can **trigger an OPTIONS ping** on a peer to actively test reachability, which is the quickest way to prove a SIP path is up when calls to it are failing.

Media Gateways

The media gateways the MSC steers voice bearers through, each row showing the gateway and its **status**, so you can confirm the media plane is available for call setup.

Routing

The MSC's number-routing table — the rules that decide, by dialled-number prefix, where a call is sent. Each rule shows its **prefix** and **destination**. An operator can:

- **Add a route** — a new prefix and its destination.
- **Remove a route** by prefix.
- **Test a route lookup** for a number — enter a number and see which route it resolves to, to confirm dialling plans before or after a change.

Paging

The pending paging requests — the subscribers the MSC is currently trying to locate over the radio network for an incoming call or SMS. An operator can also **page a subscriber** directly from here.

Logs

The **Logs** tab tails the MSC's live log output for troubleshooting a single switch without leaving OmniWeb.

Monitoring dashboards

Call rates, SMS volumes, subscriber counts and other time-series for the MSC are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- The **HSS** holds the subscriber data the VLR resolves against and where the MSC registers as the serving node.
- The **CSCF / IMS** is bridged over the MSC's **SIP peers** for voice.
- The **BSCs** connect on the radio side over the A-interface, carried on the SS7/M3UA signalling links.

Network Testing (Load Test)

Network Testing drives synthetic signalling load at a live core so you can measure how it behaves under stress. It is not one generator but a *fleet* of them: standalone load generators register themselves with OmniWeb and appear here, and you drive them — set a rate, ramp up gradually, pause, drain, stop — from one console, per generator or across the whole pool at once.

← Operations Guide

Network Testing covers three kinds of load, each on its own sub-tab under **Network Testing** in the sidebar:

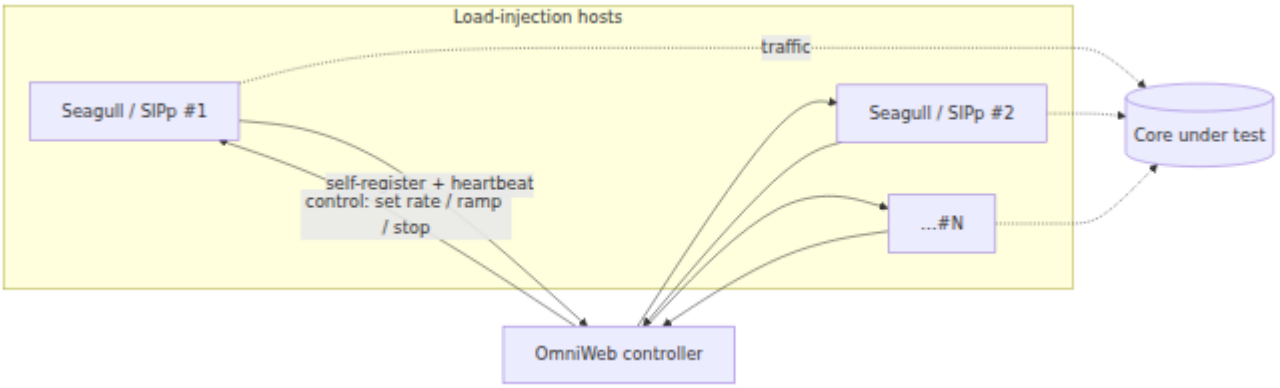
Sub-tab	Generator	Stresses	Measured as
Diameter Load Test	Seagull	The DRA + HSS (Diameter signalling)	Requests per second (rps)
SIP Load Test	SIPp	The CSCFs + HSS — IMS registrations, with IPsec	New registrations/sec, held (registered) UEs
RAN Load Test / **RAN Simulator**	RAN Simulator	The full radio→core path (S1AP/NAS, GTP-U, IMS)	See RAN Simulator

The Diameter and SIP tabs are the self-registering generator console described on this page. The **RAN** tabs are the RAN Simulator — virtual eNodeBs and UEs — documented separately.

How generators appear

You do not launch generators from OmniWeb. Each load generator (Seagull for Diameter, SIPp for SIP) is started on a load-injection host and pointed at OmniWeb; from then on it **self-registers**, heartbeating to the controller every couple of seconds with "I exist, here is my control address, and here are my live counters." Within a couple of seconds it shows up in the matching sub-tab.

If a tab is empty, it tells you exactly how to start one — copy the flag it shows (it derives the URL from the address you are browsing, so it is always correct for this deployment) and start the generator with it.



The registry is live telemetry only — it is not persisted. A generator that stops heartbeating past its TTL is shown **offline** (and, being offline, not running); it reappears the moment it heartbeats again.

Reading a generator

Each generator is a row in the left list (green dot = online, `RUN`/`IDLE`/`OFF`). Select one to see its detail:

- A **status line** — running (with the active scenario) or stopped/offline, the Diameter link state (up, and the peer) for Seagull, or the count of active IPsec SAs for SIPp.
- **Live metrics**, tailored to the tool:
  - *Diameter:* offered rate, successful, failed (timeout), average response time, current open calls.
  - *SIP:* offered rate, **held** (UEs currently registered), registered OK (200s), IPsec SAs, created/attempted, peak concurrent, failed, IPsec failed.

- **Peer CPU gauges** — when the selected site has a Monitoring (Prometheus) role, OmniWeb reads per-node CPU straight from Prometheus and shows a gauge per DRA/HSS/CSCF the traffic hits, plus (Diameter) DRA request/answer rates and the DRA→peer response delay. This lets you watch the effect of the load on the core, not just the generator's own numbers.
- **Scenario messages** (SIPp) — a per-step table of the REGISTER flow: how many of each message were sent/received, and any timeouts, unexpected messages, or retransmissions, so you can see exactly where a flow is failing under load.

MAXED

A SIPp process that is holding its full "Max UEs" cap can open no new calls, so it stops pushing traffic even though you have asked for more. OmniWeb flags this as **MAXED** (on the row, the detail header, and the Held metric). The fix is to raise the Max UEs cap — see below.

Driving traffic

For the selected generator (**Traffic control** card):

- **Scenario** — switch the running scenario live (Seagull), or see the scenario fixed at launch (SIPp).
- **Rate** — set the offered rate with a slider, presets, or a number box (rps for Diameter, new registrations/sec for SIP). `0` = idle/hold.
- **Ramp** (Seagull) — instead of a step change, climb to the target rate over N seconds.
- **Max UEs** (SIPp) — the ceiling on concurrent held (registered) UEs, applied **live with no restart**. Raise it to push a MAXED process harder.
- **Pause / Resume / Drain** (SIPp) — hold and release traffic, or drain (let held calls end without opening new ones).
- **Inject call** (SIPp) — fire a single call, for spot checks.
- **Reset stats** — clear the cumulative counters and the per-message table.
- **Stop traffic** — stop the generator pushing load.

Commands are sent from OmniWeb to the generator's control endpoint (a UDP socket for SIPp, a control port for Seagull); the buttons that appear depend on what the tool supports.

Driving the whole pool at once

When more than one generator is registered, a **Pool control** card appears above the list. It fans the same actions out across every online generator in parallel, so you drive "all 30" from one place:

- **Scope** — all generators, or one host group (generators supervised on the same host, e.g. `sipp-bsc01-01/-02/-03`, collapse into one group).
- **Fleet totals** — offered rate, held/successful, created, IPsec SAs, peak, failed, and fail-rate **summed across the scope**, so you see the aggregate load the pool is offering, not per-process figures.
- **Set a steady rate** — apply one per-process rate to every generator (the card shows the resulting pool total: N instances × rate).
- **Max UEs** (SIPp) — set the held cap on every process at once; the card flags how many are currently MAXED.
- **Ramp up gradually** — climb every instance from 0 to a target over a duration in steps. The ramp runs **in your browser** — leaving the page stops it.
- **Bulk lifecycle** — Pause / Resume / Drain / Stop all, and Reset stats.

Typical use

1. Start one or more Seagull (Diameter) or SIPp (SIP) generators on your load hosts, pointed at OmniWeb; confirm they appear online.
2. Pick a modest steady rate on one generator and confirm the core answers (watch Successful / Registered OK and the peer CPU gauges).
3. Ramp the pool up gradually toward the target and watch the fleet totals, the fail rate, and the peer CPU/response-delay gauges for the point where the core starts to degrade.
4. For SIP soak tests, set a Max UEs cap and let the fleet hold a large registered population; raise the cap if processes go MAXED.

5. Drain/Stop when done.

Related

- RAN Simulator — the RAN sub-tabs: virtual eNodeBs/UEs for functional and load testing across the whole radio→core path.
- Grafana Dashboards & Statistics — the peer CPU gauges here read the same Prometheus the dashboards do; use Grafana for the full picture during a test.

OmniRoam — Roaming Agreements & Email Automation

OmniRoam's **agreements** side turns the paperwork of setting up a roaming partner — the emails, the AA.12/AA.13 forms, the IOT rates, the SIM swaps, and the provisioning work — into a single tracked record per partner, and automates the parts that used to be copy-and-paste. Its headline trick: **forward a partner email to OmniRoam and it reads the thread, plucks out the facts, and updates the agreement for you** — then generates and e-signs the contracts, files them, and raises the work orders to get the partner launched.

← Back to Operations Guide

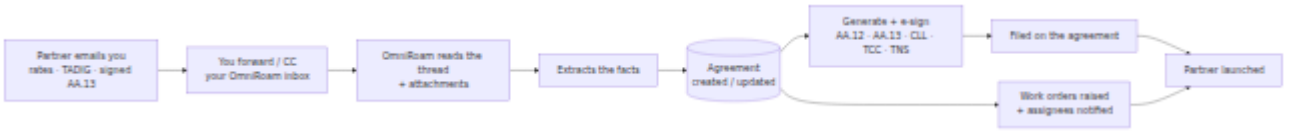
> **Two different "OmniRoam"s.** This document is about the **agreements / email / document** side of OmniRoam (the commercial and provisioning lifecycle with a roaming partner). It is **not** the same as the **roaming test books** (IR.38 / IR.48 / VoLTE conformance testing) — those are covered separately in OmniRoam — Roaming Test Automation. One product, two jobs: *get the agreement signed* (here) and *prove the roaming works* (there).

Table of Contents

The Big Picture

Everything in OmniRoam hangs off one record: the **Agreement** — one per roaming partner. An agreement carries the partner's name, country, TADIG(s)/PLMN, its lifecycle **status**, its contacts, the agreed commercial **rates**, the SIMs exchanged, the generated and received **documents**, an activity **timeline**, and the **work orders** that get the partner live.

You can build all of that by hand in the UI. But the point of OmniRoam is that most of it arrives *by email*, and OmniRoam files it for you.



The tenant you're working in is an **Operator** (your own network — the "home" side). Everything is scoped to it: your letterhead and signatory fill the generated documents, your inbound address drives the email automation, and your teams are the work-order assignees.

---

1. Inbound Email Ingestion — "plucking the data out"

What you do

Negotiating with a partner is an email conversation. To bring it into OmniRoam you do one thing: **forward the email — or CC the OmniRoam inbound address on the thread**. That's it. Your dedicated inbound address (for example `roaming@yourdomain`) is configured once per operator (Settings → paperwork details, `Inbound email`).

- **Forward a whole thread** once and OmniRoam builds the agreement from the history buried in the quoted messages.

- **CC the address on an ongoing negotiation** and every new email keeps the record current — status advances, contacts and TADIGs fill in, rates land, and attached paperwork gets filed, email by email.

You never fill in a form to do this. The email *is* the input.

What OmniRoam does

Lexical error on line 7. Unrecognized text. ...oaming/inbound-email] HOOK --> EP ----------^

Try again

Step by step, once a mail reaches OmniRoam:

1. **It figures out which operator (tenant) the mail is for** by matching a To/Cc address to an operator's configured inbound address. If nothing matches, the email is logged as `no-operator` and *nothing is changed* — a safe no-op.
2. **It reads the attachments, not just the body.** The agreed rates or the signed terms are often inside an attached **AA.13**, an **IOT rate sheet**, or a spreadsheet — so OmniRoam extracts the text from PDF, Word, Excel and CSV attachments and feeds that to the extractor too. Signature-block logos (PNG/JPG) are ignored.
3. **It extracts the facts** using the ChatGPT API. The extractor is specifically told the outer sender is usually just *forwarding* the thread (often your own roaming manager) and to read the **whole quoted history** — attributing rates, contacts, addresses and SIMs to whoever actually stated them, never to the forwarder. It returns a structured record: partner name, country, address, TADIG(s)/PLMN, best-fit lifecycle status, agreed rates (only if *both* sides explicitly agreed), partner contacts, exchanged/requested SIMs, whether a signed agreement is attached, and a one-line summary.
4. **It finds the right agreement — or creates one.** Matching is deliberately forgiving so a thread never spawns a duplicate: exact partner name, then a "normalised name" match (so *AT&T Mobility* joins *AT&T*), then a **shared PLMN** (MCC+MNC) match — a very high-precision signal since a PLMN uniquely identifies a network. No match → a brand-new agreement.

5. **It merges in only what's new.** For both new and existing agreements, OmniRoam applies whatever *this* email newly reveals and **never clobbers** existing values:
    - **Status** only ever moves *forward* along the lifecycle (never backwards).
    - **Country / address** set if blank or changed.
    - **TADIGs** added — but only codes that exist in the IR.21 catalog (so the AI can't invent one), and never your own home network's codes.
    - **Contacts** added, or existing ones *enriched* (blank fields filled in).
    - **SIMs** recorded (test SIMs the partner shared) and **SIM requests** (batches someone asked to be shipped to an address).
    - **Agreed rates** become a rate tier — only when both sides explicitly agreed, and never a duplicate of an existing tier.
    - A **signed AA.12/AA.13** (attached, or stated in the mail) advances the status to *IRA Signed*.
6. **It writes two records.** A friendly **activity-log entry** on the agreement timeline ("Email from *Jane at Partner* (forwarded): agreed data rate — status → rates_agreed, added contact Jane, recorded agreed rates"), and a raw **inbound audit row** (see below).

Processing is **idempotent** — the same email (by Message-ID) is only ever processed once, so CC'ing multiple people or re-forwarding a thread never double-counts.

Three ways the email gets in

You only need one. All three end up calling the same processing logic:

Path	How it works	When to use it
Cloudflare Email Worker	Cloudflare Email Routing forwards the raw message to a small Worker that streams the `.eml` straight to the OmniRoam webhook.	Domains already using Cloudflare; no mailbox to poll.
IMAP poller	OmniRoam logs into a real mailbox every ~60 s, processes unread messages, and flags them read.	Any standard mailbox (implicit TLS, STARTTLS, or plain).
Provider inbound-parse	A mail provider (Mailjet/SendGrid) POSTs the parsed message + attachments to the webhook.	If you already run inbound-parse.

The webhook (`/api/roaming/inbound-email`) is guarded by a **shared secret**, so only your configured mail path can post to it.

The Inbound Email Log — your audit trail

Every processed email — success *or* failure — writes a row to the **Inbound Email Log** (in the UI under the operator's Emails view). Each row shows the sender, recipients, subject, attachments, whether the **AI extraction succeeded**, exactly what it extracted, and what the email **changed** (or why it did nothing — `no-operator`, `no-partner`). This is your window into the automation: if a rate didn't get picked up, the log tells you what the extractor actually saw.

The Inbound Email Log — one row per processed email, with the extraction outcome and the changes it made. This is the audit trail for the whole email→agreement automation.

---

2. Agreements & Partners

The **Agreements** list is your book of roaming partners for the current operator. Each row is one partner, with its lifecycle **status**, TADIG/country, priority, and who the next action is waiting on ("Us" or "Them").

The Agreements list. Statuses run left-to-right along the lifecycle; the filter bar narrows by status. Agreements are created here by hand — or automatically from an inbound email.

The lifecycle (status)

An agreement moves forward through these stages. The email automation only ever advances the status, so the record reflects the furthest point the conversation has reached:

`Enquired → IOT Negotiation → IOT Pending Signature → IRA Negotiating → Awaiting IRA Signatures → IRA Signed → Rates Agreed → SIM Exchange → Testing → Launched` (plus `Default Launch`, `Suspended`, `Terminated`).

Opening a partner

Click a partner to open its **detail** page — a set of tabs/sections covering everything about the relationship:

- **Overview & timeline** — status, priority, notes, and the activity log (every email, generated document, signature, and manual note in date order).

- **TADIGs** — the partner's network codes (with MCC/MNC, connection type and direction). A **"find candidates"** action can pull the partner operator's other TADIGs from the IR.21 catalog for you to review and add.
- **Contacts** — roaming/commercial/technical people at the partner.
- **Rates** — the agreed commercial rate tiers (data/SMS/voice, currency, unit).
- **SIMs & SIM requests** — test SIMs exchanged, and batches requested for shipping.
- **Documents** — generated and received paperwork (see below).
- **Routing** — DRA routes, SS7 global titles, DNS and IP ranges for provisioning (much of it pre-populated from the partner's IR.21).

Everything on the detail page is **editable** — the automation gives you a head start, and you correct or add by hand as needed.

The IR.21 catalog

TADIG lookups draw on a **global IR.21 world-operator catalog** keyed by TADIG (organisation, network, MCC/MNC, DNS, IP ranges, SS7 nodes, contacts). This is how OmniRoam validates a TADIG the AI extracted, discovers a partner's sibling TADIGs, and pre-populates routing data.

---

3. Automatic Document Generation & Signing

Generating a document

On an agreement's **Documents** section you pick a document kind and click **Generate**. OmniRoam renders the real Word form, filled with both sides' details, ready to sign:

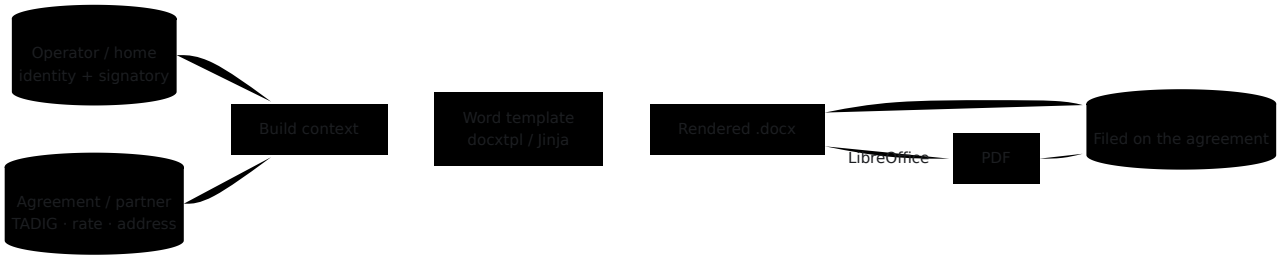
Kind	What it is	Format
AA.12	GSMA AA.12 IREG roaming agreement (the real form, both sides pre-filled)	.docx / PDF
AA.13	GSMA AA.13 common annexes, including the billing/settlement annex — see AA.13 settlement options	.docx / PDF
CLL	Commercial Launch Letter (generic)	.docx / PDF
CLL (Bilateral / Unilateral, after testing)	Launch letter confirming the start date **after successful IREG & TADIG testing**	.docx / PDF
CLL (Bilateral / Unilateral, default)	Launch letter for a launch **by default, without completing IREG/TADIG testing**	.docx / PDF
TCC (VoLTE)	VoLTE Test Call Completion	.docx / PDF
TADIG completion	TADIG completion form	.docx / PDF
TNS Roaming Agreement Form	TNS clearing-house request (delivered as legacy .doc)	.doc
TNS 3G-4G Request	TNS clearing-house request	.docx

How the fields get filled. Each document is a Word template with placeholders (docxtpl / Jinja). OmniRoam fills them from:

- **The home side** — from your **Operator** record: your name, legal name, TADIG, PLMN, country, **signatory & title**, place, and address. Because this

comes from the tenant, each hosted operator gets its *own* letterhead and signatory automatically.

- **The partner side** — from the **Agreement**: partner name, its short-form abbreviation used in the contract body, TADIG(s), PLMN(s), country and address. The partner's own signatory/title default blank — that line is left ready for them to sign.
- **Derived values** — effective date, direction (bilateral/inbound/outbound), hub company, Diameter realms, currency, and the confirmed IOT rate.



Output format — Word by default, PDF for signing

Generation produces an **editable Word** `.docx` by default — that is the working copy you review and hand-edit if needed. You get a PDF only when you actually need a fixed, final rendering:

- Tick the **PDF output** checkbox before clicking Generate to render a PDF instead of the `.docx`.
- The **e-signature** flow always uses PDF (the signed artifact must not change).
- **Download / preview** a filed document *as PDF* on demand.

A **Roaming Pack** email attaches the AA.12/AA.13 as editable `.docx` so the partner can review and comment — it is not the signing step, so it is not sent as PDF.

AA.13 settlement options (Annex C.3.2)

GSMA AA.13 Annex C.3.2 (billing & settlement) is a **menu of choices** each partner negotiates — which invoice model, which payment method, the settlement threshold and period, payment terms, and interest. Rather than lock

one set of choices into the template, OmniRoam exposes them as a settlement-options panel on the **Documents** section: expand **AA.13 settlement options**, set the values for this partner, then click **Generate AA.13**. The generated annex contains only the options you chose — the alternatives (and the GSMA "delete those which do not apply" scaffolding) are dropped automatically.

Every field has a sensible default, so generating without touching the panel yields the standard terms; change only what differs for a given partner.

Option	Default	Description
Invoice type	Single-network invoice (SNI)	Selects the settlement annex: **Single-network (SNI)** → Annex C.3.2 H, or **Multi-network (MNI)** → Annex C.3.2 I. The unused annex is removed.
Payment method	Net settlement	The settlement mechanism: **Net settlement**, **Direct full payment (no conversion)**, **Direct full payment (with conversion)**, or **Netting without conversion**. Only the chosen method's clause is kept.
Threshold settlement	Option 1 — Netting	**Option 1 (Netting)** applies a single agreed low-balance threshold; **Option 2 (Direct payment)** uses per-party (Party A / Party B) thresholds.
Threshold	50	Low-balance settlement threshold amount, in the agreement's settlement currency (e.g. SDR). Below this monthly aggregate, settlement may be deferred.
Period start / **Period end**	January / December	The non-settlement period for balances below the threshold — typically the financial year.
Payment days	60	Days from the invoice date within which the debtor must pay.
Interest home %	8	Annual interest the creditor may charge on overdue balances when the **home** operator is the creditor.
Interest partner %	8	Annual interest when the **partner** operator is the creditor.

Option	Default	Description
Send zero-value invoices	Off	When off, the annex states each party **will not** send monthly invoices with a zero (0.00) amount.
BCE on TAP invoice	Off	When off, the annex states BCE usage **will not** be included on the TAP Roaming Invoice.

How it works. The choices are passed to generation as overrides on top of the built-in defaults; the AA.13 template resolves them into the finished annex — selecting the right option blocks, filling the amounts/periods/rates, and phrasing the toggles. The settlement currency itself comes from the agreement's configured currency, not this panel.

Use case. A partner insists on 90-day terms, a 100-SDR threshold and an April–March financial year: open the panel, set **Payment days** 90, **Threshold** 100, **Period start** April, **Period end** March, and generate. Every other clause stays at the standard terms.

> **The TNS clearing-house forms are special.** The two TNS requests are *not* docxtpl templates — they are the operator's genuine TNS forms with only the fillable fields changed and the right checkboxes ticked, so TNS receives exactly the layout it expects. They're **IOT-gated**: OmniRoam won't generate them until the agreement has a **confirmed (agreed) rate** — trying earlier returns a clear "a confirmed IOT is required" message. The checkbox states (agreement type, direction, services, LTE interfaces, GRX) are auto-derived from the agreement, and all fields are overridable at generation time.

Generation to PDF (and the legacy `.doc` for TNS) is done by headless LibreOffice on the server — you just get the finished file.

The Documents section of an agreement. Generate a contract on the left; generated and received documents (with signing status) list below, each downloadable.

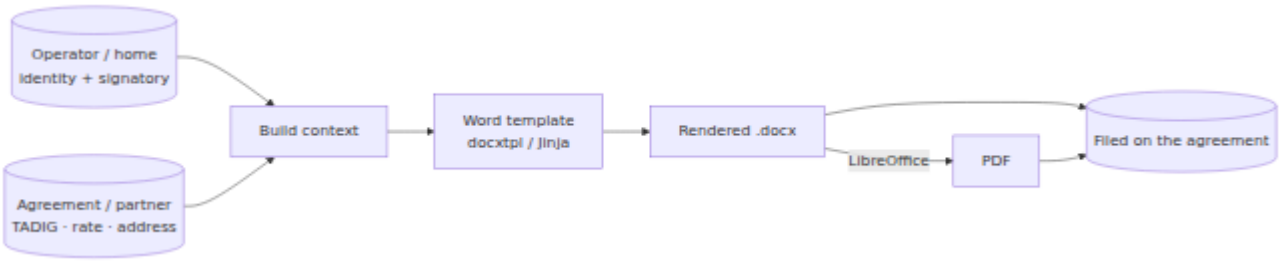
Signing — online or offline

OmniRoam supports two signing routes:

Online e-signature (partner signs in the browser, no login). From a generated AA.12/AA.13 you click **Send for signature**, name the signer and their email, and OmniRoam:

1. Creates a signature request with an opaque token and emails the partner a **secure link** (no account needed).
2. When they open it, OmniRoam marks the request *viewed* and logs it on the timeline. They review the document, optionally correct their signatory name/title/company, and **sign** — typing their name and/or drawing a signature.
3. OmniRoam produces the **signed PDF** and emails it to **both parties** (signer + your operator's return mailbox), files it on the agreement, and logs "Signed by … on …".

4. The partner can also **decline** (with a reason), which is recorded too.



Offline signing. Prefer to sign the old way? Download the generated document, sign it externally, and upload the signed copy back onto the agreement. A filename containing *signed*/*executed*/*countersigned* is recognised as a signed copy — and an inbound email that attaches one advances the status to *IRA Signed* automatically.

The Roaming Pack — one-click first contact

Instead of generating each form separately, **Send roaming pack** emails a partner contact the **AA.12 + AA.13 + IOT offer** together in one message, generating the contracts as PDFs on the fly and attaching your most recent IOT offer. Pick a contact (or type a name + email — it's backfilled as a contact), and OmniRoam sends the pack with your saved template wording. This is the "let's do a roaming deal" opening email, sent from inside OmniRoam.

Editable email templates

The three automated emails — **signature request**, **signed copy**, and **roaming pack** — use editable templates (per-operator override, else a global default), with placeholders like `{{ signer_name }}`, `{{ partner }}`, `{{ operator }}`, `{{ sign_url }}` and `{{ document_name }}`. Tune the wording once in Settings and every send uses it.

> **How email actually leaves OmniRoam.** All outbound mail goes via the **Mailjet HTTPS Send API**, not SMTP. Hosting platforms (Railway) block outbound SMTP ports, so OmniRoam POSTs to Mailjet over HTTPS instead. Sends are **best-effort and non-blocking**: the UI button returns immediately and the mail goes out on a background thread, so a slow mail server never stalls the app.

---

4. Filing & Storage

Every document — **generated** (AA.12, AA.13, CLL, TCC, TNS…) or **received** (a signed copy attached to an email, an uploaded IR.21, an IOT sheet) — is filed **directly on the agreement** it belongs to. There's no separate document store to manage:

- Documents are stored as blobs **in the database**, tagged with a **kind** (`aa12`, `aa13`, `aa12_signed`, `iot`, `ir21`, `cll`, `other`, …), filename, content type, size, and whether OmniRoam **generated** it.
- Signed copies are tagged distinctly (e.g. `aa12_signed`) so the signed set is always distinguishable from drafts.
- **Inbound attachments are de-duplicated** — an identical copy (same filename + byte length) that arrives again is not re-filed.
- Every generation, upload, signature, and IOT export writes a **timeline entry**, so the agreement's activity log is a complete paper trail of what was produced and received, and when.

To retrieve a document, open the agreement's **Documents** section and download it (generated Word/PDF, or the received original). Signed PDFs produced by the online signing flow are stored on the signature request and also emailed to both parties.

You can also **export the agreed rate as a GSMA RAEX IOT 5.0 XML** file from the agreement — the same rate selection the TNS forms use, so the exported IOT and the generated forms always agree.

---

5. Work Orders & Assignee Notifications

Getting a partner live is *work* — shipping SIMs, changing routing, running IREG/TADIG tests, audits, chasing issues. In OmniRoam that work is tracked as **Work Orders**: the single tracked unit of roaming work. A work order can hang off an agreement (a partner) or stand alone.

The Work Orders view. Each order has a type, priority, status, optional partner link, and an assignee — the party notified when it's assigned.

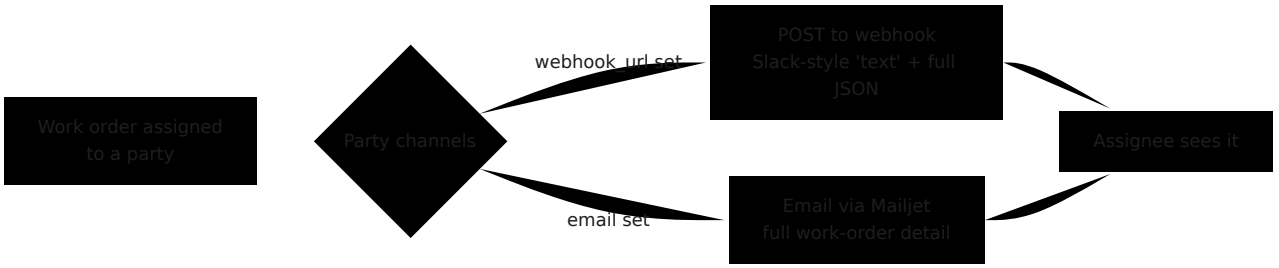
Creating and assigning

A work order has a **type** (SIM Dispatch, Routing, IREG Testing, TADIG Testing, Audit, Issue), a title, priority, status (Open / In Progress / Blocked / Done / Cancelled), an optional partner (agreement) link, free-text details, a due date, and type-specific structured fields (e.g. a SIM Dispatch carries quantity, postal address, carrier and tracking).

You assign a work order to a **party** — a named internal team or external contact (e.g. "Roaming NOC", "TADIG admin") managed per operator in **Settings**. You can set a **default assignee per work-order type**, so a new SIM Dispatch auto-assigns to whoever handles dispatch unless you override it.

The notification — webhook + email

When a work order is **assigned** (created with an assignee, or re-assigned to a new party), OmniRoam **notifies that assignee** through whatever channels the party has configured:



- **Webhook** — if the party has a webhook URL (e.g. a **Slack incoming webhook**), OmniRoam POSTs a message with a human-readable `text` block *and* the full work-order as structured JSON (type, priority, status, partner, the type-specific payload, a link back to the Work Orders page).
- **Email** — if the party has an email address, OmniRoam sends the same detail as an email (via Mailjet), subject `[Work Order] <title> assigned to you`.

A party with **neither** channel set is simply a silent assignee (no-op). Both notifications are best-effort and non-blocking — a slow webhook or mail server can never stall the app — and re-assignment only notifies when the assignee actually **changes** to a new party (so editing other fields doesn't re-ping people).

> **From the assignee's chair:** you get a Slack message (or email) the moment a job lands on your team, with everything you need — what to do, for which partner, the SIM count and shipping address (or the routing change, or the test scope), the priority and due date, and a link straight to the work order.

The Full Lifecycle, End to End

Putting it all together — the journey from "a partner emailed us" to "the partner is live":

Partner: 'interested in roaming'

you forward to OmniRoam inbox

Agreement created status: Enquired

You Send Roaming Pack AA.12 + AA.13 + IOT offer

Partner replies with agreed rates + signed AA.13 attached

CC'd on the thread

Rate tier recorded signed doc filed status → IRA Signed / Rates Agreed

Counter-sign online or upload offline copy

At every step the agreement's **timeline** records what happened, and the **inbound email log** keeps the raw evidence of what each email contributed.

What You Set Up Once

Most of the automation just needs a little one-time configuration per operator (in **Settings**):

Setting	Why	Where it's used
Paperwork details (legal name, TADIG, PLMN, country, **signatory & title**, place, address)	Fills the home side of every generated document	Document generation
Inbound email address	The address you forward/CC partner mail to	Email ingestion
Signed-return mailbox	Where offline-signed and signed-copy emails go	Signing
Email templates (signature request, signed copy, roaming pack)	Your wording for the automated emails	Outbound email
Work-order parties + per-type default assignees	Who gets notified for each kind of work	Work orders

With those in place, forwarding a partner email really does become: *email in → agreement out → contracts generated and signed → work orders raised and your team notified.*

See Also

- OmniRoam — Roaming Test Automation — the *other* OmniRoam: running the GSMA roaming test books (IR.38 / IR.48 / VoLTE) against a live network. Complementary to this: sign the agreement here, prove it works there.
- Common Operations — the viewing/editing patterns shared across all of OmniWeb.

OmniRoam — Roaming Test Automation

OmniRoam is the roaming test-book automation feature of OmniWeb. It runs the GSMA roaming compliance test books against a live network, drives the individual test cases automatically from a test device, records a pass/fail verdict with supporting evidence for each case, and produces a completed GSMA test-book workbook ready to hand to a roaming partner or hub.

In short: OmniRoam turns a manual, multi-hour roaming compliance exercise into a button press (or a scheduled job), and hands back the same official GSMA spreadsheet a tester would otherwise fill in by hand.

> **OmniRoam has two sides.** This document covers the **roaming test books** (IR.38 / IR.48 / VoLTE conformance testing — *does the roaming actually work?*). The other side is the **roaming agreements & email automation** — turning a partner email into a signed agreement, generated AA.12/AA.13/CLL/TCC/TNS paperwork, and the work orders to get the partner live. That is documented separately in **Roaming Agreements & Email Automation**. One product, two jobs: *get the agreement signed* (there) and *prove the roaming works* (here).

The Roaming Test Books screen — each row is a book run against a roaming partner, with its current status.

Quick Links

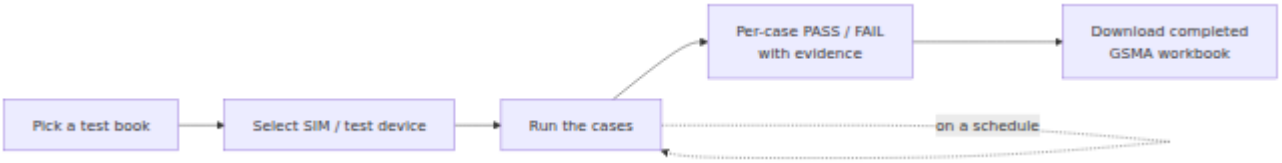
- **Test Books** — The available books (IR.38, IR.48, VoLTE) and every test case they contain
- **Running Tests** — Creating a book, running it, customising the case set, and scheduling routine runs
- **Test Book Explorer** — Inspecting what each book tests and how each case is driven
- **Completed Test-Book Documents** — How OmniRoam generates the filled GSMA workbook
- **Roaming Agreements & Email Automation** — The *other* side of OmniRoam: forward a partner email → agreement created, contracts generated and e-signed, filed, and work orders raised

What OmniRoam Does

A roaming test book is a checklist defined by the GSMA: a list of test cases that prove a subscriber from one network (the **Home PLMN**, "PLMN(a)") receives the expected services while roaming on another network (the **Visited PLMN**, "PLMN(b)"). Historically each case is performed by hand and the result written into a spreadsheet.

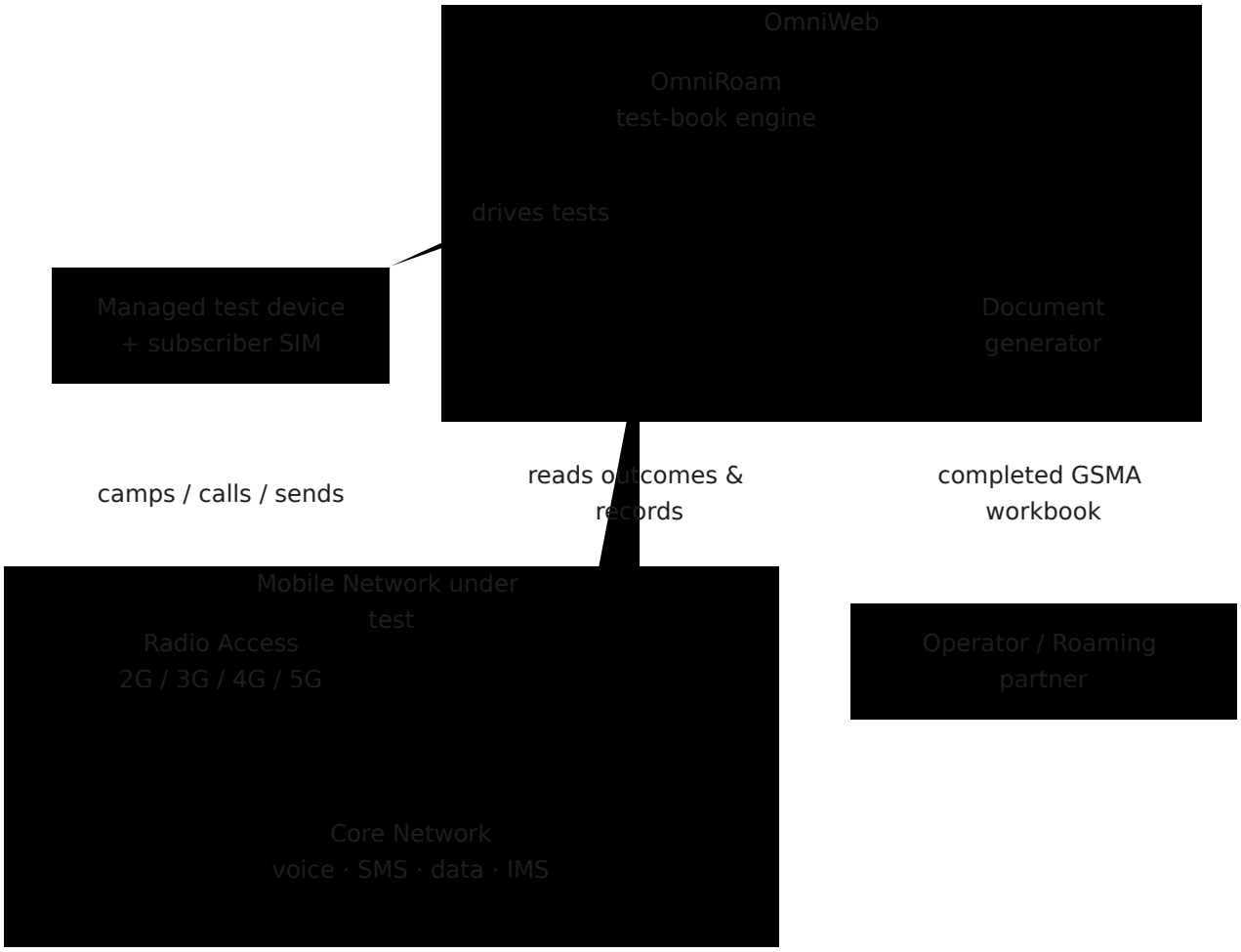
OmniRoam automates three things:

1. **Driving the test** — placing the call, sending the SMS, running the data session, or provisioning the supplementary service on a test device.
2. **Judging the result** — comparing what happened against the GSMA expected result and recording **PASS**, **FAIL** or **NOT PERFORMED**, with the supporting evidence (timings, throughput, network release causes, and so on).
3. **Producing the document** — filling the official GSMA workbook with the results and all the surrounding metadata (identities, dates, tester details).



Where It Fits

OmniRoam exercises a real subscriber on the real network using a managed test device. Each automated case drives the device, then reads the outcome back from the device and — where applicable — from the network's own records (for example, the serving gateway's charging records for a data session). Nothing is simulated: a call that is reported as connected actually connected.


OmniWeb
OmniRoam
test-book engine
drives tests
Managed test device
+ subscriber SIM
Document
generator
camps / calls / sends
reads outcomes &
records
completed GSMA
workbook
Mobile Network under
test
Radio Access
2G / 3G / 4G / 5G
Core Network
voice · SMS · data · IMS
Operator / Roaming
partner

Key Concepts

Term	Meaning
Test book	A GSMA-defined set of test cases (e.g. IR.38, IR.48, VoLTE). See Test Books.
Test case	A single check within a book, identified by its GSMA number (e.g. IR.48 case `2.3`, "Barring of All Outgoing Calls").
Run	One execution of a book (or a chosen subset of its cases) against a device and SIM.
Verdict	The recorded outcome of a case: `PASS`, `FAIL`, or `NOT PERFORMED`.
Evidence	The supporting detail captured for a verdict — timings, throughput figures, network causes, interrogation read-backs, etc.
Home PLMN (a)	The network that owns the subscriber being tested.
Visited PLMN (b)	The network the subscriber is roaming on.

Automated vs. Manual Cases

Most cases in each book are **auto-driven**: OmniRoam performs them end-to-end and records the verdict without operator interaction. A few cases depend on network conditions that cannot be induced from a test device alone (for example, the optional CAMEL prepaid negative tests, or a multimedia-messaging delivery) and remain **manual** — OmniRoam still lists them in the

book and the operator records the result by hand, so the final document is complete.

The Test Book Explorer shows exactly which cases in a book are automated and how each one is driven.

Standards Referenced

OmniRoam follows the GSMA Permanent Reference Documents and the underlying 3GPP specifications:

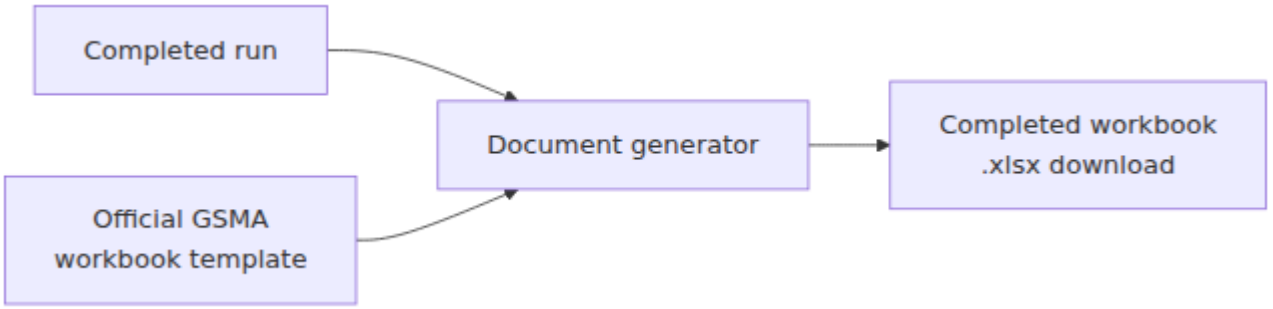
Reference	Applies to
GSMA IR.38	LTE roaming data test book
GSMA IR.48 / IR.24	Bilateral roaming (voice / SMS / MMS / data) test book
GSMA BA.65	Roaming VoLTE testing
GSMA IR.92	IMS profile for voice and SMS (VoLTE behaviour)
GSMA IR.88	LTE roaming guidelines (attach reject causes)
3GPP TS 22.011	Service accessibility (operator-determined barring)
3GPP TS 23.011	Supplementary service status (Active/Registered/Provisioned/Quiescent)
3GPP TS 22.082	Call Forwarding supplementary services
3GPP TS 22.088	Call Barring supplementary services

Completed Test-Book Documents

The end product of an OmniRoam run is a **completed GSMA test-book workbook** — the official GSMA spreadsheet for the book, filled with the run's results and all the surrounding detail, ready to send to a roaming partner or hub. It is produced with a single **Download Test Book** action and requires no manual transcription.

What You Get

OmniRoam takes the official GSMA template for the book and fills it in place, so the result is the same workbook a partner expects to receive — same sheets, same case numbering, same layout — but populated with real results instead of blank fields.



What Gets Filled In

Three categories of information are written into the workbook.

1. Per-case results

For every case in the book, the workbook's result cell is set to the recorded verdict — **PASS**, **FAIL**, or **NOT PERFORMED** — and the case's comments cell is filled with the supporting evidence captured during the run.

In the document	Comes from
Test case result (PASS / FAIL / NOT PERFORMED)	The case verdict
Test case comments	The evidence: timings, throughput, network causes, interrogation read-backs, etc.

2. Identities and metadata

The header and operator-information sections are filled from the run:

In the document	Comes from
IMSI	The test SIM
MSISDN	The test SIM
Home PLMN (a) / Visited PLMN (b)	The roaming partner / PLMNs chosen for the book
Date of tests	The run date
Tester name, phone, email	The tester details on the book
APN (where applicable)	The data configuration used

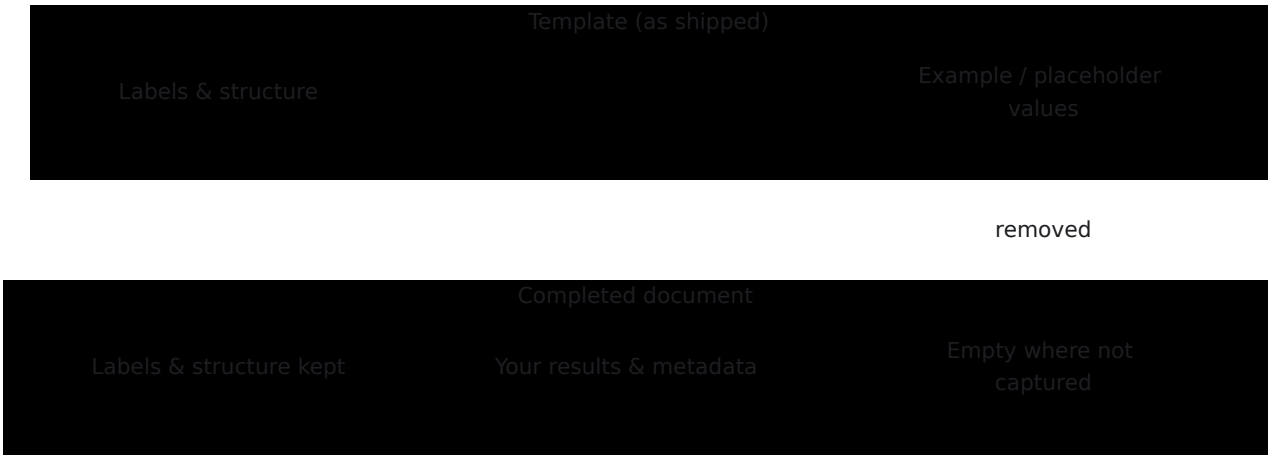
3. Measured values

For the data and supplementary-service cases, the specific measured values the GSMA sheet asks for are written into their dedicated cells:

In the document	Comes from
Data session start / end times	The data session
Data volumes sent / received	The serving gateway's charging records
Session duration	The data session
Throughput / mean data rate	The throughput measurement
Service success indications (Yes/No)	The case outcome

Clean, Presentable Output

The blank GSMA templates ship with example and placeholder content in the fill-in cells — sample timings and reference notes intended to guide a manual tester. OmniRoam removes that placeholder content when generating the document, so the completed workbook shows only your real results and empty fields where a value was not captured. It never carries the vendor's sample data into your delivered report, and it never overwrites the GSMA labels, headings, or structure.



When to Generate

The document can be downloaded at any point — it always reflects the current state of the run. A typical flow is to run the book (or scheduled routine), review the per-case verdicts, re-run any cases that need attention, and then download the finished workbook once the results look right. Because the export reflects the live run, regenerating after a re-run simply produces an updated document.

Manual Cases in the Document

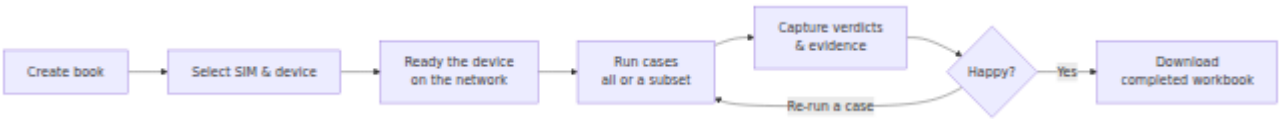
Cases that OmniRoam cannot drive automatically still appear in the workbook with their case number and title, ready for the tester to record a result by hand. This keeps the delivered document complete: the automated cases are filled with real evidence, and the manual cases are presented in the right place for the tester to finish.

See Running Tests for how verdicts are produced and Test Book Explorer for the exact cell-by-cell mapping of what gets filled.

Running OmniRoam Tests

This guide covers the full lifecycle of a roaming test: creating a book, running it, watching the cases execute, customising which cases run, re-running a single case, and scheduling routine unattended runs.

The Test Lifecycle



Creating a Test Book

The Roaming Test Books screen lists every book with its type, the home and visited networks, the subscriber identity, the tester, and the current status.

A test book is created from the **Roaming Test Books** screen. When creating a book you choose:

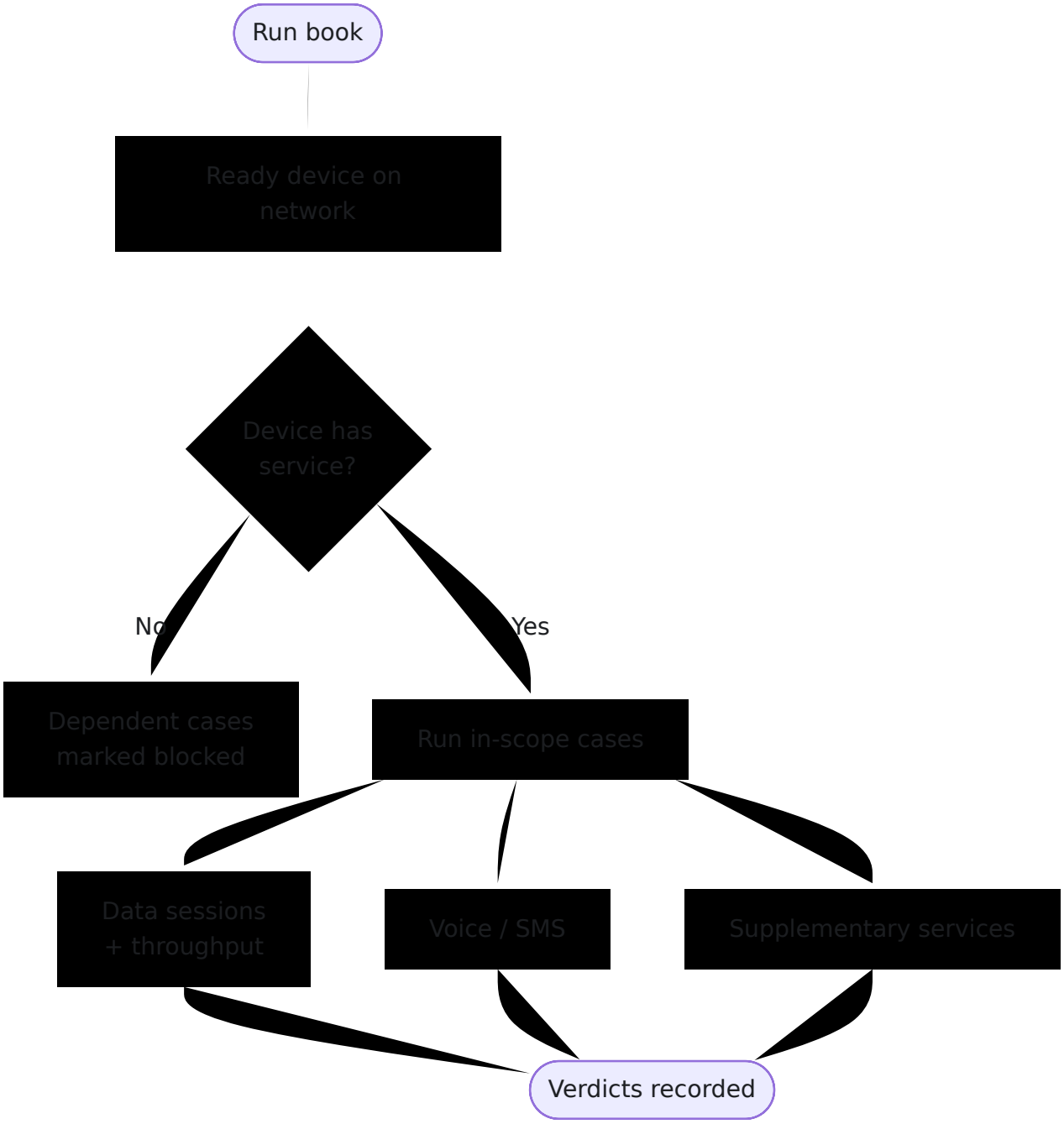
Field	Purpose
Book type	Which GSMA book to run — IR.38, IR.48, or VoLTE. This selects the case set.
Roaming partner / PLMNs	The home (PLMN a) and visited (PLMN b) networks. Used to populate the document header and to pick the correct throughput threshold.
SIM	The subscriber SIM to test with. Its IMSI and MSISDN are read and used throughout the run and in the final document.
Tester details	Name, phone, and email of the person responsible — written into the completed workbook.

Once created, the book lists every case from the chosen GSMA book, each starting in a "not yet performed" state.

Running a Book

Running a book performs each in-scope case in turn. Before the test cases run, OmniRoam readies the test device on the network — presenting the SIM and confirming the device has attached and has service. This is a **prerequisite gate**: if the device cannot get service, the dependent cases are reported as blocked rather than failed, because there is no point running a data or voice test with no bearer.

As part of readying the device, OmniRoam **reads the subscriber identity off the modem and fills the book automatically**: the **IMSI** is read from the SIM, the **home network (HPLMN)** is derived from it, and the **visited network (VPLMN)** is read from the network the device is currently camped on. These populate the book's header — and therefore the completed document — with what was actually tested, rather than relying on values typed in by hand.



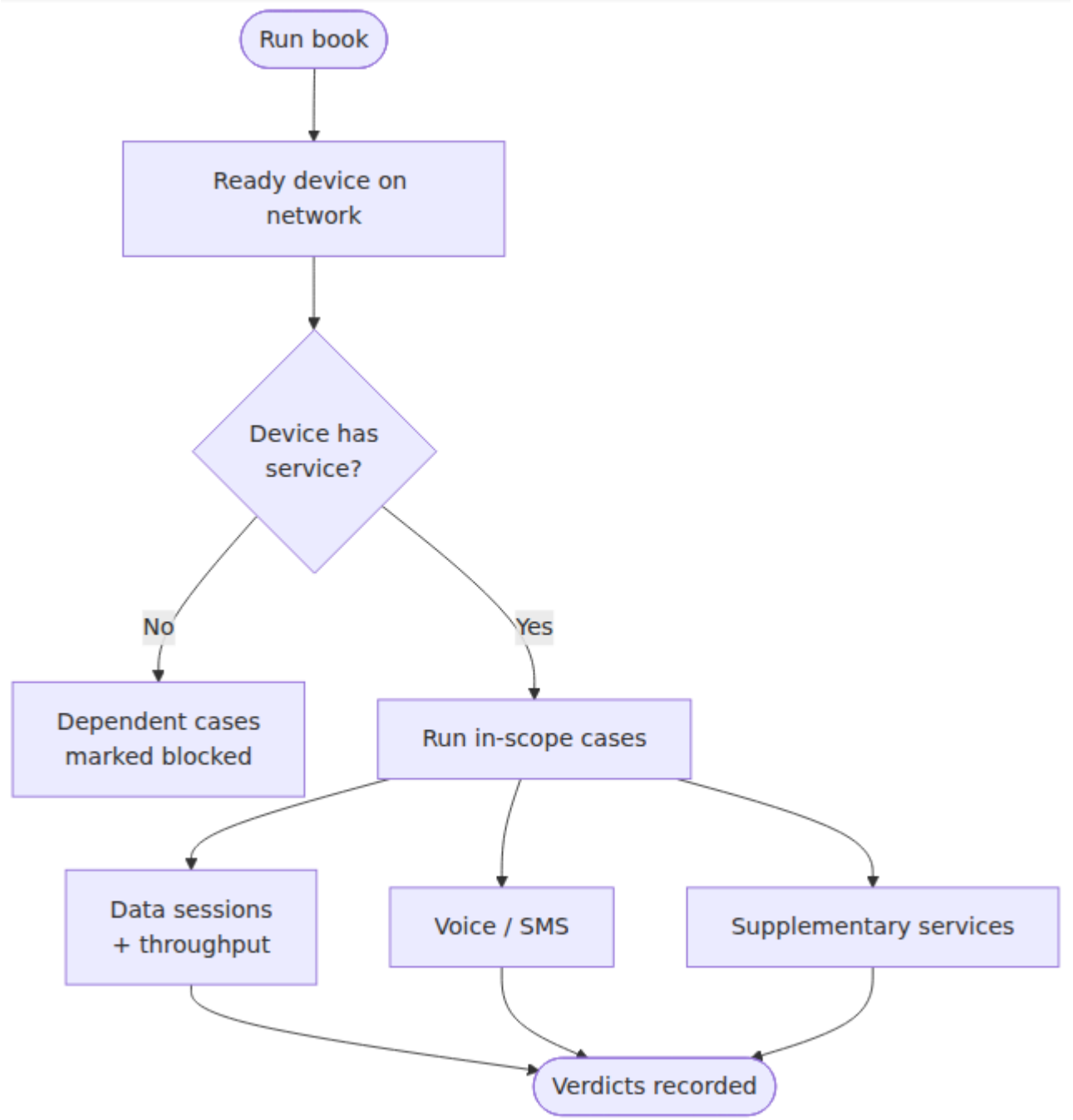
While a book runs, the screen shows each step as it happens, with the current step highlighted and a live pass/fail indication per case.

How Automated Cases Are Driven

Each automated case drives the test device, observes what the network does, and compares the outcome against the GSMA expected result. A case only passes on positive evidence — a step that cannot be confirmed is recorded as a failure with the reason, never a silent pass.

Data sessions and throughput

For data cases, OmniRoam establishes a packet data session, runs traffic to measure the achieved download/upload rate and latency, then disconnects so the network writes a clean charging record for the session. The data volumes and timings are read back from the serving gateway's charging records, and — for the IR.38 speed case — the measured rate is asserted against the throughput floor for the home/visited distance.



Voice calls

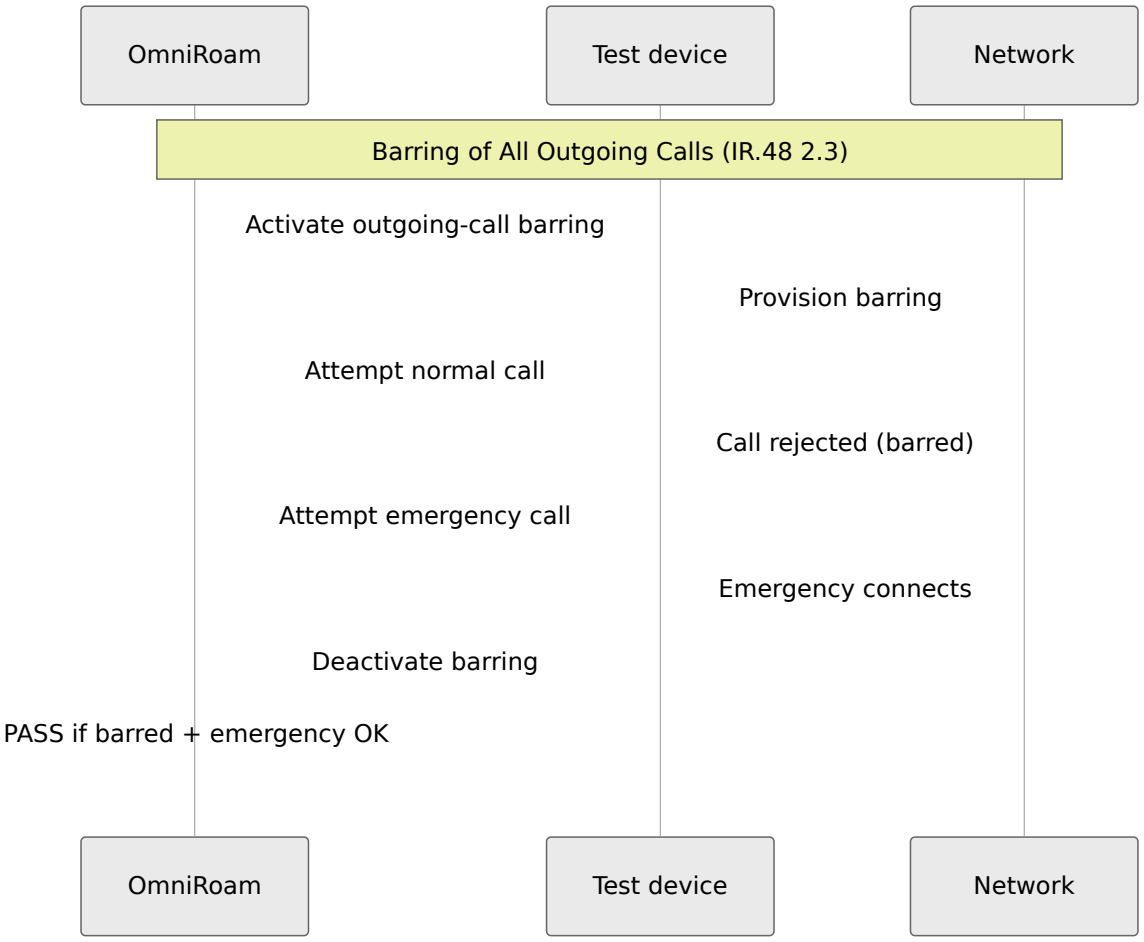
OmniRoam places a call from the device to a lab answering service, confirms the call reaches the active (answered) state, optionally holds it to confirm stability, then clears it. The verdict records the time to connect and whether the call stayed up.

SMS

OmniRoam sends a message with a unique body to a lab answering service and waits for the reply. Delivery of the reply confirms mobile-originated and mobile-terminated SMS end-to-end. A missing reply is a clear failure with the reason, not a crash.

Supplementary services (call barring and forwarding)

Supplementary services are provisioned over the air from the test device itself — exactly as a subscriber would using the standard service control codes — and then exercised:



For **Call Forwarding on No Reply**, OmniRoam registers the forwarding to a target number with a ring timer, then interrogates the service and confirms the network stored the forward-to number and timer that were registered.

Cancel Location (IR.38 3.1.2)

Some cases are driven from the network side rather than the device. For **LTE Cancel Location**, OmniRoam asks the home subscriber server to cancel the subscriber's location on the visited network, and records a pass when the request is accepted — confirming the home network can withdraw a roaming registration.

IMS registration (VoLTE)

For VoLTE, OmniRoam first confirms the device is registered to IMS. This gates the IMS-dependent cases: if registration is not confirmed, the VoLTE call and SMS-over-IMS cases are reported as blocked.

Customising a Run

A run does not have to cover the whole book. Using the **test picker**, the operator selects exactly which cases to run. This is useful for:

- Re-validating only the cases that previously failed.
- Running just the voice cases, or just the data cases, during fault-finding.
- Skipping cases that depend on an arrangement not yet in place (for example, a CAMEL agreement).

Cases left out of the scope keep their existing verdict and are untouched.

Re-running a Single Case

Every case has a re-run control. Selecting it runs just that one case again, against the same device and SIM, and updates only that case's verdict. This is the fastest way to confirm a fix — for example, re-running the throughput case after a transport change — without repeating the whole book.

Routine and Scheduled Runs

OmniRoam can run tests **unattended on a schedule** so a roaming relationship is continuously validated rather than checked once.

- **Routine checks** run a compact set of health checks against a device — data, speed, voice, and SMS — and report a combined pass/fail. These are the same drives used by the full books, packaged as a quick recurring confidence check.
- **Schedules** run a chosen routine on a recurring interval automatically. Each scheduled run records its own result, building a history over time so a regression (for example, throughput dropping below the floor, or SMS delivery failing) is caught and visible without anyone running a test by hand.

Setting Up a Schedule

Schedules are managed on the **Schedules** screen under Roaming Tests. A schedule defines:

Setting	Purpose
Name	Identifies the schedule (and labels its metrics and history).
Modem(s)	The test device(s) the routine runs against.
Checks	Which checks the routine performs — data, speedtest, voice, SMS.
Interval	How often the routine runs (for example, every hour).

Once saved, the schedule runs automatically at its interval with no further interaction. The **Routine Test** screen shows the run history — each run's combined pass/fail, the measured speeds, and the per-check detail — and a failing run surfaces the same evidence as an interactive run.

Prometheus Monitoring

Scheduled routine results are exported as **Prometheus metrics**, so roaming health can be alerted on and dashboarded alongside the rest of the network — the same way the scheduled tests on test devices are monitored. The exposition is served at `/api/ir38/routine/metrics` (reads the latest scheduled run per schedule and modem):

Metric	Type	Me
`omniweb_ir38_routine_pass`	gauge	Latest sche result (1 = p
`omniweb_ir38_routine_check_pass`	gauge	Per-check re latest run (l data/speedt
`omniweb_ir38_routine_download_mbps`	gauge	Download ra on the lates
`omniweb_ir38_routine_upload_mbps`	gauge	Upload rate the latest ru
`omniweb_ir38_routine_latency_ms`	gauge	Latency me latest run
`omniweb_ir38_routine_last_run_timestamp_seconds`	gauge	Unix time of scheduled r

Every series carries `schedule`, `serial`, `modem`, and `imsi` labels (plus `check` on the per-check metric), so you can alert per schedule, per device, or per home subscriber. The endpoint needs no login (scrapers can't carry one) and can be gated with a bearer token when required.

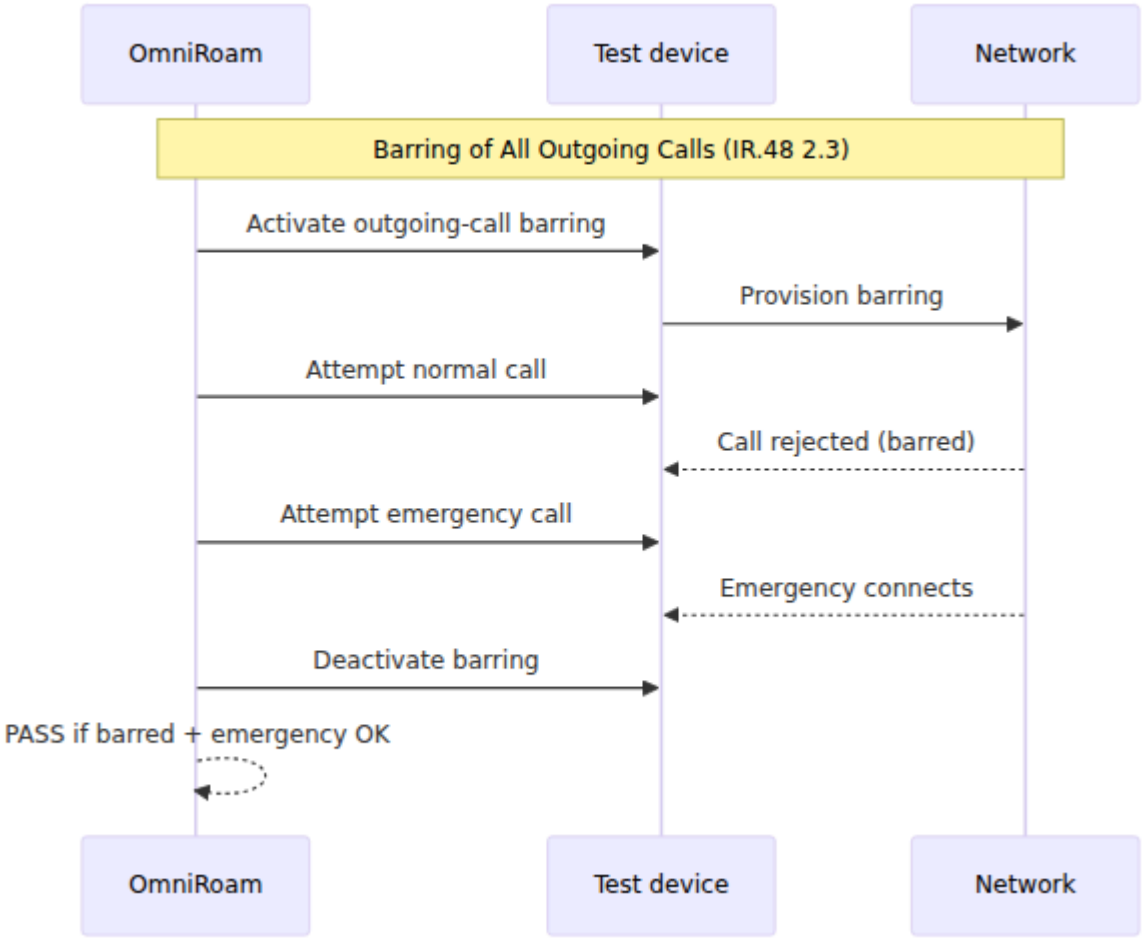
Example alert: roaming voice has been failing on a device for 15 minutes:

```
omniweb_ir38_routine_check_pass{check="voice"} == 0
```

Scheduled results accumulate into a history that can be reviewed at any time, and a failing run surfaces the same evidence as an interactive run.

Reading the Results

Each case shows its verdict — **PASS**, **FAIL**, or **NOT PERFORMED** — together with the evidence behind it:



- **PASS** — the GSMA expected result was confirmed, with the supporting figures.
- **FAIL** — the expected result was not met; the reason is recorded (for example, "throughput below floor", "no echo reply", or the network release cause for a call that did not connect).
- **NOT PERFORMED** — the case was not run, or is a manual case awaiting a tester.

When the results look right, generate the completed document — see Completed Test-Book Documents.

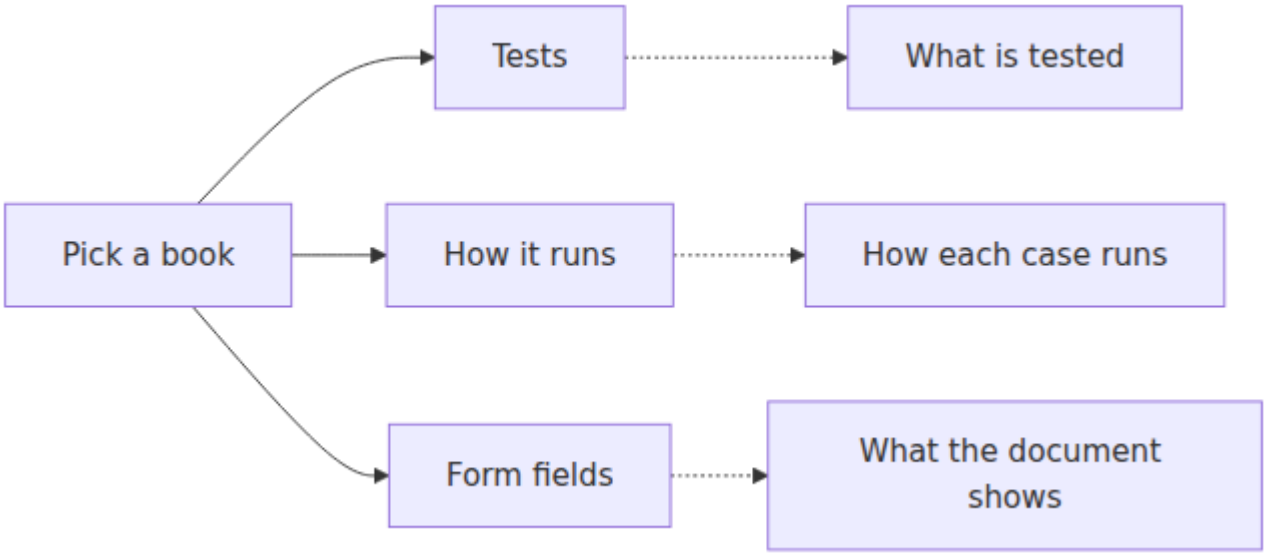
Test Book Explorer

The Test Book Explorer is a read-only reference view that shows, for each test book, exactly what it tests and how. It answers three questions an operator typically asks before trusting an automated run:

1. **What cases are in this book, and which are automated?**
2. **How is each case actually performed?**
3. **What ends up in the completed document, and where does it come from?**

The Explorer is reached from the Roaming Test Books screen via **Explore books**, and lets you switch between IR.38, IR.48, and VoLTE.

The Explorer showing the IR.48 book — each case listed with whether it is automated or manual and a description of how it is driven.



Tests

The **Tests** view lists every case in the book with:

- the **case number** and **title** (matching the GSMA specification),
- whether the case is **automated** or **manual**, and
- a plain description of **how it is driven** — for example "data session + charging-record check", "voice call", "barring activation and call attempt", or "tester records the result" for manual cases.

This is the quickest way to see, at a glance, how much of a given book OmniRoam performs for you and what remains a manual check.

How It Runs

The **How it runs** view shows the **step sequence** a full run of the book will perform, in order — readying the device, the data sessions, the throughput check, the voice and SMS drives, the supplementary-service drives, and so on. It reflects the cases currently in scope, so it is an accurate preview of what a run will do before you start it.

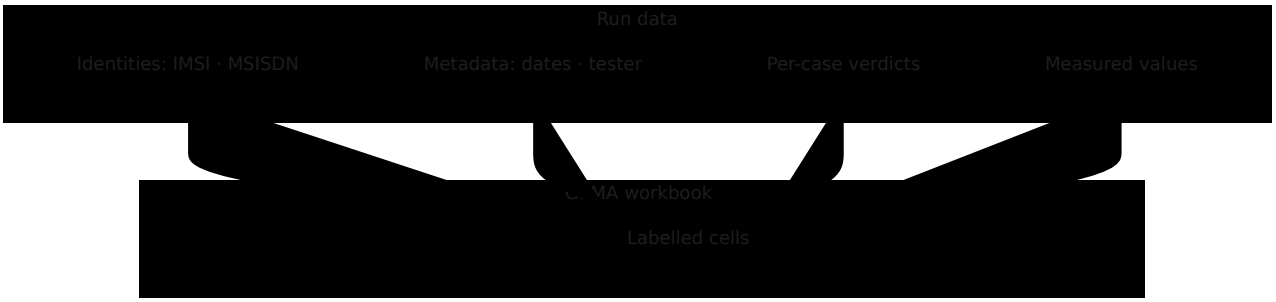
This view is useful for understanding the shape of a run and for confirming that a customised scope will perform the steps you expect.

Form Fields

The **Form fields** view lists every cell in the completed GSMA workbook that OmniRoam fills, each annotated with:

- the **sheet and cell** in the GSMA template,
- the **GSMA field label** (read from the template itself), and
- the **source** of the value — for example the subscriber MSISDN, the test date, a per-case pass/fail verdict, or a measured value such as data volume.

Because this list is derived from the same logic that produces the document, it cannot drift out of step with the actual export: what the Explorer says will be filled is exactly what gets filled. See Completed Test-Book Documents for the document itself.



Download Template

The Explorer also offers a **Download Template** action, which downloads the blank GSMA workbook for the selected book — the official template with the vendor's example/placeholder content removed, ready to fill in by hand if needed. This is the empty counterpart to the completed document a run produces (see Completed Test-Book Documents).

Why It Matters

A roaming partner accepting a completed test book needs confidence that the results are real and that the document faithfully represents what was tested.

The Explorer makes the mapping transparent: every automated case can be traced from the GSMA case number, through the way it is driven, to the exact cell it lands in on the final document.

OmniRoam Test Books

OmniRoam ships three GSMA roaming test books. Each book is the official GSMA test set, with cases identified by their GSMA case numbers so the completed document maps one-to-one onto the published specification.

- IR.38 — LTE Roaming Data Test
- IR.48 — Bilateral Roaming (Voice / SMS / MMS / Data)
- VoLTE — Roaming VoLTE Testing

For how each automated case is actually driven and judged, see Running Tests. To browse a book's cases interactively, use the Test Book Explorer.

In the tables below, **Driven** indicates whether OmniRoam performs the case automatically (**Auto**) or leaves it for the tester to record (**Manual**).

IR.38 — LTE Roaming Data Test

Based on **GSMA IR.38 v3.0**. Proves that a roaming subscriber can attach for data on the visited LTE network, reach services through the home network, and receive acceptable throughput — plus the circuit-switched fallback voice and SMS, and 2G/3G data, where those are in scope.

Case	Title	Driven	What it proves
`3.1.1`	LTE speed test (data + throughput)	Auto	A data session establishes through the home gateway, carries traffic, and meets the throughput floor.
`3.1.2`	LTE Cancel Location	Auto	The home network can cancel the subscriber's location on the visited network (triggered from the home subscriber server).
`3.1.3`	LTE Operator Determined Barring	Manual	Operator-determined barring applied in the home subscriber record is honoured.
`3.2.2`	CS Fallback — Mobile Originating Voice Call	Auto	A subscriber-originated voice call succeeds via CS fallback.
`3.2.3`	CS Fallback — Mobile Terminating Voice Call	Manual	A call to the subscriber succeeds via CS fallback.
`3.2.4`	SMS over SGs	Auto	Mobile-originated and terminated SMS works over the SGs interface.
`3.3.1`	Data Access from 2G/3G using PGW in HPLMN	Auto	A 2G/3G data session reaches services through the home gateway.
`3.4`	Data Access from 4G + 5G NSA over S8	Auto	Packet data is available and reaches the internet (confirmed by a data session with connectivity check).

Cancel Location (3.1.2)

OmniRoam automates Cancel Location by asking the home subscriber server to cancel the subscriber's location on the visited network, then confirming the request was accepted. This is the home-network-initiated removal of the roaming registration (per the location-cancellation procedure in 3GPP TS 29.272).

The Throughput KPI

Case `3.1.1` does more than confirm a data session — it measures the achieved download rate and asserts it against the IR.38 throughput floor. The floor depends on the geographic distance between the home and visited networks (longer paths permit lower rates):

Protocol	Distance < 4000 km	Distance ≥ 4000 km
HTTP	≥ 11 Mbit/s	≥ 5 Mbit/s
FTP	≥ 13.5 Mbit/s	≥ 6 Mbit/s

OmniRoam estimates the distance from the home and visited network identities and selects the correct threshold automatically; the operator can override the distance or the protocol when the defaults do not match the test setup. Recording a speed without testing it against the floor is never treated as a pass.

IR.48 — Bilateral Roaming (Voice / SMS / MMS / Data)

Based on **GSMA IR.48 v2.1** (internally the IR.24 test set). The classic 2G/3G bilateral roaming book: basic calls, supplementary services, SMS, and packet data, exercised for a subscriber roamed into the visited network.

Case	Title	Driven	What it proves
`2.1`	MS1(a) calls MS2(a), both roamed in VPLMN(b)	Auto	A basic voice call establishes and holds.
`2.2`	CAMEL negative test (MOC, prepaid, no agreement)	Manual	A prepaid call is correctly barred where there is no live CAMEL agreement.
`2.3`	Barring of All Outgoing Calls (BAOC)	Auto	With outgoing-call barring active, normal calls fail while emergency calls still succeed.
`2.4`	Call Forwarding on No Reply (CFNRY)	Auto	No-reply call forwarding can be registered and interrogated, with the stored configuration confirmed.
`2.5`	Mobile Originated / Terminated SMS	Auto	An SMS sent from the roamed subscriber is delivered and a reply is received.
`2.6`	CAMEL negative test (SMS-MO, prepaid)	Manual	Prepaid SMS behaviour where there is no live CAMEL agreement.
`2.7`	Intranet / ISP access using home gateway	Auto	A data session reaches the internet/intranet through the home gateway.
`2.8`	MMS using home gateway and MMSC	Manual	A multimedia message is delivered via the home MMSC.

Supplementary Services in IR.48

Two supplementary-service cases are fully automated:

- **BAOC (2.3) — Barring of All Outgoing Calls.** OmniRoam activates outgoing-call barring on the test subscriber, confirms that a normal outgoing call is rejected by the network, confirms that an emergency call still connects (the required exception), then removes the bar to leave the SIM clean.

- **CFNRY (2.4) — Call Forwarding on No Reply.** OmniRoam registers no-reply call forwarding to a target number with a ring timer, then interrogates the service (the equivalent of dialling the `*#61#` status code) and confirms the network stored the forward-to number and timer that were registered. Observing the live divert requires an inbound call from a third party, so that final leg is left for a manual check; the provisioning and interrogation are automated.

The status of every supplementary service is reported using the four standard 3GPP flags — **Provisioned**, **Registered**, **Active**, and **Quiescent** (see 3GPP TS 23.011).

VoLTE — Roaming VoLTE Testing

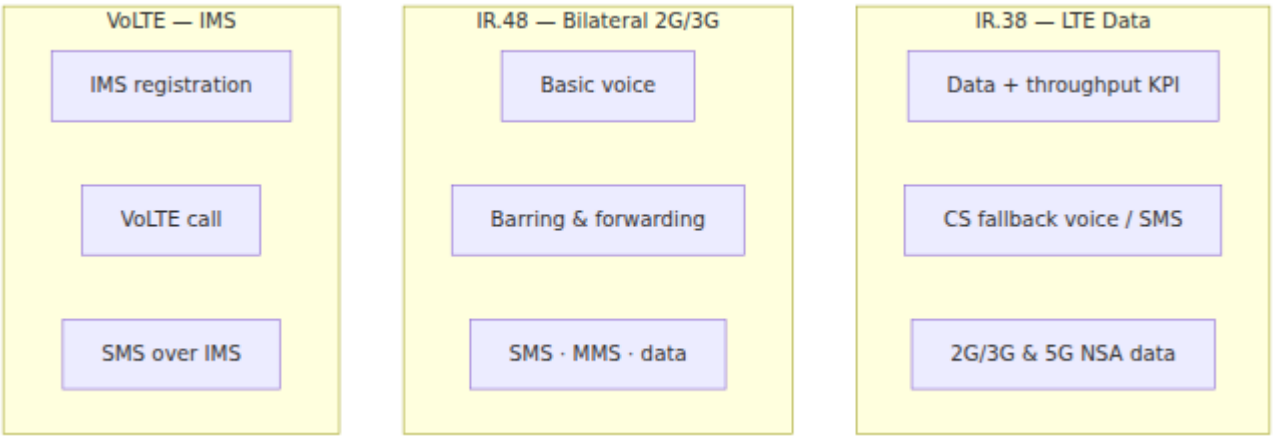
Based on **GSMA BA.65 §5.3**, with VoLTE behaviour per **GSMA IR.92**. Proves the subscriber can register to IMS while roaming and place/receive VoLTE calls and SMS-over-IMS.

Case	Title	Driven	What it proves
`ims_reg`	IMS registration over roaming	Auto	The device registers to IMS while roaming (the prerequisite for all VoLTE services).
`mo_call`	VoLTE mobile-originated call	Auto	A subscriber-originated VoLTE call establishes and holds.
`mt_call`	VoLTE mobile-terminated call	Manual	A call to the subscriber over VoLTE establishes.
`sms_ims`	SMS over IMS (SMSoIP)	Auto	SMS is delivered over the IMS network.
`emergency`	VoLTE emergency call	Manual	Emergency calling works over VoLTE (never auto-dialled).

The IMS Registration Gate

VoLTE services depend on IMS registration. OmniRoam runs `ims_reg` first as a **gate**: if the device is not IMS-registered, the IMS-dependent cases (`mo_call`, `sms_ims`) are reported as blocked rather than failed — there is no value in dialling a VoLTE call when IMS is down, and a blocked result tells the operator exactly why.

How the Books Compare



Book	Primary domain	Best for
IR.38	LTE / EPC data	Validating a new LTE roaming data relationship and throughput
IR.48	2G/3G circuit-switched + packet	Validating voice, supplementary services, SMS and basic data
VoLTE	IMS	Validating VoLTE roaming once IMS interconnect is in place

OTA SIM Management — over-the-air RFM/RAM

The **OTA** module manages SIMs **over the air**: it reads and edits files on a remote card by sending **secured packets** (ETSI TS 102 225 / 3GPP TS 31.115 — "03.48" / SCP80) that carry Remote File Management (RFM) and Remote Application Management (RAM) commands. The card verifies and decrypts each packet, executes the commands, and returns a signed **Proof-of-Receipt (POR)**.

In practice it gives you the same file-level editing as the offline SIM Filesystem Explorer — browse the file tree, decode an EF, edit the fields, write it back — except the card is reached **remotely**: through a PC/SC reader on the central host, a reader at a remote agent, or over **SMPP** to a SIM in the field.

It builds on the same per-ICCID key store as the SIM Key Store: a profile says *which keyset and algorithms*, the key store holds the actual **KIC/KID** for that card, and the engine never logs or returns the keys — only the resulting ciphertext and the POR.

What a secured packet is

A secured packet protects the command APDUs end-to-end, independent of the (plain) bearer. The engine has been validated against physical ISIM cards.

```
Command Packet  =  CPL | CHL | SPI | KIc | KID | TAR(3) | CIPHERED
CIPHERED        =  3DES-CBC( CNTR(5) | PCNTR(1) | CC(8) |
APDUs+padding )   IV=0
CC              =  last8( 3DES-CBC( KID,
zeropad(CPL+hdr+CNTR+PCNTR+APDUs) ) )
```

- **TAR** (Toolkit Application Reference, 3 bytes) routes the packet to a service on the card — a specific RFM application or the RAM (card manager).
- **SPI** flags select ciphering, the cryptographic checksum (CC), the counter policy, and what the POR must contain.
- **KIc / KID** index which transport (ciphering) and authentication keyset the card uses; the **values** come from the key store per ICCID.

The APDUs inside are an ordinary C-APDU script — `SELECT` + `READ BINARY` for an RFM read, `SELECT` + `UPDATE BINARY` for a write, GlobalPlatform `DELETE` / `INSTALL` for RAM.

On the wire (an SMS-PP) the packet rides in the TP-User-Data with a UDH Command Packet Identifier (IEI `0x70`), PID `0x7F` (SIM data download), DCS `0xF6` (8-bit, message class 2). In-shell it is delivered as a CAT `ENVELOPE(SMS-PP-Download)` APDU and the POR comes back synchronously.

Profiles, contexts, and auto-routing

An **OTA profile** is a reusable security preset — it holds **no key material**:

Field	Meaning
`tar`	3-byte Toolkit Application Reference (the target service).
`kic_index` / `kid_index`	Which keyset (1/2/3) → `KIC{n}`/`KID{n}` from the card's key-store row.
`cipher_algo` / `cc_algo`	Ciphering and checksum algorithm (`triple_des_cbc2`, `aes_cbc`, …).
`ciphering`, `rc_cc_ds`, `counter_mode`	SPI: encrypt? CC vs RC vs none? replay-counter policy.
`por`, `por_ciphered`, `por_rc_cc_ds`	POR required? ciphered? signed?

Two profiles are seeded, matching the values validated on sysmoISIM:

- **RFM-USIM-3DES-keyset3** — TAR `B00011`, keyset 3, 3DES, ciphering + CC, POR required + ciphered + CC. Operates inside **ADF.USIM**.
- **RFM-SIM-3DES-keyset2** — TAR `B00010`, keyset 2. Operates on the **MF-rooted** 2G tree (DF.GSM / DF.TELECOM).

Different applications need different services, so the explorer **routes the profile and the SELECT context automatically per file**, from the file's application:

File application	Context	Profile (by TAR)	Pre-SELECT before the ENVELOPE
`ADF.USIM`	`usim`	TAR `B00011`	`SELECT ADF.USIM` (by AID)
`DF.GSM`, `DF.TELECOM`	`sim`	TAR `B00010`	none (MF-rooted)
`ADF.ISIM`	`isim`	TAR `B00013`	`SELECT ADF.ISIM` (by AID)

Each context maps to one of your profiles (defaulted by matching TAR, overridable in the explorer). Pick a card and click any file — USIM, GSM or TELECOM — and the right profile, keyset and SELECT are applied without switching anything. ISIM works as soon as you create a profile for it (its TAR varies by card).

Transports

A transport delivers the secured packet and, where the bearer allows, collects the POR. The pluggable adapters share one interface:

Kind	Reaches	POR	Notes
`local_pcsc`	a card in a PC/SC reader **on the central host**	synchronous	the directly-attached validation path.
`agent_reader`	a card in a reader **at a remote agent**	synchronous	rides the same agent WebSocket as Test Devices / SIM Bank (`session_apdu` relay).
`smpp`	a **field SIM** over SMPP `submit_sm`	asynchronous	MT SMS to an SMSC; the POR returns later as an MO SMS.

The **Explorer** uses only the in-shell transports (it needs a synchronous POR); **Campaigns** can use any, including SMPP.

Explorer

`src/pages/ota/OtaExplorer.tsx` — the remote SIM file editor.

1. Pick a **target ICCID** (from the SIM Key Store) and a **transport**.
2. Browse the **file catalog** — every known EF from the offline file model (ADF.USIM, ADF.ISIM, DF.GSM, DF.TELECOM), grouped by application and searchable by code / name / description.
3. Select a file → **Read over OTA**. The secured RFM read is delivered, the POR decoded, and the EF rendered: POR status, last SW, raw hex + ASCII, and a **typed field form** (decoded to named fields).
4. Edit the **fields** (re-encoded automatically) or the **raw hex**, then **Write over OTA**. The secured RFM `UPDATE` is sent and the file is **read back** automatically so you see the on-card state after the write.

Reads/writes are gated by the `ota` control permission; everything is recorded as a **job**.

Campaigns

`src/pages/ota/OtaCampaigns.tsx` — a saved operation fanned out across a set of SIMs: **target query → operation → profile → transport**.

- **Target query** is evaluated against the SIM Bank inventory at run time: ICCID list / prefix, IMSI prefix or range, agent, bank. An empty query deliberately matches nothing (no accidental fan-out).
- **Operation** is the RFM/RAM command (read/update binary or record, GP delete / get-status / install).
- **Run** creates one **job** per matched card. SMPP campaigns are fire-and-forget (the POR arrives later as an MO SMS).

A run is bounded (`CAMPAIGN_MAX_TARGETS`) so a fat-finger query can't blast thousands of cards in one request.

Jobs

`src/pages/ota/OtaJobs.tsx` — every OTA delivery (explorer reads/writes and campaign runs) with its status and POR. The detail drawer shows the **secured request packet** (ciphertext), the **Proof-of-Receipt**, and the decoded result (commands executed, last SW, response data). Card secrets are **never** stored on a job — only the ciphertext and POR.

REST API (`/api/ota`, JWT, RBAC

element type `ota`)

Method	Route	Purpose
GET	`/profiles` · POST · `/profiles/<id>` PUT/DELETE	OTA security profiles.
GET	`/campaigns` · POST · `/campaigns/<id>` GET/PUT/DELETE	Campaigns.
GET	`/campaigns/<id>/targets`	Preview which SIMs the query currently matches.
POST	`/campaigns/<id>/run`	Execute — one job per matched card.
GET	`/jobs`, `/jobs/<id>`	Job history (filter by `campaign_id` / `iccid`).
GET	`/files/catalog`	The browsable EF catalog (offline file model).
POST	`/explore/read`	Read one EF over OTA; returns the job + decoded EF.
POST	`/execute`	Run one operation (RFM update, RAM, raw) against one card.
POST	`/preview`	Build the secured packet **without** sending (inspect the ciphertext + SMS framings).

Method	Route	Purpose
POST	`/decode` · `/encode`	Decode EF bytes ↔ typed fields (offline).

RBAC mirrors `simbank`/`ir38`: admins always pass; otherwise a JWT permission with `element_type == "ota"` grants view, and `POST` in its methods grants control. Reads are treated as control (they transmit a packet to the card). Routes are auto-documented in the API Reference.

Database schema

Table	Row	Key columns
`ota_profiles`	one security preset	`name`, `tar`, `kic_index`/`kid_index`, algorithms, SPI flags.
`ota_campaigns`	one saved fan-out	`name`, `profile_id`, `target_query` (JSON), `operation` (JSON), `transport` (JSON), `status`.
`ota_jobs`	one delivery to one card	`iccid`, `operation`, `request_hex` (ciphertext), `response_hex` (POR), `por_status`, `result` (JSON), `status`.

Keys live in the existing `simbank_keys` table (the SIM Key Store), referenced per ICCID by the profile's keyset index — **not** duplicated here.

How it works

The OTA engine handles the crypto, secured-packet structure, CAT/SMS TLVs and the offline file model needed to build and decode 03.48/SCP80 RFM/RAM

packets. It runs in the agent-role backend service and shares the agent connection for the in-shell transport. Delivery is pluggable across the local-reader, remote-agent-reader, and SMPP transports.

Security notes

- Card secrets (KIC/KID) are read from the key store to build packets but are **never** written to the audit log or returned in any response — only the ciphertext request and the POR are stored.
- Writing a file is destructive; the explorer confirms before each write and reads back the result.
- The secured packet is genuine 03.48: tamper a byte and the card rejects it (the CC fails); with ciphering on, the plaintext APDUs never appear on the wire.
- SCP80 only — this is **not** SCP81 (no PSK-TLS / HTTPS Admin Agent).

PGW-C — PDN Gateway (Control Plane)

The **PGW-C** is the control-plane half of the EPC **PDN Gateway**. It is the anchor point for every subscriber's data session: it terminates **S5/S8** to the SGW-C, sets up and holds each **PDN connection**, allocates the UE its IP address from an operator **IP pool**, and selects and controls the user plane (**UPF**) over **Sx/PFCP**. It also runs the session's policy and charging over Diameter — **Gx** to the PCF/PCRF for policy and QoS, and **Gy** to the OCS for online charging — and raises **CDRs** for offline charging. In the 5G core the equivalent function is the **SMF**. OmniWeb surfaces the PGW-C's live sessions, IP pools, UPF and Diameter peers, a topology map and session history, and diagnostic tools; longer-run trends are on the Grafana dashboards reached from the same page.

← Operations Guide

The PGW-C is a per-instance element page reached from the sidebar when one runs in the site's inventory, with a fleet view where several are deployed. All traffic goes through the OmniWeb backend proxy to the PGW-C's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

A single-screen health read: the **API status** of the PGW-C and a summary of its live state — active session count, IP-pool utilisation, UPF peers and Diameter peers. It is the quickest way to confirm the node is up and carrying sessions.

Sessions

Every active PGW session, each row keyed by subscriber **IMSI** and **MSISDN** with its assigned **UE IP** and **APN**. Sessions can be searched by IMSI, UE IP, MSISDN or APN. Opening a session shows its full state — the PDN connection and bearers, the serving SGW and selected UPF, and the live Gx/Gy policy and charging context. A session is editable here: an operator can adjust live state such as the session **AMBR** (which pushes a new APN-AMBR down to the UPF) and the requested **Gx event triggers** that control location-change reporting. An operator can also **delete** a session to force-release a stuck or misbehaving subscriber. Because edits are applied directly to live state, they take effect immediately.

IP Pools

The operator IP pools the PGW-C allocates UE addresses from, each showing its **size**, **allocated** count and **utilisation**. Opening a pool lists the individual allocated addresses. This is where you confirm a pool is not exhausted — an exhausted pool is a common cause of session-setup failures — and see which subscriber holds a given address.

UPF Peers

The user-plane (UPF) peers the PGW-C controls over **Sx/PFCP**, each showing its **IP address** and PFCP **association state**. Opening a peer shows its details. This is where you confirm the PFCP associations that carry every session's user plane are up.

Diameter

The Diameter peers the PGW-C talks to — the **PCF/PCRF** over **Gx** for policy and the **OCS** over **Gy** for online charging — each with its **hostname** and **connection state**. Opening a peer shows its details. This is where you confirm the policy and charging paths are up when sessions are being rejected or throttled.

History

A searchable, paged log of session lifecycle **events** — *created*, *modified*, *deleted*, *handover* and *failed*. Events can be filtered by type and searched by IMSI, MSISDN, IP address or TEID. This is where you reconstruct what happened to a specific subscriber's session over time, including failures.

Topology

A map of the PGW-C and the neighbours it works with — the SGWs it anchors sessions for, the UPFs it controls, and its Diameter peers — giving a one-picture view of which signalling and user-plane neighbours the node is connected to.

UPF Selection

The rules that decide which UPF a new session is placed on, together with the current **health status** of each candidate UPF and the **PCO** (Protocol Configuration Options) configuration the PGW-C returns to the UE. This is where you confirm why sessions are landing on a particular UPF and check that all candidates are healthy.

UE Search

A direct lookup of a single subscriber by **IMSI**, **MSISDN** or **IP address**, returning that UE's information on the PGW-C. It is the fastest way to find one subscriber's session without scrolling the full Sessions list.

Charging

The live **online-charging (Gy/OCS)** state of every active session, each row keyed by **IMSI** and **MSISDN** and searchable by either. At a glance each session shows whether **Gy** online charging is on (with its **session id**, and the **OCS host** and **realm** it talks to), whether **offline (Gz)** accounting is also running, and whether the OCS has pushed the subscriber into a **walled garden** redirect. Opening a session expands its full charging detail — the per-URR **granted** quotas against the **used** volume (uplink, downlink and total octets), the **final-unit action** the OCS set for when credit runs out, the offline (Gz) counters, and any redirect target. This is how you read a subscriber's live credit-control state — whether they have credit, how much of their grant they have consumed, and why they may have been throttled or redirected.

CDRs

The offline **charging records** the PGW-C writes for billing — one row per session **start**, **interim** and **stop** event, most-recent first, and filterable by **IMSI** or **MSISDN**. Each record carries the subscriber's **IMSI**, **MSISDN** and **IMEI**, the **APN** and **QCI**, the **charging id** that ties a session's records together, the **uplink/downlink octets** counted for that interval, the **S-GW**, **PGW** and **UE** IP addresses, and the **source file** the record was read from. This is where you reconcile a subscriber's data usage against the offline (Gz) accounting the operator bills on, and trace which record file a given session landed in.

Gy Test

An operator tool that issues a **Gy Credit-Control-Request** against the OCS to validate the online-charging path end to end. You supply an **IMSI** (and optionally an MSISDN and a requested/used octet count), pick the request type — **Initial (CCR-I)** to open a session and request units, **Update (CCR-U)** to report usage and ask for more, or **Terminate (CCR-T)** to report final usage and close it — and send it; the rating group, Service-Context-Id and Auth-Application-Id are defaulted by the PGW-C. The parsed **Credit-Control-Answer** comes back on the same screen. Unlike the other tabs this one **does not fetch anything on load** — it is a form you submit — and it is the quickest way to prove the OCS is reachable and granting credit when online charging is misbehaving.

P-CSCF Monitor

The status and metrics of the P-CSCF discovery monitor — the PGW-C returns a P-CSCF address to VoLTE devices during PDN setup, and this tab confirms that discovery is healthy so IMS registration can proceed.

Logs

The **Logs** tab tails the PGW-C's live log output for troubleshooting a single node — session-setup failures, PFCP and Diameter errors — without leaving OmniWeb.

Monitoring

Session-count trends, throughput and other time-series for the PGW-C are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

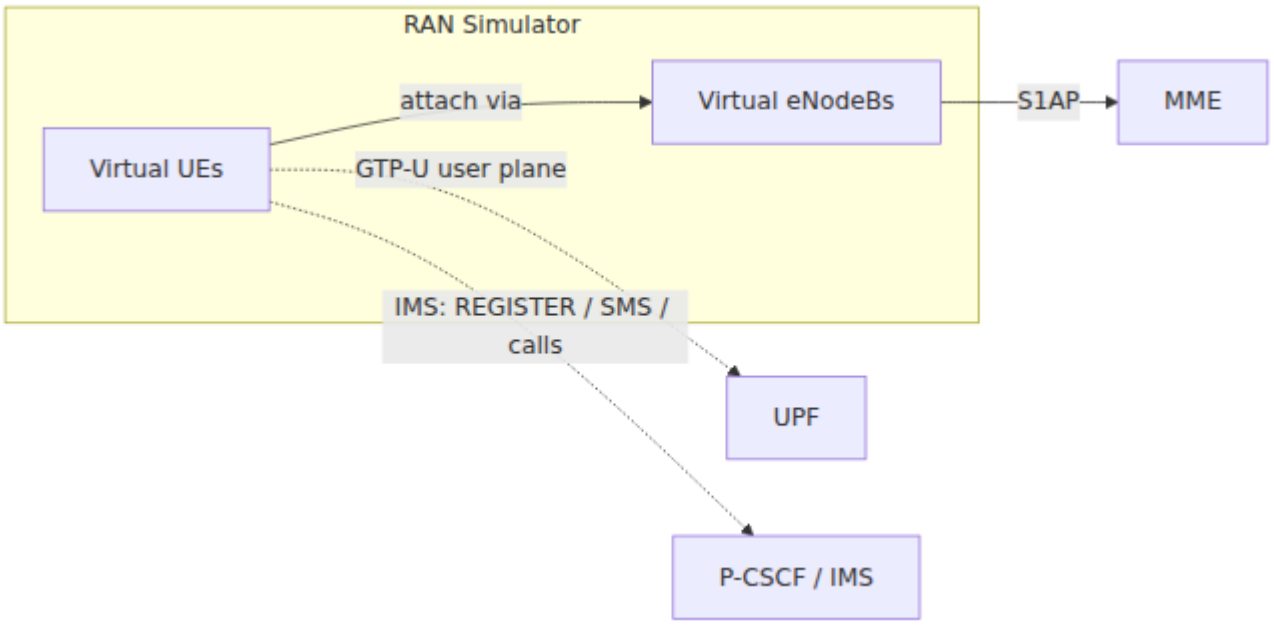
- The **SGW-C** is the control peer over **S5/S8**; it anchors the access-side bearers the PGW-C connects to the PDN.
- The **UPF** is the user plane the PGW-C selects and controls over **Sx/PFCP**.
- The **PCF/PCRF** provides policy and QoS over **Gx**.
- The **CHF/OCS** performs online charging over **Gy**; the PGW-C also raises CDRs for offline charging.

RAN Simulator

The **RAN Simulator** lets you exercise a live core network without physical radios or handsets. It stands up virtual **eNodeBs** and **UEs** that speak real S1AP/NAS to the MME and GTP-U to the gateways, so you can drive attaches, data sessions, IMS registrations, SMS and calls — for functional testing, load testing, or reproducing field scenarios — all from OmniWeb.

← Operations Guide

What It Is



The simulator runs alongside the core it targets. In OmniWeb it appears as its own **RAN Simulator** area (Network Map, eNodeBs, UEs, IMS, Load Test), and — because a simulated UE behaves like a device — its UEs also appear in **Test Devices** so you can drive them from the same place as real handsets.

By default it starts with **no eNodeBs** — you turn up exactly the topology you want.

eNodeBs

Turning up an eNodeB attaches it to the MME (S1 Setup). An eNB can be connected to **more than one MME** (an MME pool) — the Network Map shows the full set of MMEs an eNB is homed to.

Turn-up is **fire-and-forget**: OmniWeb asks the simulator to bring the eNB up and returns; the S1 link establishes in the background (watch the Network Map / eNodeBs list to confirm it came up).

Disconnecting an eNodeB

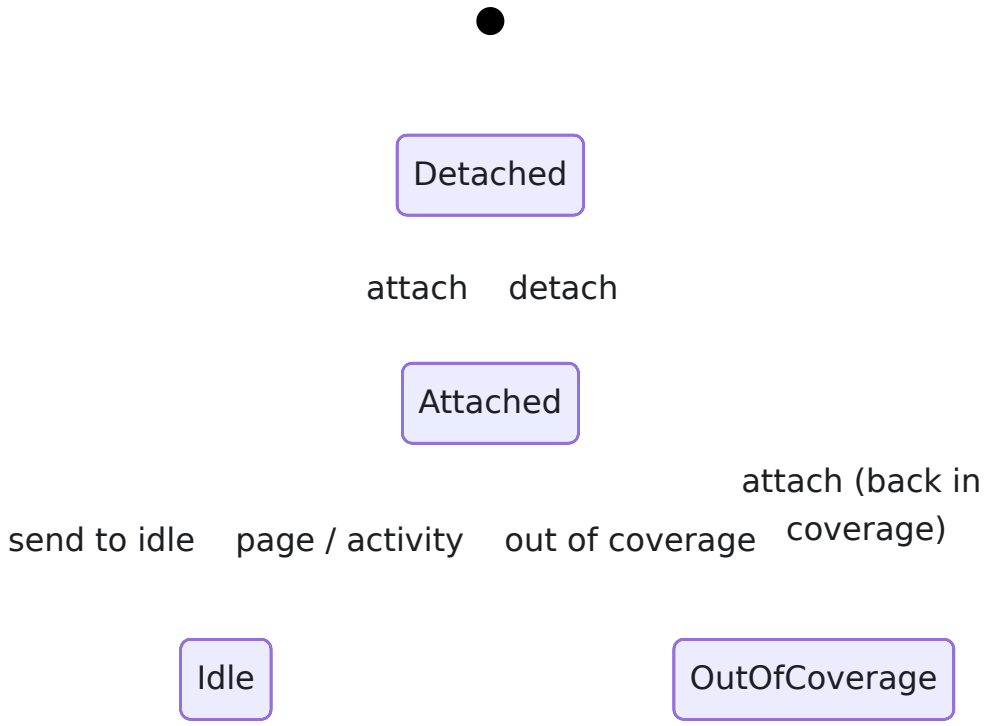
You can take an eNB down two ways, which model different real-world events:

Action	Models	Effect on UEs
Clean S1 disconnect	An orderly eNB shutdown / S1 reset	UEs are released cleanly and **survive** — they re-attach when a link is available
Drop link	The eNB abruptly losing its S1 (fibre cut, power loss)	The S1 association is aborted; UEs **survive** the eNB going away and re-attach rather than being torn down

Critically, taking an eNB down does **not** destroy its UEs — the UEs persist and re-attach, so you can simulate transport outages without rebuilding your fleet of subscribers.

UEs

Each simulated UE has a lifecycle you can drive directly:

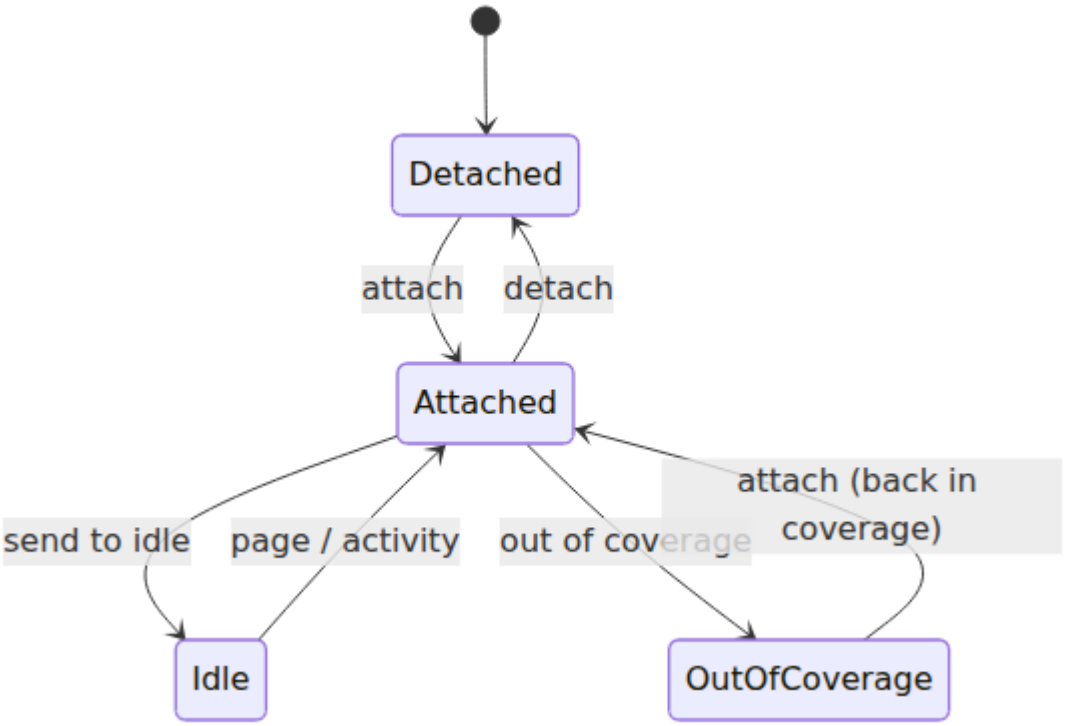


- **Attach / Detach** — join or leave the network.
- **Send to idle** — move to ECM-IDLE (as a real UE does between activity); paging or new activity brings it back.
- **Out of coverage** — an *unclean* loss of radio (the UE drops without signalling), to test how the core handles a UE that simply vanishes. Attaching brings it back into coverage.

For each UE the simulator shows the **serving MME** it is attached to — the MME Group ID and MME Code (decoded from the UE's GUTI) — so in a pooled deployment you can see which MME picked up each subscriber.

IMS registration & location

A simulated UE can register to IMS and place SMS/calls. When it registers, it includes its **cell location** (`P-Access-Network-Info` with the `utran-cell-id-3gpp` for its serving cell), exactly as a real UE does — so the S-CSCF's third-party REGISTER propagates that location to the SMSc, TAS, and other application servers. This lets you test location-dependent IMS behaviour end to end without real radios.



Driving UEs from Devices

Because a simulated UE is just another device, it shows up in **Test Devices** tagged as a simulator, and can be filtered to there. The **same UE control panel** is used whether you open a UE from the RAN Simulator's Network Map or from the Devices fleet — one place to attach/detach, idle, drive out-of-coverage, register to IMS, send an SMS or place a call, and watch the action log.

Filter Test Devices to the RAN Simulator UEs to drive emulated subscribers alongside real handsets.

Typical Uses

- **Functional testing** — turn up an eNB and a UE, attach, bring up a data session, register to IMS, send an SMS — confirm each core element behaves.
- **Load testing** — bring up many UEs to generate attach/session/IMS load against the core (see the Load Test view).

- **Scenario reproduction** — model outages (drop link, out of coverage) and pooled-MME behaviour to reproduce and diagnose field issues.

Related

- Test Devices — real handsets and simulated UEs, driven from one place.
- Session Trace — correlate the signalling the simulator generates.

Remote SIM & Virtual SIM

OmniWeb can present **any SIM to any modem over the network**, and can even present a SIM that **does not physically exist** — emulated from a saved card image plus the subscriber's keys held in the HSS. This turns a rack of SIM readers, card emulators ("cardems") and modems at one or more sites into a shared pool: any tester can drive any subscriber profile against any modem without walking a plastic SIM to a card slot.

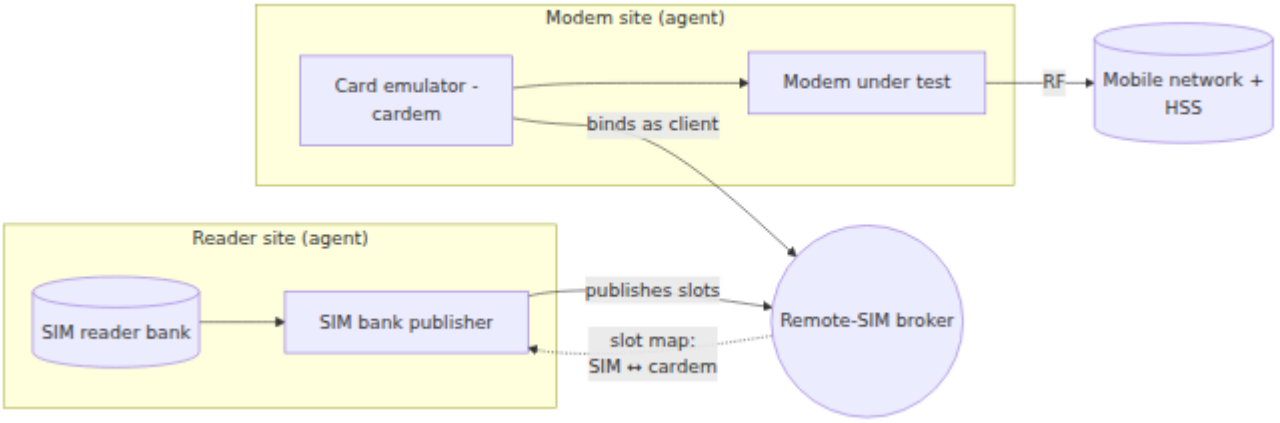
This page is the operations guide for that capability. The **inventory** half — discovering which SIMs are in which readers — is covered in SIM Bank; this page covers **presenting** a SIM (real or virtual) to a modem and using it in a test. Saving a card's file system as a portable profile package — and using one to back a virtual SIM — is covered in SIM Images (SAIP Profile Packages).

Table of Contents

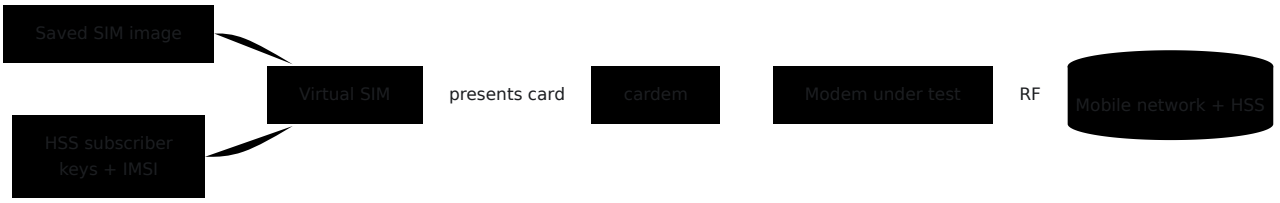
Architecture Overview

A SIM reader bank, one or more card emulators, and the modems under test connect to OmniWeb through its device agents. A central broker maps a chosen

SIM slot to a chosen cardem; the cardem then presents that SIM to the modem plugged into it.



For a **virtual** SIM, the reader bank is replaced by a software card built from a saved SIM image and the subscriber's keys — everything else is identical:



Concepts

Term	Meaning
SIM bank	A set of smart-card readers on an agent, published so their SIMs can be presented elsewhere. Each populated reader is one **slot**.
cardem	A card-emulator board (USB) that pretends to be a SIM towards a modem plugged into it. It plays back whatever card it is mapped to.
Slot map	The binding "present **this** SIM slot to **that** cardem". Creating it connects the two across the network; removing it disconnects them.
Modem under test	The modem behind a cardem. It attaches to the live network using whichever SIM the cardem is presenting.
Virtual SIM	A software card built from a saved SIM image + a subscriber's keys, presented to a cardem exactly like a real SIM — no physical card required.

The SIM Bank — what's where

The **SIM Bank** page lists every reader slot on every connected agent, grouped by agent, so the bank layout is the truth — empty, unreadable, and in-use slots all show, not only populated ones.

*Each row is one physical reader. The status chip shows whether a card is present and readable; the row then shows the SIM's identity (ICCID, IMSI, MCC/MNC, country, operator name). Readers held by an active emulation session show as **In use**. Persisted SIMs whose reader is currently offline appear under **Previously seen** and reappear when the agent reconnects with the SIM inserted.*

Status chip	Meaning
SIM	A card is present and its identity was read.
Card present	A card is physically seated but its identity could not be read (e.g. an incompatible reader/card pairing).
In use	The reader is held by a remote-SIM/emulation session right now.
Empty	No card in the slot.

Per-row actions open the **SIM Filesystem** explorer or the **APDU Terminal** for that exact slot, and **Rescan** re-reads an agent's readers on demand.

Remote SIM — connect a SIM to a cardem

The **Remote SIM** page is where you bind a SIM to a cardem. Bindings work **across agents and sites** — the SIM can be in a reader at one location and the cardem/modem at another.

Steps

1. **Start the SIM bank** on the agent that holds the readers (**Start bankd**). This publishes its reader slots.
2. **Start the cardem client** for the card emulator you want to use (**Start client**). This binds the cardem so a SIM can be mapped to it.
3. Under **Connect a SIM**, pick a published slot and choose a cardem to bind it to. The mapping is created and shows under **Active mappings** as **ACTIVE**.
4. The cardem now presents that SIM to its modem. Reset the modem (see the cardem console) so it reads the newly-presented SIM and attaches.

Global Reset tears down all mappings and returns every cardem to an idle state — use it to recover from a confused test bench.

Field (Active mappings)	Meaning
State	`ACTIVE` once the cardem is presenting the SIM.
SIM (bank/slot)	Which agent's bank and slot the card lives in.
ICCID	The SIM presented.
Cardem (client/slot)	Which cardem (and which agent) is presenting it.
By	The operator who created the mapping.

The Cardem Console

In **Devices**, a card emulator is shown as its own device type (`ngff-cardem`), alongside modems and phones — it is never opened as a phone. Unknown device types are labelled plainly rather than guessed.

Opening the cardem gives a dedicated console for the emulator and its modem.

Control	What it does
Start client	Binds the cardem as a remote-SIM client so a SIM can be mapped to it.
Reset modem	Re-presents the emulated card, which makes the modem re-read the SIM and re-run attach — the way you "swap the SIM" without touching hardware.
Remote SIM	Connect or disconnect which remote SIM is presented to this cardem.

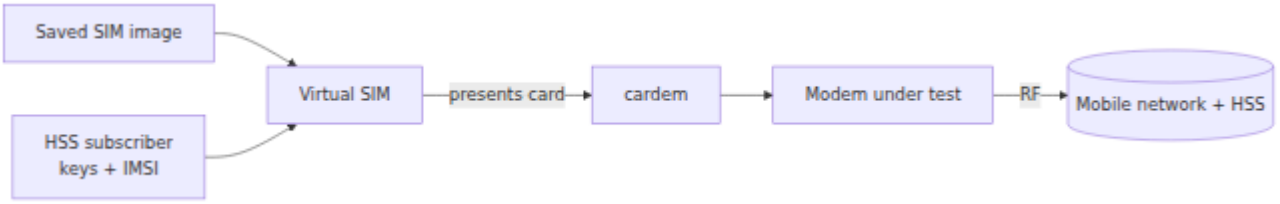
The **SIM source** panel states honestly what is and isn't available: the cardem feeds the modem a *remote* SIM. Presenting a *local* card seated on the emulator board is not offered here yet, and the console says so rather than showing a control that does nothing.

Virtual SIM — a subscriber with no physical card

A **virtual SIM** lets a modem attach as a subscriber **you do not have a SIM for**. It is built from two things you already have:

1. A **base SIM image** — a saved snapshot of a real card's file structure (any compatible test SIM works as the base; see **SIM Images**).
2. The subscriber's **identity and keys from the HSS** — the IMSI and authentication keys of a subscriber already provisioned in the network.

OmniWeb combines them into a software card that answers the modem's file reads from the image and answers the network's authentication challenge using the HSS keys — overlaying the chosen subscriber's IMSI. Presented through a cardem, the modem cannot tell it from a plastic SIM: it reads the IMSI, runs authentication, and **attaches to the live network**.



When to use it

- Reproduce a specific subscriber's roaming or attach behaviour without sourcing their physical SIM.
- Exercise many provisioned subscribers against one modem, back to back.
- Test profiles that only exist in the HSS (newly provisioned, pre-shipment).

What it proves. A modem on the bench has attached to the live network (`registered, home network`) as an HSS subscriber for which no plastic SIM exists — reading that subscriber's IMSI and completing the network's authentication entirely from the virtual card.

> A dedicated **Virtual SIM Reader** element — where you "load" a virtual SIM from a chosen image and subscriber and present it like any other reader slot — is the next step on this capability. Today the virtual card is wired through the same Remote SIM broker and cardem path described above.

Using a SIM in a Roaming Test Book

Roaming **Test Books** (GSMA IR.38 / IR.48) can drive the SIM presentation as part of a run. In a test book's **Run Tests** panel you can choose:

- a **SIM (by ICCID / IMSI)** — tracked by identity, its current reader resolved automatically at run time, and
- a **Cardem modem** — the cardem the SIM is emulated to.

Present SIM to cardem establishes the mapping before testing; **Go — Run All Tests** presents the SIM first (when both a SIM and a cardem are selected), then reads the IMSI and runs the data cases. If the SIM is not currently in a reader, or the cardem's client is not running, the panel says exactly why instead of silently running against the wrong card.

See Test Devices and the test-book documentation for the rest of the IR.38 flow.

Troubleshooting

A SIM doesn't appear in the bank

Symptoms: a reader is plugged in but no slot (or an empty slot) shows.

Possible causes & resolution:

- The card isn't seated, or the reader/card pairing can't power the card — the slot shows **Card present** or **Empty**. Re-seat the SIM.
- The reader was just plugged in — readers are re-scanned on hot-plug, but you can force it with **Rescan** on the agent's group.

The modem won't attach after connecting a SIM

Possible causes & resolution:

- The modem hasn't re-read the SIM. Use **Reset modem** on the cardem console to re-present the card.
- The mapping isn't **ACTIVE** — confirm under **Active mappings** on the Remote SIM page that the SIM is bound to the right cardem.
- The subscriber isn't provisioned (or is barred) in the serving network's HSS.

"Card present (unreadable)" on a reader

Symptoms: a card is detected but its identity can't be read.

Resolution: that reader cannot talk to that card (commonly a voltage/protocol mismatch on inexpensive readers). Move the SIM to a different reader in the bank.

Virtual SIM: modem reads the card but doesn't attach

Possible causes & resolution:

- The chosen subscriber must be **enabled** and fully provisioned (with a data profile) in the HSS for the serving network.
- The base image must come from a compatible card type. Re-capture the base image from a known-good SIM if init stalls.

SCEF — Service Capability Exposure Function

The **SCEF** (Service Capability Exposure Function), Omnitouch's **OmniSCEF**, is the network's **northbound gateway for the Internet of Things**. It exposes core-network capabilities to external Application Servers over the **3GPP T8** RESTful API defined in TS 29.122, so a machine-to-machine platform can subscribe to events about a fleet of devices and exchange small amounts of data with them without ever speaking Diameter or GTP. It fronts two families of service: **Monitoring Events** — asking the network to report when a device becomes reachable, loses connectivity, changes location and so on — and **Non-IP Data Delivery (NIDD)** — carrying small non-IP payloads to and from constrained NB-IoT devices over the control plane. Internally the SCEF reaches the MME over the **T6a** Diameter interface (TS 29.128); OmniWeb surfaces its status, the T8 resources an Application Server has created, and the live NIDD sessions it is carrying.

← Operations Guide

The dashboard is a per-instance element page reached from the sidebar when a SCEF runs in the site's inventory. All traffic goes through the OmniWeb backend proxy to the SCEF's T8 API, so it stays behind the single authenticated gateway. A documentation link in the page header gives in-context help. The page is organised into tabs.

The SCS/AS ID model

Every T8 resource — every monitoring subscription and every NIDD configuration — is **scoped to the SCS/AS identity** the Application Server registered under (the *Services Capability Server / Application Server*). The T8 API has **no endpoint that enumerates the SCS/AS IDs** in use, so OmniWeb cannot discover them for you: the operator **sets the SCS/AS ID** on the page,

and OmniWeb remembers it **across the SCEF tabs** so you set it once and it carries from Overview to Monitoring Events to NIDD. Until an SCS/AS ID is set, the scoped tabs prompt for one rather than showing an empty list — the resources exist under an identity you have to name. Enter the SCS/AS ID an Application Server was onboarded with (for example `iot-platform-01@omnitouch`) to work with that platform's subscriptions and configurations.

Overview

A single-screen health read of the SCEF itself: its **status**, **product** name, **licence** state and the **Diameter host** it uses for T6a towards the MME. Beneath it is the **application context** — the SCS/AS ID field described above — and, once an SCS/AS ID is set, the count of **monitoring subscriptions** and **NIDD configurations** currently held for that Application Server. It is the quickest confirmation that the SCEF is up and how much a given platform is carrying. Monitoring and statistics dashboards are provided as Grafana embeds, reached from the same page.

Monitoring Events

The **monitoring event subscriptions** an Application Server has created for the current SCS/AS ID. Each subscription asks the network to report a class of event and POST it to the Application Server; each row shows:

- **Event type** — the monitoring type, such as **UE Reachability**, **Loss of Connectivity**, **Location Reporting**, **Availability after DDN Failure**, **Roaming Status** or **Number of UEs in an Area**.
- **Target** — the device or group the subscription watches, identified by an **External ID**, an **MSISDN** or an **External Group ID** for a whole fleet.
- **Notification destination** — the **callback URL** the SCEF POSTs each event report to; it must be an absolute `http(s)` address the Application Server exposes.
- **Max Reports** and, for a reachability subscription, the **reachability type** (`SMS` or `DATA`) — together with the monitoring **duration/expiry**, these bound how many events are sent and until when the subscription lives.

Subscriptions can be created, edited and deleted; deleting one stops the Application Server receiving those event reports.

NIDD

The **NIDD configurations** — the Non-IP Data Delivery bindings an Application Server has created for the current SCS/AS ID. A configuration authorises small non-IP payloads to be exchanged with a device over the control plane. Each row shows:

- **Target** — the device by **External ID**, **MSISDN** or **External Group ID**.
- **DNN / APN** — the Data Network Name (APN) the NIDD PDN connection uses (for example `nidd.iot.omnitouch`).
- **Status** — `ACTIVE`, or terminated (`TERMINATED`, `TERMINATED_UE_NOT_AUTHORIZED`).
- **Max Packet** — the **maximum packet size** in bytes the non-IP payload may reach.
- **RDS** — whether the **Reliable Data Service** is enabled, adding a lightweight acknowledged transport with source/destination ports over the non-IP link.

A configuration also carries a **PDN establishment option** that decides what happens when downlink data arrives while the device has no NIDD PDN

connection up — **wait for the UE** to establish one, **indicate an error**, or **establish the PDN connection** by paging the device.

Downlink data delivery

From a NIDD configuration you can **deliver a downlink payload** to the device. The payload is entered as text and **base64-encoded** before it is sent over NIDD, and each delivery can carry a **priority** and a **maximum latency**. Recent deliveries are listed with their **delivery status** — `SUCCESS`, `BUFFERING`, `SENDING`, or one of the failure reasons such as `FAILURE_UE_NOT_REACHABLE` — and a queued or buffered delivery can be cancelled. This is how a platform pushes a command or a firmware nudge to a constrained device that has no IP address.

NIDD Sessions

The **live T6a NIDD sessions** — the active Non-IP Data Delivery bindings the SCEF holds with the MME right now. Unlike the NIDD configurations, which are T8 resources scoped to an SCS/AS ID, these are **server-global runtime state**: they are **read-only** and are not scoped by SCS/AS ID, so this tab shows every active session regardless of which application context you are viewing. Each row shows the device's **IMSI**, the **APN** the NIDD PDN connection uses, the **EPS Bearer Identity (EBI)**, the session **state** (`ACTIVE`, `IDLE`), the **T6a session id**, the serving **MME host** and **MME realm**, and the **configuration id** the session was set up against. It is the fastest way to see which devices actually have a live non-IP binding, and through which MME.

Logs

The **Logs** tab tails the SCEF's live log output for troubleshooting a single instance without leaving OmniWeb.

Monitoring dashboards

T8 request rates, event-report throughput and NIDD delivery counts for the SCEF are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- **MME** — the SCEF reaches it over **T6a** to set up NIDD PDN connections and to gather the monitoring events reported here; the MME host and realm each session uses are shown on the NIDD Sessions tab.
- **HSS** — holds the subscription data (External Identifiers, NIDD authorisation) the SCEF's monitoring and NIDD services depend on.
- **PGW-C / SMF** and **UPF** — carry the ordinary IP data plane; NIDD deliberately bypasses them to move small non-IP payloads over the control plane instead.
- Operations Guide — back to the element index.

SGW-C — Serving Gateway (Control Plane)

The **SGW-C** is the control-plane half of the EPC Serving Gateway. It anchors each subscriber's bearers between the access network and the packet gateway: it terminates **S11** to the MME (session and bearer signalling), **S5/S8** to the PGW-C (the PDN connection), and programmes the user plane (SGW-U) that forwards traffic on **S1-U** to the eNodeB. In short, it is the node that holds the live bearer state for every attached subscriber and keeps the MME, PGW and user plane in step. OmniWeb surfaces the SGW-C's live sessions, session counts and logs; longer-run trends (throughput, session growth) are on the Grafana dashboards reached from the same page.

← Operations Guide

The SGW-C is a per-instance element page reached from the sidebar when one runs in the site's inventory, and — where several are deployed — a fleet view for aggregate status. All traffic goes through the OmniWeb backend proxy to the SGW-C's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

A single-screen health read: the **API status** of the SGW-C, the **active session count**, and a summary derived from the live sessions — the number of unique **MMEs** (distinct S11 peers), unique **PGWs** (distinct S5/S8 peers), and the **total bearers** currently anchored. It is the quickest way to confirm the SGW-C is up, how many sessions it holds, and how many MMEs and PGWs it is working with.

Sessions

Every anchored session, each row keyed by subscriber **IMSI** and **MSISDN** with its connection **status**. Opening a session shows its full bearer state: the **S11** control endpoints (MME-side and SGW-side F-TEIDs), each **PDN connection** (APN, default bearer, and the S5/S8 control F-TEIDs to the PGW), and each **bearer** (EBI, the S1-U and S5/S8 user-plane F-TEIDs, and its QoS). This is where you confirm a specific subscriber's bearers are established and see which MME and PGW are serving them.

Logs

The **Logs** tab tails the SGW-C's live log output for troubleshooting a single gateway without leaving OmniWeb.

Monitoring dashboards

Throughput, session-count trends and other time-series for the SGW-C are presented as Grafana dashboards, reached from the element page — OmniWeb

does not duplicate those charts natively.

Related

- MME — the control peer over **S11**; it drives session and bearer setup on the SGW-C.
- The **PGW-C** anchors the PDN connection over **S5/S8**.
- The **SGW-U / UPF** is the user plane the SGW-C programmes to forward traffic on S1-U.

SIM Card Tools — Filesystem, APDU & eSIM

Beyond discovering and presenting SIMs, OmniWeb gives you a set of **low-level card tools** that reach through any reader on any agent to talk to the card directly: browse and edit its file system, send raw APDUs by hand, watch the live APDU trace of a remote-SIM session, and manage eSIM profiles on an eUICC. This page is the operations guide for those tools.

← Operations Guide

These tools sit alongside — and reuse — the existing SIM features. Start here for the bigger picture:

- **SIM Bank** — the inventory: which SIMs are in which readers across every agent. Every tool on this page targets a reader from that inventory, and the SIM Bank's per-slot actions deep-link straight into them.
- **SIM Images (SAIP Profile Packages)** — saving a read-out file system as a portable image, and browsing/diffing saved images.
- **Remote SIM & Virtual SIM** — presenting a SIM (real or virtual) to a modem over the network via card emulators.
- **OTA SIM Management** — the same file-level editing, but reaching a card **over the air** with secured (03.48 / SCP80) packets instead of a local reader.

All four tools below live in the sidebar under the **SIMs** group.

One reader, one consumer at a time

A physical PC/SC reader can only be driven by one thing at a time. Whenever a tool on this page needs a reader, and something else already holds it — another operator reading it, or a live remote-SIM banking session presenting it to a modem — the tool shows **who holds it and why**, and offers an explicit **Take over**. There is no silent kick: taking over cancels the other session (and, for a banking session, disconnects that SIM↔cardem link). Editing/sending is a control action gated behind the SIM-bank control permission — without it you can view readers but not drive cards.

SIM Filesystem Explorer

Pick a reader and **read the whole file tree** of its card: the MF, DF.GSM / DF.TELECOM, ADF.USIM and ADF.ISIM and all their EFs, with every readable file decoded into named fields. A progress bar shows the walk ("read X of Y files — EF.…") since a full read touches many files.

- **Search** the tree by file code (FID), short name, or description — a search for "Emergency Call Codes" finds EF.ECC, not just `6FB7`.
- **PIN-protected files are flagged, never read** — the explorer will not burn PIN retries.
- **Edit an EF** (with the control permission): the detail pane offers a schema-driven form (decoded fields, typed) where available, falling back to raw hex. Writing is destructive and asks for confirmation. Where an EF needs authentication, use **Verify CHV/ADM** first — it tests a PIN/ADM key against the card (reporting remaining tries, never burning the last retry) and, once proven, reuses it for the write. Saved keys for a card's ICCID are offered automatically from the SIM key store.
- **Save image** snapshots the decoded tree as a named SIM Image; **Compare with image** diffs the live card against a saved one; **Saved**

images opens the image library. You can also open a saved image *as* a tree (`?image=<id>`) to browse a card you no longer have in a reader.

The reader must be free (not held by a remote-SIM session) — or you can Take over as above.

APDU Terminal

The APDU Terminal is a **hand console for the card**: type a raw APDU in hex, send it to the card in a chosen reader, and see the response decoded — the command's INS name, the status word and its meaning, and an expanded interpretation of any response data (an FCP TLV tree, decoded EF fields, or an AUTHENTICATE result). Canned buttons (SELECT MF, STATUS, READ BINARY, GET RESPONSE, …) fill common commands. Input is whitespace/colon-tolerant and validated (whole bytes, at least CLA/INS/P1/P2).

Two modes:

Mode	Behaviour	Use for
Single-shot	Each APDU is self-contained; the card resets to the MF between sends.	One-off probes.
Session	Opens a persistent, exclusive card session so state (selection, authentication) **persists** across APDUs — a SELECT then a READ BINARY works; a VERIFY shares its auth with a later UPDATE. The reader stays held until you close the session.	Stateful sequences.

The SIM Bank's per-slot **APDU** action deep-links here with that exact reader preselected. Same one-consumer/Take-over rules apply.

APDU Log

Where the terminal is you driving one card, the **APDU Log** is a passive **live trace** of the command/response APDUs exchanged between a **modem and a (remote) SIM** through the remote-SIM / cardem chain. Turn on **capture** and it tails the exchange: each row pairs a `modem→card` command with its decoded status word, plus the same response decoding as the terminal (FCP tree, EF fields, ATR/reset events). You can pause the view (capture keeps running), filter by command/hex/SW, and auto-scroll.

Use it to see what a modem is actually asking the SIM during an attach or an IR.38 test run — which files it selects, where a VERIFY fails, whether an AUTHENTICATE succeeds — without instrumenting the modem. It pairs naturally with Remote SIM, which sets up the modem↔SIM link the log then traces.

---

eSIM (LPA)

The **eSIM** tool manages an **eUICC** — a physical eSIM chip in one of your readers — through the agent's **Local Profile Assistant (LPA)**. An eUICC holds several profiles but only one is active at a time.

- **eUICC card** — shows the chip's **EID**.
- **Installed profiles** — every profile on the card: its state (active/disabled), nickname, provider, name, and ICCID. Per profile you can **enable** (make it the active eSIM), **disable**, **rename** (nickname), or **delete** (removes it from the eUICC — irreversible, so it confirms first).
- **Download a profile** — pull a new profile from an operator's SM-DP+ and install it onto the card, either from a whole **activation code** (`LPA:1$smdp.example.com$MATCHING-ID`) or from a split **SM-DP+ address + Matching ID**, with an optional confirmation code.

This is for eSIM (eUICC) cards specifically — for classic SIM/UICC file editing use the Filesystem Explorer above, and for editing a card **remotely** use OTA.

Related

- SIM Bank — the reader/SIM inventory these tools target.
- SIM Images — save, browse and diff read-out file systems.
- Remote SIM & Virtual SIM — present a SIM to a modem over the network (the link the APDU Log traces).
- OTA SIM Management — the same file editing, over the air.

SIM Images — SAIP Profile Packages

A **SIM Image** is a saved snapshot of a card's whole file system. OmniWeb stores each image as a **SAIP Unprotected Profile Package (UPP)** in DER encoding — the same interoperable format eUICC profiles use, defined by the TCA *eUICC Profile Package: Interoperable Format Technical Specification*. One captured card becomes one portable `.der` file that any SAIP-aware tool can read.

This page is the operations guide for capturing, exporting, uploading, browsing and diffing SIM images, and for using a saved image to back a virtual SIM. The live reader inventory is covered in SIM Bank.

Table of Contents

Why SAIP UPP

Earlier builds stored a SIM image as a decoded JSON tree used only for diffing. That format was OmniWeb-private and could not be consumed by any external

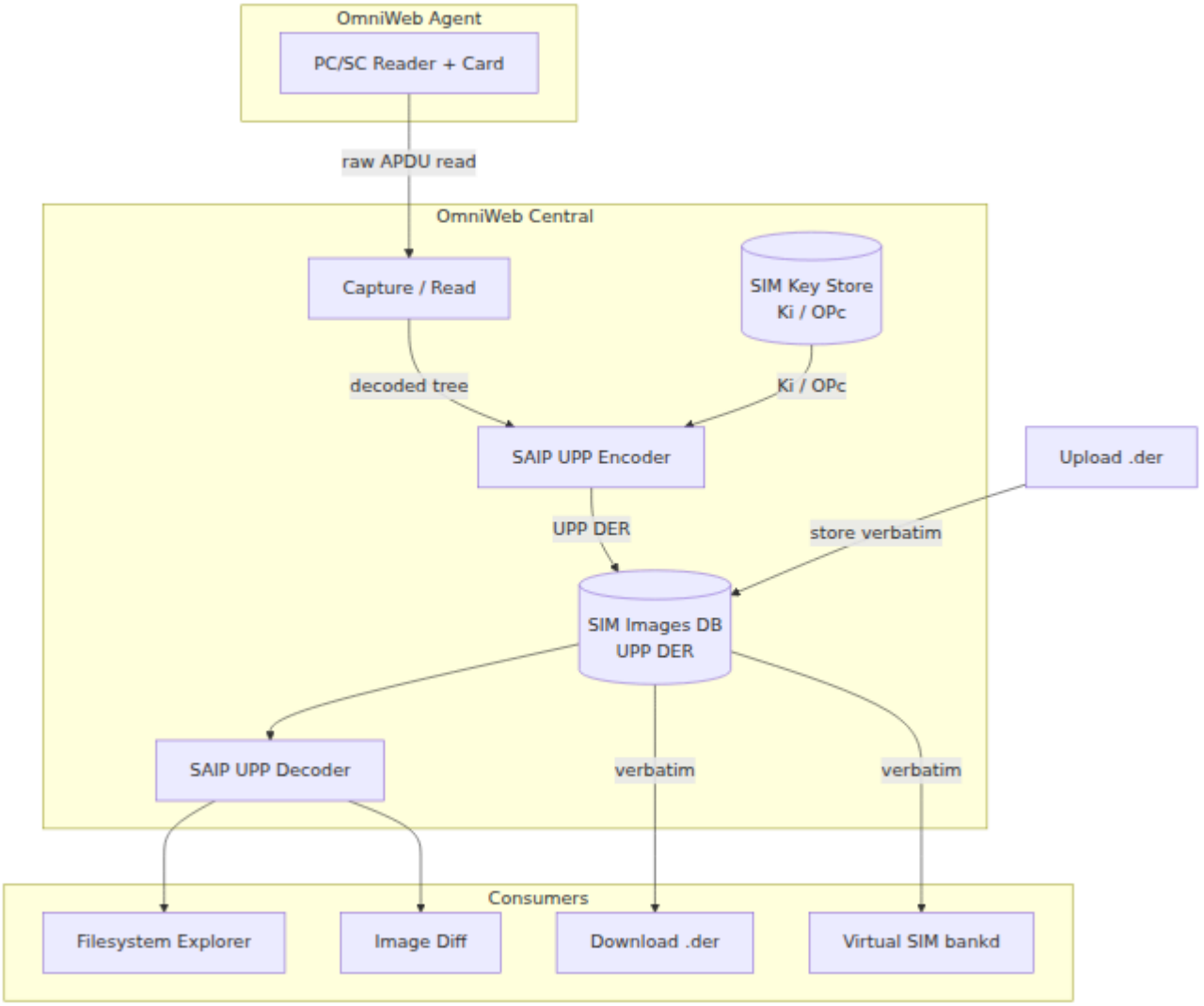
tool.

Storing the image as a **SAIP UPP** instead means:

- **Portability** — the exported `.der` is a standard profile package. It loads in SAIP profile validators and any SAIP-compliant toolchain.
- **One format everywhere** — the same `.der` an operator downloads is what backs a virtual SIM and what the file-system explorer renders. There is no second, divergent representation.
- **Reprovisioning** — when the card's authentication keys are known (see Capturing an Image), the UPP includes a `PE-AKAParameter` element carrying Ki and OPc, so the package describes a complete, installable profile rather than a read-only dump.

The package is a raw concatenation of DER-encoded Profile Elements; it is **not** encrypted (it is an *Unprotected* Profile Package). Treat an exported `.der` with the same care as any file holding Ki/OPc — see Troubleshooting.

Architecture Overview



Capture reads the card, the encoder produces a UPP (folding in Ki/OPc from the SIM Key Store if available), and the package is stored. Every consumer reads back from that single stored package — the explorer and diff decode it to a tree on demand; download and the virtual-SIM bankd use the bytes verbatim.

What's in an Image

A captured image holds, for the card's standard MF / ADF.USIM layout:

Element	SAIP Profile Element	Contents
File system	`PE-GenericFileManagement`	Every reachable EF: its File Control Parameters (FCP) and raw content (transparent EFs) or every record (linear/cyclic EFs)
Identity	`PE-Header`	The card's ICCID
Authentication	`PE-AKAParameter`	Ki and OPc for Milenage — **only when the card's keys are in the SIM Key Store**

Files are created with `PE-GenericFileManagement` (one operation group per DF) rather than the standardised SAIP file-system templates: a card read recovers raw bytes, not the template a profile creator originally used, so the generic representation is always exact.

Capturing an Image

Capture happens from the SIM Filesystem explorer or the SIM Bank slot actions:

1. Read a card's file system in the explorer.
2. Choose **Save image** and give it a label.

The decoded file system is encoded to a UPP and stored. If a SIM Key Store entry exists for the card's ICCID with both **Ki** and **OPc**, those keys are written into a `PE-AKAParameter` so the resulting package is fully installable. Without a key-store entry the package still captures the complete file system; only the authentication element is omitted.

Identity columns (ICCID, IMSI, card type, file count) are derived from the file system at capture time so the image list and diff header never have to re-decode the package.

Exporting an Image

Each row in the saved-images list has a **download** action that serves the stored package verbatim as `UPP_<iccid>.der`.

The downloaded `.der` is a standard SAIP UPP and can be inspected with any SAIP-compliant profile tool.

How it works: The file is the exact bytes stored at capture (or upload), with no re-encoding on download, so a round-trip download is byte-identical to what is stored.

Uploading an Image

The saved-images page has an **Upload .der** action that accepts a SAIP UPP file and stores it as a new image. Use it to bring in profiles generated elsewhere — for example the GSMA TS.48 *Generic eUICC Test Profile* packages, or a UPP produced by another SAIP tool.

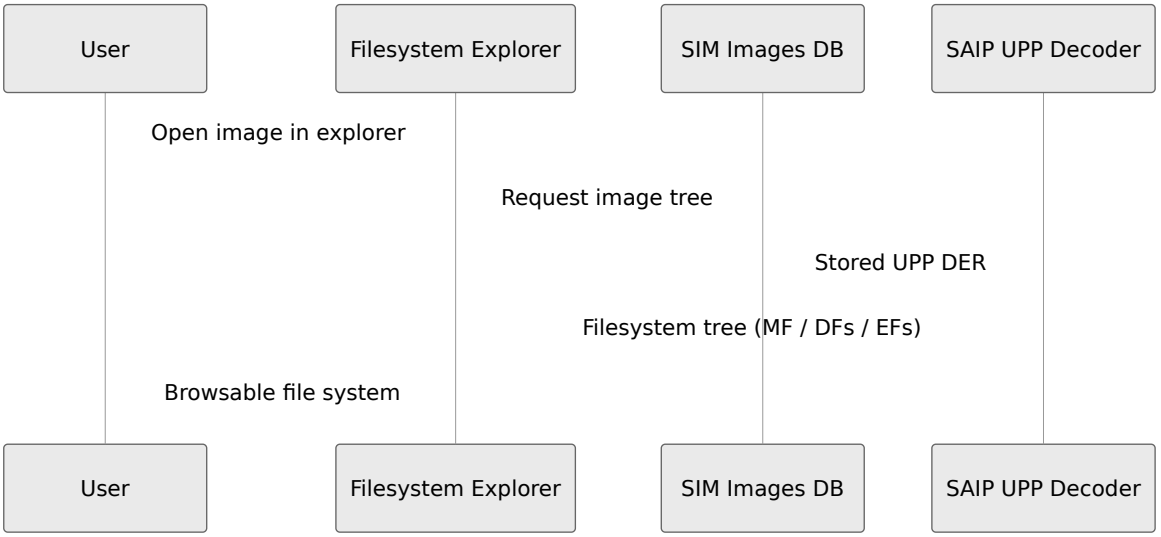
On upload, OmniWeb:

1. Validates the file begins with a SAIP Profile Element tag (a context-class constructed tag — `A0`, `A1` or `B0`).
2. Stores the package verbatim.
3. Decodes the `PE-Header` to populate the ICCID column where present.

An uploaded image is a first-class image: it can be downloaded, browsed in the explorer, diffed, and used to back a virtual SIM exactly like a captured one.

Browsing an Image in the Explorer

Every saved image has a **Browse in filesystem explorer** action. It opens the SIM Filesystem explorer against the stored package — **no physical card and no reader are needed**. The package is decoded back into the same file-system tree the explorer renders for a live card, so you can inspect every EF's raw content offline.



Diffing Two Images

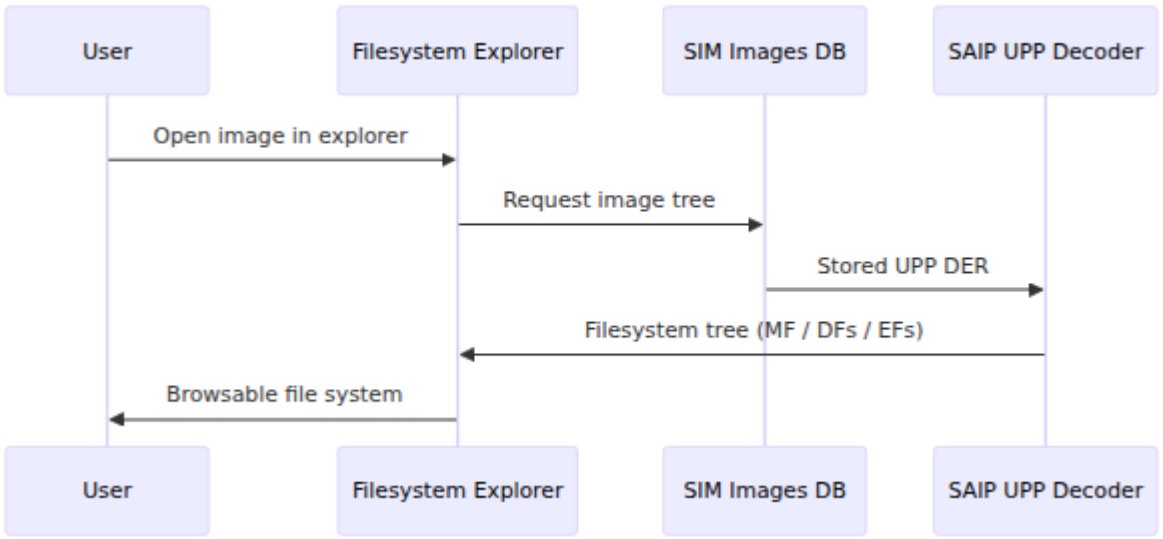
The saved-images page lets you pick two images and diff them. Both packages are decoded to file-system trees and aligned by EF path, so the comparison is file-by-file regardless of capture order:

Status	Meaning
`same`	EF present in both with identical content
`changed`	EF present in both, content differs
`only_a`	EF present only in the first image
`only_b`	EF present only in the second image

For a **changed** EF the diff reports the differing byte ranges (and per-record deltas for record-structured files), so you can see exactly which bytes moved — e.g. a test SIM before and after a personalisation change, or two operators' SIMs side by side.

Backing a Virtual SIM with an Image

A saved image can stand in for a physical card when presenting a virtual SIM. In the virtual-SIM composer, saved images appear alongside the captured rspro images as selectable file-system sources. Pair one with a subscriber identity (from the HSS or the SIM Key Store) and OmniWeb launches a virtual `bankd` backed by that image's file system, with the subscriber's Ki/OPc overlaid for authentication.



The file system the modem reads is replayed from the image; the identity and keys are overlaid from the chosen subscriber, so one image can back many subscribers.

Profile Element Reference

The UPP is a sequence of Profile Elements, each a DER TLV whose context-class tag identifies its type. OmniWeb writes and reads the subset below; other elements in an uploaded package (PIN, PUK, Security Domain, file-system templates) are preserved on storage and skipped when decoding to a file-system tree.

Tag	Profile Element	Used by OmniWeb
`A0`	`PE-Header`	ICCID; profile metadata
`A1`	`PE-GenericFileManagement`	The file system (FCP + content)
`A4`	`PE-AKAParameter`	Ki and OPc (Milenage)
`AA`	`PE-End`	End-of-package marker
`B0`	`PE-MF`	(Templated MF — read on upload, not emitted)

For the full element catalogue and encoding rules, see the TCA eUICC Profile Package: Interoperable Format Technical Specification.

Troubleshooting

Exported image has no authentication keys

Symptoms: A downloaded `.der` has no `PE-AKAParameter`; a virtual SIM backed by the image cannot authenticate without keys supplied separately.

Possible causes:

- No SIM Key Store entry existed for the card's ICCID at capture time.
- The key-store entry was missing Ki or OPc.

Resolution:

1. Add the card's Ki and OPc to the SIM Key Store, keyed by its ICCID.
2. Re-capture the image — the new capture folds the keys into a `PE-AKAParameter`.
3. Alternatively, supply the keys at virtual-SIM presentation time (from the HSS or an explicit override).

Upload rejected as "does not look like a SAIP UPP"

Symptoms: Uploading a file returns a validation error.

Possible causes:

- The file is not a SAIP UPP — it does not begin with a Profile Element tag (`A0`, `A1` or `B0`).
- The file is a *protected* profile package (encrypted), not an *unprotected* one.

Resolution:

1. Confirm the file is an **Unprotected** Profile Package in DER encoding.
2. Verify it begins with a `PE-Header` (`A0`) — most well-formed UPPs do.
3. If the package is protected/encrypted, it cannot be stored or browsed; only unprotected packages are supported.

Browsing an uploaded image shows files by FID, not by name

Symptoms: EFs in the explorer are labelled by file identifier (e.g. `EF.6FXX`) rather than a friendly name.

Possible causes:

- A card read recovers file identifiers and raw content, not the symbolic names a profile creator used. OmniWeb maps the well-known USIM EFs to friendly names; any EF outside that map is shown by its FID.

Resolution: No action needed — the FID is the authoritative identifier and the raw content is exact. This is expected for non-standard or vendor-specific files.

Handling exported packages safely

An exported UPP is **unencrypted** and, when keys were available at capture, contains Ki and OPc. Treat a downloaded `.der` as sensitive key material:

- Do not send it over untrusted channels.
- Delete local copies when no longer needed.
- Prefer keeping images inside OmniWeb (where access is gated by the SIM Bank permission) over distributing `.der` files.

SIM Bank — distributed SIM inventory

The **SIM Bank** shows every SIM in every smart-card reader across every OmniWeb agent. Each agent scans its local PC/SC readers, reads each SIM's identity, and reports the inventory to central exactly like it already reports modems/phones as devices. Central aggregates the inventory per agent and persists a database of SIMs keyed by ICCID, so the "what SIMs are where" view survives reader and agent disconnects.

This is the inventory half of a distributed remote-SIM testbed. The osmo-remsim card-emulation/banking layer that lets you **connect any SIM to any cardem over the network** is documented separately in Remote SIM (osmo-remsim); this page covers discovery and inventory. Saving, exporting and reusing a card's file system as a portable profile package is covered in SIM Images (SAIP Profile Packages).

The page is **reader-centric**: one row per physical reader slot, grouped by agent, so empty, unreadable and in-use slots all show — the bank layout is the truth, not just the populated slots.

How the agent reads a SIM

The agent scanner mirrors the modem scanner: enumeration is slow, so results are cached and rescanned on an interval.

For each PC/SC reader on the host:

1. **Enumerate** the PC/SC readers attached to the host.
2. **Probe** each reader with an *exclusive* connect.
   - No card → `present=false`.
   - Connect raises a **sharing violation** → the reader is held by another process (e.g. an osmo-remsim `bankd` or a card-emulation session) → `present=true, in_use=true`, and **no read is attempted**. A busy reader never crashes the scan.
   - Card present and free → read it.
3. **Read** the SIM identity with a self-contained set of raw APDUs:
   - UICC-vs-classic SIM detection (select `3F00`; `6E00` → classic GSM, so use `CLA=A0` and classic selection mode),
   - EF.ICCID (`2FE2`), EF.IMSI (`6F07`), EF.SPN (`6F46`),
   - `EF.AD` (`6FAD`) for the MNC length (so MCC/MNC are split correctly),
   - a **safe** default-PIN (`0000`) VERIFY on a locked card — it probes the retry counter first (a non-decrementing empty VERIFY) and **refuses** if only one retry remains, so it **never** burns the last retry or touches the PUK (a SIM bank must not brick cards),
   - the card ATR.

Each reader becomes a dict:

```
{
  "reader_name": "Identiv SCR35xx USB Smart Card Reader …",
  "reader_index": 0,
  "present": true,
  "in_use": false,
  "iccid": "8961…", "imsi": "505…", "mcc": "505", "mnc": "01",
  "country": "Australia", "spn": "Telstra", "atr": "3b9f96…"
}
```

The list is sent to central as a `sims` message (alongside the device list) on register and whenever it changes; a `simbank_rescan` command forces an immediate uncached scan.

Card-reading capability

A host reads SIMs through its PC/SC smart-card stack. Where deeper card decoding is needed, the agent uses an optional, separately-licensed smart-card library kept isolated from the rest of the agent (it is not merged into OmniWeb's own code). A host **without** the smart-card stack still runs the agent fine and simply reports no SIMs — the SIM bank is an optional per-host capability, like modems.

The reference reader is an SCM/Identiv SCR35xx. Multiple readers per host are supported.

Inventory → central → UI flow

```
agent host                          central (android-role process)
browser
──────────                          ──────────────────────────────
────────
simbank.list_sims()  ──sims──▶     android agent_socket
  (PC/SC read)                       └─
registry.update_sims(agent_id, …)   GET /api/simbank/sims
                                     └─ simbank.record_sims(…)  ─▶
DB       GET /api/simbank/readers

POST /api/simbank/agents/<id>/rescan
                      ◀─simbank_rescan──  (forces an uncached
scan)
```

- The agent dials **out** to central over the same WebSocket it uses for phones (`/api/android/agent`). SIM inventory rides that connection — there is no new transport.

- Central holds the **live** per-agent reader snapshot in the in-memory `android.Registry` (`update_sims` / `all_sims`), and mirrors each ICCID into the persistent `simbank_sims` table via `simbank.record_sims`.
- Because this depends on the single-process agent Registry, the SIM-bank API is served **only by the agent-role backend process** (same constraint as Test Devices).

REST API (`/api/simbank`, JWT, RBAC element type `simbank`)

Method	Route	Purpose
GET	`/sims`	Every SIM across all agents (incl. offline persisted-only), with live present/in-use overlaid.
GET	`/readers`	Every PC/SC reader on every **connected** agent (live), incl. empty and in-use readers.
GET	`/sims/<iccid>`	One SIM.
PATCH	`/sims/<iccid>`	Edit operator-assigned fields (notes today). Creates the row if the ICCID is unseen.
DELETE	`/sims/<iccid>`	Forget a SIM (drops the persisted record; reappears on next scan if still inserted).
POST	`/agents/<id>/rescan`	Force an immediate uncached PC/SC rescan on one agent and return its fresh inventory.

RBAC mirrors `ir38`/`android-control`: admins always pass; otherwise a JWT permission with `element_type == "simbank"` grants view, and `POST` in its methods grants control.

Frontend

`src/pages/simbank/SimBank.tsx` — a searchable, filterable table of all SIMs: status (present / in use / offline), agent/site, reader, ICCID, IMSI, MCC/MNC, country, SPN, editable notes, last seen. Per-agent **Rescan** button. Routed at `/simbank` and linked from the sidebar under **Roaming Tests › SIM Bank**.

Database schema

`simbank_sims` — one row per ICCID ever seen:

Column	Notes
`iccid` (PK)	SIM serial.
`imsi`, `mcc`, `mnc`, `country`, `spn`, `atr`	Last-known identity (only overwritten by non-empty reads).
`notes`	Operator-assigned, editable.
`last_agent_id`, `last_site`, `last_reader_name`, `last_reader_index`	Where it was last (or is currently) seen.
`first_seen`, `last_seen`	Timestamps (UTC).
`remsim_bank_id`, `remsim_slot_nr`, `remsim_mapped_cardem`	**Phase-2 reserved** (nullable, unused).

Empty and in-use readers carry **no** ICCID, so they are not persisted as SIMs — they only appear in the live `/readers` snapshot.

Future: osmo-remsim

Phase 2 maps a physical reader/SIM into an osmo-remsim bank so a SIM in one site can be presented to a modem/phone in another (remote SIM). The hooks are already in place — search the code for `# TODO remsim`:

- `simbank_sims.remsim_bank_id` / `remsim_slot_nr` / `remsim_mapped_cardem` (nullable columns, surfaced in `to_dict` and the TS `SimRow` type as `null`),
- the agent's per-reader `in_use` flag already distinguishes a reader held by a `bankd`/emulation session — that's the reader phase 2 will map,
- `PATCH /api/simbank/sims/<iccid>` is where editing the remsim mapping will be wired.

Nothing here drives remsim today; the shape is reserved so the mapping slots in without reworking the inventory hot path.

Site Editor — Site Configuration Viewer

The **Site Editor** shows the raw configuration file for the currently selected site — the YAML host/inventory file that defines that deployment's network functions, infrastructure, and settings. It is a read-only, syntax-highlighted view, so you can see exactly what OmniWeb is working from for a site without opening a shell on a host.

← Operations Guide

What it shows

OmniWeb drives every site from a per-site configuration file. That file lists the site's elements (which NFs are present and at what addresses), its infrastructure hosts, its monitoring role, its `fivegc` block (see 5G Core), and so on — everything the sidebar, topology, element pages, and dashboards derive from.

The Site Editor loads that file for the site chosen in the top-bar site dropdown and displays it in a full editor pane (Monaco, the same editor VS Code uses) with YAML highlighting, line numbers, folding, and a minimap. The header shows the customer, the site, and the source filename.

Parse error on line 3: ... SE -->|/api/sites/{customer}/{site}/ra ----------
--^ Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND',
'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND',
'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got
'DIAMOND_START'

Try again

If no site is selected, the page prompts you to pick one from the dropdown first.

Read-only by design

The Site Editor is a **viewer** — the pane is read-only (marked with a **read-only** chip in the header). It is there to *inspect* the effective configuration, not to change it.

Site configuration is not edited from the browser. As noted throughout this guide, OmniWeb deployment and configuration is managed via **Ansible** — do not edit configuration files directly on hosts, and adding or changing an element is a configuration change made through that pipeline (see the configuration model). The Site Editor lets you read the result of that pipeline for the selected site.

When to use it

- **Confirm what a site is configured with** — which elements, hosts, and 5GC NFs OmniWeb thinks are present, and at which addresses.
- **Troubleshoot the UI against the config** — if an element, a sidebar entry, or a 5GC section is missing or pointing at the wrong host, the raw file is the source of truth to check against.
- **Reference the exact filename and contents** when preparing an Ansible change.

Related

- Architecture — the configuration model — how site config drives OmniWeb and how it is managed.
- 5G Core — the `fivegc` block of the site config and what it surfaces.
- Network Topology — the topology and Host List views built from the same site configuration.

SMPP Gateway

The **SMPP Gateway** bridges external SMPP peers to the messaging core. External applications and interworking partners — **ESMEs** (External Short Message Entities) — connect over SMPP either as clients the gateway dials out to or as servers the gateway accepts binds from, and the gateway carries their short messages to and from the SMSC. Each connection is an SMPP **bind** (transceiver, transmitter or receiver), and the gateway's job is to keep those binds up, authenticate and police them, and hand traffic on to the messaging core. OmniWeb surfaces the live bind status, the configured client and server peers, and the gateway's logs.

← Operations Guide

The SMPP Gateway is a per-instance element page reached from the sidebar when one runs in the site's inventory, with a fleet view where several are deployed. All traffic goes through the OmniWeb backend proxy to the gateway's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

A single-screen health read: the aggregate bind statistics — the **total** number of binds and the **connected / disconnected** breakdown split across **client** (outbound) and **server** (inbound) binds. It is the quickest way to confirm the gateway is up and that its peers are bound, and to spot at a glance if any expected connection has dropped.

Bind Status

Every enabled bind — client and server — with its current **connection status**, the underlying **session details** and per-bind **metrics**. This is the live view of who is bound right now. From here an operator can act on a bind:

- **Drop** an active bind — for a client bind this terminates the outbound connection; for a server bind it disconnects all active client connections. Use it to clear a stuck or misbehaving connection.
- **Reconnect** a client bind — forces a client (outbound) bind to redial its remote host, which is the quickest way to bring an outbound peer back after a network blip.

Client Peers

The configured outbound SMPP **client** connections — the peers the gateway dials out to. Each shows its **name**, the remote **host** and **port**, the **system ID** it binds with, and the **bind type** (transceiver, transmitter or receiver). Client peers are fully editable here: an operator can **add** a peer, **edit** an existing one (host, port, credentials, bind type, enquire-link interval, TPS limit and the other bind parameters), and **delete** one that is no longer needed. Enabling a peer brings its outbound bind into service; disabling it takes the peer out without discarding its configuration.

Server Peers

The configured inbound SMPP **server** authorisations — the credentials and rules for peers that bind *into* the gateway. Each entry defines a **system ID** and password, the **allowed bind types**, and the guard rails that police an inbound bind: an **IP whitelist** of permitted source addresses and a **source-address whitelist** of permitted originating-address patterns, plus a **TPS limit**. Server peers are editable here in the same way — **add**, **edit** and **delete** — so an operator can onboard a new inbound partner, tighten its whitelists, or revoke it. Enabling and disabling an authorisation controls whether that partner is allowed to bind.

Logs

The **Logs** tab tails the gateway's live log output for troubleshooting a single gateway — bind failures, authentication rejects and dropped connections — without leaving OmniWeb.

Monitoring

Message rates, throughput and other time-series for the SMPP Gateway are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- The **SMSC** is the messaging core the gateway hands short messages to and from.
- The **IP-SM-GW** carries SMS over IMS for VoLTE devices, complementing the SMPP path to external peers.

SMSC — Short Message Service Centre

The **SMSC** (Short Message Service Centre) is the store-and-forward heart of SMS in the core. It **accepts** messages from the network — over SS7 (MAP) from the mobile side, over Diameter, and over SMPP from application front-ends — then **routes**, **translates** and **delivers** them, holding on to anything it cannot deliver immediately and retrying until the message is delivered, expires or is dropped. Along the way it tracks the delivery state of every message, translates calling and called numbers, federates with peer SMSCs so a message can be handed to whichever node can reach the recipient, and drives bulk SMS **campaigns**. OmniWeb surfaces all of this — live message state, subscriber locations, front-end registrations, Diameter and federation health, the routing and translation rule sets, and campaign management — from a single per-instance page.

← Operations Guide

The SMSC is a per-instance element page reached from the sidebar when one runs in the site's inventory, with a fleet view where several are deployed. All traffic goes through the OmniWeb backend proxy to the SMSC's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

The landing tab is a single-screen health read for the node: current message counts and delivery state, front-end and peer health at a glance, and a **search by MSISDN** that looks across both stored messages and subscriber locations — the quickest way to answer "what is happening to this subscriber's SMS right now" from one box.

Messages

Every message the SMSC is holding or has processed, each row showing the sending and receiving numbers, the message **state** (pending, delivered, failed, expired, dropped or auto-replied), the number of **delivery attempts**, and its age. Messages can be filtered by state, by **SMSC**, and to include **dead-letter** or **unrouted** traffic, and searched by MSISDN, node name or message ID. Selecting a message opens its detail — including the decoded plaintext — and an operator can **reroute** a message to force it back through routing, for example after fixing a routing rule.

Frontends

The application and SMPP **front-ends** currently registered with the SMSC — the clients that submit and receive messages. Each entry shows its hostname, when it was **last seen**, and when its registration **expires**, so you can confirm the message-handling front-ends are connected and their registrations are live.

Locations

The **subscriber locations** the SMSC has learned — the last known serving location for each subscriber by **IMSI** and **MSISDN**. This is the information the SMSC uses to reach a recipient, and it can be searched by IMSI or MSISDN and filtered to include expired entries. It is where you confirm the SMSC knows where a subscriber is before it can deliver to them.

Diameter

The Diameter peer connections the SMSC uses for messaging, each row showing the peer's **realm**, **IP address**, **transport**, connection-initiation direction, reported **product name**, negotiated **application IDs** and **connection state**. This is where you confirm the Diameter path the SMSC depends on is up.

Federation

The **federation** view shows how this SMSC cooperates with peer SMSCs so a message can be handed to whichever node can reach the recipient. It reports **this node**, the count of **healthy**, **degraded** and **unhealthy** peers, the number of **cached federated messages** (highlighted when non-zero), the **discovery** method in use (DNS domain or static peers), and a per-peer table of node name, status, recent failures, last successful health check and the front-ends each peer offers. It is the fastest way to see whether the SMSCs in a federation are talking to each other.

Routing

The **SMS routing rules** that decide where each message goes, listed by **priority**. Each rule matches on a **calling** and **called** number pattern (and optionally a source SMSC or source type such as SS7 or circuit-switched), and resolves to a **destination** — a target SMSC or an **ENUM** lookup — with a description and an enabled state. Rules are fully editable: an operator can add, edit, enable, disable or delete them, and the rule set can be exported and re-imported.

Translations

The **number-translation rules** that rewrite calling and called numbers as messages pass through — normalising formats, applying prefixes and the like. Each rule is listed by **priority** with its calling and called match, optional source SMSC, a **continue** flag (so several rules can apply in sequence), a description and an enabled state. As with routing, the rules are editable — add, edit, enable, disable, delete — and can be exported and re-imported.

Campaigns

Bulk SMS **campaigns** are created and driven from this tab. A **New Campaign** is defined with a name and description, a **sender** number, the source SMSC, a **recipient list**, an **audience** (active recipients only or everyone in the list), a **drip** rate in messages per second, a message **validity** window, an optional **deliver-after** time and the **message template**. Once created, a campaign is controlled from here — **start**, **pause**, **resume** or **cancel** — and its **targets** can be inspected to see the per-recipient send and delivery state as the campaign progresses.

Campaign Lists

The recipient **lists** that campaigns send to. Lists are fully managed here — create a **New Recipient List** with a name, description and an initial set of recipients, **append** more recipients later, edit, or delete a list. Recipients are supplied as **CSV**, so an existing contact list can be imported directly rather than entered by hand.

NAPTR Test

The **NAPTR Test** tab is an operator tool for resolving an **E.164 number (or domain) through ENUM/NAPTR DNS** to see how the SMSC will route a message towards it. You enter the phone number and the **ENUM domain** to query (`e164.arpa` by default) and submit; nothing is fetched until you do. The SMSC reverses the number's digits into the ENUM zone, looks up the **NAPTR** records published there, and returns them in **order** and **preference** — each with its **flags**, **service**, the rewrite **regexp** and **replacement**, and the resolved **result domain** the record points the message at. It is where you confirm the ENUM data behind a routing rule really resolves a number to the URI or route you expect, before you rely on it in production.

MWI

The **MWI** tab sends a **Message Waiting Indication** update for a subscriber — the notification that sets or clears the "waiting messages" flag on a handset, typically after a voicemail is left or collected. You give the **destination MSISDN** to notify and the **source MSISDN** it comes from, pick the **indication type** (voicemail, fax, email or other), and set whether the indicator should be **active** (on) or cleared (off), with an optional **message body** and originating **SMSC**. It is a submit form: the update is only sent when you post it, making it the direct way to raise or clear a subscriber's waiting-message indicator by hand.

System Logs

The **System Logs** tab is the SMSC's own API-backed log view. It can be filtered by **level** (debug, info, warning, error) and by a **limit** on the number of lines returned, so you can pull just the recent errors or a fuller trace when investigating a specific problem — without leaving OmniWeb.

More tools

Alongside the tabs above, the SMSC page offers a few focused tools:

- **Submit** — a form to send a one-off SMS directly from OmniWeb (from, to and message text), for testing delivery to a specific recipient.
- **Operations** — node-wide maintenance actions, including clearing the registered **front-ends**, stored **messages**, **routing** rules and **translation** rules. Each is a destructive action and asks for confirmation first.

Logs

The **Logs** tab tails the SMSC's live log output for troubleshooting a single node without leaving OmniWeb. For a filtered, API-backed view of the same node, use **System Logs** above.

Monitoring

Message throughput, delivery-success rates, queue depth and other time-series for the SMSC are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- **HSS** — holds the subscriber data and routing information the SMSC's deliveries resolve against.
- The **DRA** — relays the Diameter signalling the SMSC uses for messaging.
- **STP** — the SS7 signalling path over which the SMSC exchanges MAP messages with the mobile network.
- The **CSCF/TAS** — the IMS side that originates and terminates messaging for VoLTE subscribers.

Software Management & Lifecycle

OmniWeb provides a single place to see and manage the software running across the fleet — package versions, available updates, and licensing — so the lifecycle of the network's software is visible from the same portal used to operate it.

← Back to Operations Guide

Software Page (APT Repository)

The **Software** page is an APT repository browser for Omnitouch packages. It connects to the configured APT repository server and shows, for the packages it manages:

- **Available packages** in the repository.
- **Versions** offered, including the latest available.
- **Update status** — whether a newer version exists than what is deployed.

This lets operators see at a glance which Omnitouch components have updates pending and which versions are current, providing the inventory side of the software lifecycle. Package installation/upgrade on hosts is performed through the standard APT/`dpkg` tooling (and is typically automated via Ansible); OmniWeb gives the visibility layer over what is published and current.

The Software page — each Omnitouch package with its latest repository and cache versions and an in-sync indicator that flags where a newer version is published than is cached/deployed (here `omni-upf` and `omni-msc` are out of sync).

Syncing Packages to the Cache

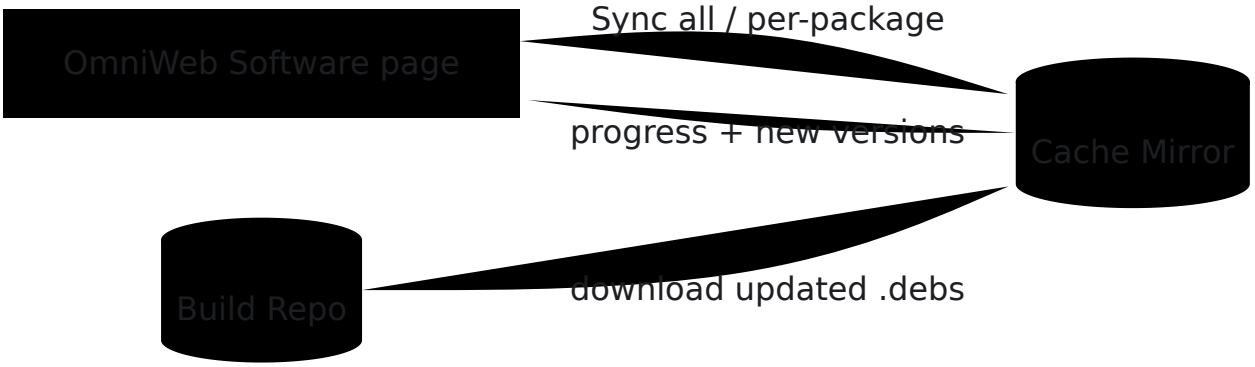
Deployments that run an on-site **APT cache mirror** (for airgapped or bandwidth-limited sites) keep a local copy of the packages published on the build repository. Over time the cache falls behind — the Software page flags each package that is **out of sync** (a newer version is published than is cached). From the same page you can pull those updates into the cache without leaving OmniWeb:

- **Sync all** — pull every updated package from the repo into the cache.
- **Sync** (per row) — pull just one package. Click several and they **queue**: they download one after another, and the package index is regenerated

once at the end (not per package), so the cache is never left half-updated.

While a sync runs, the page shows live progress — which package is downloading, how many are queued, and a streaming log — and refreshes the version comparison when it finishes.

A sync in progress — `omni-msc` downloading with `omni-upf` queued, the phase banner, and the live log. When the queue drains, the cache index is regenerated once and the in-sync status updates.



The sync runs on the cache mirror itself (OmniWeb triggers it and streams the progress); the mirror pulls the updated `.deb`s from the build repo and

regenerates its index. Hosts at the site then install the updated packages from the cache as usual.

Versioning

OmniWeb reports its own build version (version, commit, and build date) so operators can confirm exactly which build of the portal they are running. This is the reference point when reporting issues or coordinating an upgrade of the portal itself.

License Management

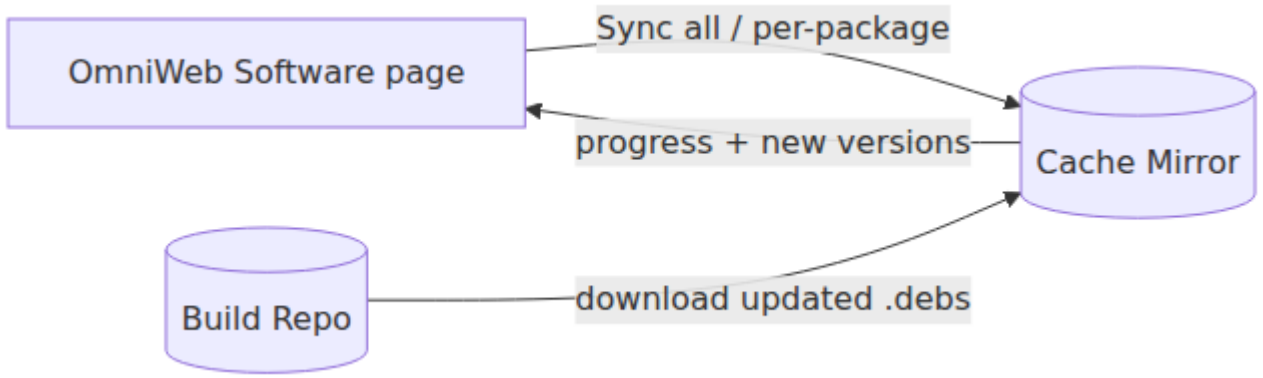
The **License Server** is administered through OmniWeb like any other element, using the same generic patterns (see Common Operations). It surfaces:

License Server overview — current license status and validity, including trusted-time state.

View	Description
Overview	Current license status, validity dates, and feature entitlements.
Request Logs	Historical license requests, with filtering — useful for auditing what was checked and when.
Query	Look up the license status for a specific licensee/product on demand.

Licensing ties into the software lifecycle: entitlements determine which products and features a deployment is authorised to run, and the request logs give an audit trail of license activity over time.

Lifecycle at a Glance



1. **See what's available** — the Software page shows pending updates.
2. **Confirm entitlements** — the License Server shows what the deployment is licensed for.
3. **Apply** — upgrades run via APT/Ansible on the hosts.
4. **Verify** — confirm versions, then watch element Logs and Dashboards to confirm a healthy result.

SS7 — Signalling Stack (STP / HLR / IP-SM-GW / CAMEL-GW)

The **SS7 stack** (OmniSS7) terminates SIGTRAN transport — SCTP associations carrying **M3UA** — and routes SS7/MAP traffic across the signalling network. A single service provides the whole stack, and OmniWeb presents it as four element views for its different roles: **STP** (Signalling Transfer Point, the SS7 router), **HLR** (Home Location Register), **IP-SM-GW** (IP Short Message Gateway) and **CAMEL-GW** (CAMEL Gateway). All four are backed by the same shared client, so the SIGTRAN transport and routing they share is the same underlying stack seen from different angles.

← Operations Guide

Each view is a per-instance element page reached from the sidebar when an SS7 stack runs in the site's inventory. All traffic goes through the OmniWeb backend proxy to the stack's API, so it stays behind the single authenticated gateway; responses come back in a `{status, response}` envelope. The documentation link in each element page header (the in-context help button) opens this page. Each view is organised into tabs. Message rates and other time-series are presented as Grafana dashboards reached from the element page — OmniWeb does not duplicate those charts natively.

The stack at a glance

The SIGTRAN transport underneath every role is the same, so the **Peers** and **SCTP Connections** tabs appear in all four views and read the same data.

The **Peers** tab reports M3UA peer status (`/api/m3ua-status`): every M3UA/M2PA peer, with its **ASP state**, **association state**, local and remote **endpoints**, **routing context**, protocol and mode, and per-peer **uptime** statistics. This is where you confirm each signalling neighbour's application-server process is up and in service.

The **SCTP Connections** tab lists the underlying transport (`/api/sctp-connections`): every SCTP association with its **status**, **role**, local and remote **IP/port**, inbound and outbound **stream** counts, **bytes** and **messages** sent and received, **last activity**, and any last error — plus aggregate totals across all connections. It is the layer to check when a peer shows down: whether the SCTP association itself is established.

STP — routing

The **STP** view is the SS7 router. Alongside the shared transport tabs it carries the **Routing** tab, plus **Network Map**, **Overview** and an **SS7 Events** feed.

The **Routing** tab shows the full M3UA routing configuration (`/api/routing`): the configured **peers**, the **point-code (PC) routes** and the **global-title (GT) routes**.

- **Point-code routes** direct traffic to a destination point code (`dest_pc`) via a target **peer**, with a **network indicator**, point-code **mask** and **priority**. Operators can **add** a PC route (`POST /api/routing/pc-routes`, giving the destination point code and target peer) and **delete** one (`DELETE /api/routing/pc-routes/{dest_pc}-{peer_id}`).
- **Global-title routes** match a GT **prefix** and forward to a target **peer** with a **priority** and optional SSN, translation-type and nature-of-address details and a description. Operators can **add** a GT route (`POST /api/routing/gt-routes`, giving the prefix and target peer) and **delete** one (`DELETE /api/routing/gt-routes/{prefix}:{peer_id}`).

This is where you confirm, and adjust, how the STP forwards SS7 messages toward each destination.

HLR

The **HLR** view surfaces the register's active subscribers and the link to the HLR back-end, alongside the shared transport tabs, an **Overview** and an **SS7 Client** tab.

The **Subscribers** tab lists the active subscriber register (`/api/subscribers`): every subscriber who has sent an **UpdateLocation** and not yet a **CancelLocation**, each row showing **IMSI**, **VLR**, serving **MSC**, registration time and **duration**, with counts of unique VLRs and MSCs. An operator can **clear** all tracked active subscribers (`DELETE /api/subscribers`) to reset the register's view.

The **HLR Links** tab checks connectivity to the configured HLR API (`/api/hlr-links`): the **status** (up, degraded or down), measured **latency**, reported **version** and the target **host**. It is the quickest way to confirm the stack can reach the HLR back-end.

IP-SM-GW

The **IP-SM-GW** view is the short-message gateway. It carries the SMSc subscriber tracker and the link to the SMSc back-end, alongside the shared transport tabs and an **Overview**.

The **SMSc Links** tab checks connectivity to the configured SMSc API (`/api/smsc-links`): **status** (up, degraded or down), **latency**, reported **version** and target **host** — the SMS-path equivalent of the HLR link check.

The **SMSc Subscribers** tab lists the subscribers tracked by the SMSc subscriber tracker (`/api/smsc-subscribers`): each with **MSISDN**, **IMSI**, **HLR GT** and **VLR GT**, **messages sent** and **received**, status and last update, plus active/failed and unique-HLR counts. An operator can **clear** all tracked SMSc subscribers (`DELETE /api/smsc-subscribers`).

CAMEL-GW

The **CAMEL-GW** view surfaces the CAMEL sessions the gateway is running, alongside the shared transport tabs and an **Overview**.

The **CAMEL Sessions** tab lists all active CAMEL sessions managed by the gateway (`/api/camel-sessions`): each session's **state**, **CAP version**, **service key**, **IMSI**, **calling** and **called** numbers, start time and **duration**, with counts of sessions initiated and answered. It is where you watch live CAMEL call handling in real time.

Status

The **Status** endpoint (`/api/status`) reports the overall node health that the **Overview** tabs summarise: the node identity and **role**, whether the service is **ok**, its **uptime**, and a peer summary of how many signalling peers are active out of the total. This is the first read to confirm the stack is running and its peers are in service.

Related

- **STP** — the SS7 routing element (point-code and global-title routes).
- **HLR** — the Home Location Register element (active subscriber register and HLR link).
- **IP-SM-GW** — the IP Short Message Gateway element (SMSc subscribers and link).
- **CAMEL-GW** — the CAMEL Gateway element (live CAMEL sessions).
- The **MSC/VLR** raises the UpdateLocation and CAMEL signalling this stack routes and tracks.
- The **SMSc** is the back-end the IP-SM-GW's SMSc link and subscriber tracker resolve against.
- ← Operations Guide

TAS — Telephony Application Server

The **TAS** (Telephony Application Server) is the IMS application server that delivers the subscriber's telephony experience. It handles **call control** and **supplementary services** (call forwarding, barring, waiting, hold and the rest), runs **conferences**, takes and plays back **voicemail**, records **CDRs** for every call, resolves **routing** and number translation, and talks to the **OCS** over Diameter so that calls are charged in real time. Where the CSCF decides *how* a call is signalled, the TAS decides *what* the service does. OmniWeb surfaces its live calls, subscriber registrations, gateways, Diameter peers and configuration on one element page; call-volume and trend charts live on the Grafana dashboards reached from the same page.

← Operations Guide

The TAS is a per-instance element page reached from the sidebar when one runs in the site's inventory. All traffic goes through the OmniWeb backend proxy to the TAS's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Overview

The landing tab is a single-screen health read for the application server: the number of **active calls**, **registered subscribers**, running **conferences** and stored **voicemail** records, the health of the **Diameter** peers and **SIP gateways**, and the **licence** status. It is the fastest way to confirm the TAS is up and carrying traffic.

Active Calls

Every call currently in progress, each row showing the calling and called parties, the **direction**, the call **state** and its **duration**. Selecting a call opens its detail so you can see how a specific call is legged and where it sits in the dialplan — the first place to look when a subscriber reports a call in trouble right now.

CDR

The **call detail records** the TAS has written — one row per completed call, with the parties, **start time**, **duration** and **disposition**. Open any record for its full detail. CDRs are the accounting and after-the-fact troubleshooting trail: use them to confirm a call happened, how long it ran and how it ended.

Subscribers

The subscriber **registrations** the TAS knows about, each with the subscriber **identity**, registration **state** and current **contact**. Open a subscriber for the full registration detail. This is where you confirm a subscriber is registered to the application server before chasing why a service is not working for them.

Diameter

The **Diameter peers** the TAS connects to — including the path to the **OCS** for charging and the **HSS** for user data — with each peer's hostname and **connection state**. Open a peer for its detail. When charging or subscriber lookups fail, this tab is where you confirm the underlying Diameter paths are up.

Gateways

The **SIP gateways** the TAS uses to reach other networks and trunks, each showing its name and current **status**. Open a gateway for its detail. Use this tab to confirm the interconnect gateways are registered and available before diagnosing failed off-net calls.

Conferences

The active **conferences** and their participants. Open a conference to see who is in it, and act on it in place — **lock** or **unlock** the bridge, **kick** a participant, turn **video on or off**, or **destroy** the conference entirely. This is the operator's live control over a running bridge.

Voicemail

The stored **voicemail** records, per subscriber. Each record can be **deleted** individually — for example to clear a message left in error or to free a full mailbox.

OCS Test

The active **OCS (Online Charging System) sessions** the TAS is holding, so you can see which calls are drawing against a real-time balance. An individual charging session can be **terminated** from here — useful for clearing a session that has stuck open after its call has gone.

Routing

The **dialplan templates** the TAS routes calls against. The list shows each template; open one to read its content. This is the reference for how numbers are matched and where calls are sent — read it alongside the **Number Translation** tool to confirm a given number routes as intended.

Configuration

The **application configuration** the TAS is running, grouped by application. Open an entry to read its settings. This is the read reference for confirming the server is configured as expected without leaving OmniWeb.

Prompts

The **text-to-speech prompt recordings** the TAS plays to callers — the announcements and menu prompts — with the status of each. Use this tab to confirm the expected prompts are present and rendered.

Tools

Alongside the tabs above, the TAS page carries a set of on-demand tools for validating routing and subscriber data. Each takes a small form, runs the request against the application server through the OmniWeb proxy, and shows the result in place.

Call Simulator

Places a test call through the TAS to validate routing and media. Enter a **destination** (and optionally a source number, source IP and a forced disposition), and the simulator walks the call through the dialplan **without ringing anyone** — it returns the resolved **call type** (MT, MO or emergency), the on-/off-net decision, every **dialplan variable** it set, and a step-by-step list of **processing notes**, so you can see exactly how a number would be handled before a real call is placed. The OCS authorisation and HLR lookup can each be skipped to isolate a stage. A separate **echo-test call** action rings a destination and loops the party's own audio back, which confirms the live voice path to a handset.

Sh Query

Queries the HSS over the **Sh** interface for a subscriber's user data. Enter the subscriber's **public identity** and choose the **data reference** to fetch — from the full **repository data** to a specific element such as the IMS user state, the serving S-CSCF name, the initial filter criteria, the MSISDN or the IMSI — and the tool returns the transparent and repository service data the HSS holds. Use it to confirm what the application server sees for a subscriber when a service is not behaving as expected.

HLR Lookup

Performs an **HLR lookup** over SS7 MAP (a Send Routing Information request) to resolve a subscriber's routing and roaming state. Enter the **MSISDN** in international format and the tool returns the HLR response together with the **dialplan variables** the result would set on a call. This is how you confirm a number's current network location before diagnosing why calls to it route the way they do.

Number Translation

Tests how a dialled number is **normalised and translated** by the TAS routing rules. Enter a **country code**, the **phone number** and an optional **disposition**, and the tool returns the translated number the dialplan would use — read it alongside the **Routing** tab to confirm a given number is matched and rewritten as intended.

ESL

An interactive **console** for issuing FreeSWITCH-style commands to the TAS engine for diagnostics. Type a command and press Enter — the output is printed back in the console, and Arrow Up/Down recalls previous commands. Example commands such as `status`, `sofia status`, `show channels` and `show registrations` are offered as one-click chips. It is the lowest-level view of the running engine, for confirming live channels, gateway state and registrations directly.

Logs

The **Logs** tab tails the TAS's live log output for troubleshooting a single application server without leaving OmniWeb.

Monitoring dashboards

Call volumes, per-service counters and other time-series for the TAS are presented as Grafana dashboards, reached from the element page — OmniWeb

does not duplicate those charts natively.

Related

- The **CSCF** signals the calls the TAS applies services to — the two together deliver IMS voice.
- The **HSS** holds the subscriber and Sh user data the TAS resolves against.
- The **OCS** rates and charges the TAS's calls in real time over Diameter.
- The **DRA** relays the TAS's Diameter signalling (Sh, Gy) to those peers.

Test Devices

Test Devices turns real Android handsets at remote sites into a remotely-controllable test farm. From OmniWeb you can watch a phone's screen live, drive it (tap, swipe, type, hardware keys), place calls and send texts, and run scheduled health checks — VoLTE registration, signal, connectivity — whose pass/fail results are exported to Prometheus for alerting and dashboards.

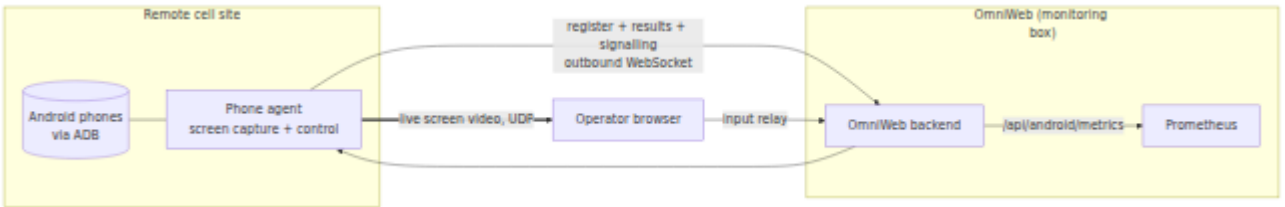
← Back to Operations Guide

It exists to answer the question a NOC keeps asking about a remote cell site: *"is a real phone actually working on this network right now?"* — registered for VoLTE, able to call, text, and reach the internet — without anyone driving to the site.

Architecture

Each host with phones plugged in via ADB runs a lightweight **agent**. The agent dials *out* to central OmniWeb, so sites behind NAT need no inbound access. Live video uses **WebRTC** (UDP) so it survives lossy, high-latency links far better than TCP screen-sharing; input and test results are relayed through central.

> **Network paths matter.** The web UI, input and call setup ("signalling") ride the control plane over HTTPS to the OmniWeb server, but the live screen video streams **directly between the operator's browser and the agent host over UDP** — it does not pass through the server. If a browser can reach the server but not the agent host over UDP, the console will hang on "Connecting…". See **Network Requirements** for the firewall rule to request and how to recognise the problem.



Component	Role
Phone agent	Discovers ADB devices, captures the screen, injects input, runs host test scripts. One per site; installed by the `device_tester` Ansible role.
Central OmniWeb	Device registry, WebRTC signalling relay, script dispatch, results history, Prometheus metrics.
Prometheus	Scrapes scheduled-test results from `/api/android/metrics`.

Device types

The farm is no longer phones-only. Every connected device is one of three kinds, auto-detected by the agent and shown with its own badge (and the Android robot logo for phones) in the fleet:

Kind	What it is	Console
Android phone	A normal handset (Android 5.0+ / API 21+) over ADB	Live screen + full control
USB modem	A cellular module exposing AT ports (e.g. Quectel EP06)	AT-driven cellular console — status, calls, SMS, cell lock, VoLTE
Old-Android dongle	A cheap LTE dongle running old Android (API < 21, e.g. UZ801) too old for the live-screen view	Poll-based **screencap** view + control
RAN Simulator UE	A virtual UE from the RAN Simulator — no hardware	UE control panel — attach/detach, idle, out-of-coverage, IMS register, SMS, calls

Clicking a device opens the console appropriate to its kind. Inside any console the **Swap phone** button (top-right) switches to another device without going back to the fleet grid.

Simulated UEs from the RAN Simulator appear in the fleet tagged as simulators, so you can drive emulated subscribers alongside real handsets — and the Test Devices filter lets you narrow to just the RAN Simulator UEs. They use the **same UE control panel** shown in the RAN Simulator's own Network Map, so there is one place to attach, idle, drop out of coverage, register to IMS, and send SMS/calls. See RAN Simulator for the full UE lifecycle.

Adding a device to a site

A device joins the fleet automatically once the **agent host** at that site can see it over ADB. There are two steps: enable ADB on the handset, then — if the agent runs in a VM or container — pass the USB device through to it.

1. Enable ADB on the phone

1. On the handset, open **Settings → About phone** and tap **Build number** seven times to unlock **Developer options**.
2. In **Settings → System → Developer options**, turn on **USB debugging**.
3. Plug the phone into the agent host with a **data-capable** USB cable.
4. The phone shows an *"Allow USB debugging?"* prompt with the host's RSA key — tick **Always allow from this computer** and accept. (Without the tick, the prompt returns every time it re-connects and the device shows as `unauthorized`.)

A normal handset needs nothing more — the test scripts and live screen use only standard ADB. Keep the screen unlocked for the first connection so the prompt can be accepted.

2. Pass the USB device through to the agent host

Skip this if the agent runs on bare metal. If the agent host is a virtual machine or container, the hypervisor owns the USB port, so the device must be forwarded:

Platform	How to forward the device
Proxmox — VM	`qm set <vmid> -usbN host=<bus>-<port>` to bind a physical port (the mapping survives swapping phones on that port), or `host=<vendor>:<product>` to follow one specific device. Find the ids with `lsusb` on the Proxmox host.
Proxmox — LXC container	Add to `/etc/pve/lxc/<ctid>.conf`: a device-cgroup allow for USB (`lxc.cgroup2.devices.allow: c 189:* rwm`) and a bind-mount of the USB bus (`lxc.mount.entry: /dev/bus/usb/<bus> dev/bus/usb/<bus> none bind,optional,create=dir`), then restart the container. Re-plugging a device onto a different bus needs the matching bus mounted.
VMware	With the VM running, **VM → Removable Devices → <phone> → Connect** (or add a USB controller and attach the device in the VM settings).

> USB modems and old-Android dongles forward the same way — they appear as the same USB device class, so a bus that's passed through carries phones, modems and dongles alike.

3. Confirm it appears

On the agent host, `adb devices` should list the handset as `device` (not `unauthorized` or `offline`). Within a few seconds it appears in the **Fleet View** for that site. If it doesn't, see Troubleshooting.

Fleet View

The **Test Devices** view lists every connected handset across all sites, grouped by site. Each card shows the model, manufacturer, Android version, screen resolution, network operator, battery, and build — at a glance you can see which phones are online and healthy.

The fleet groups devices by site (e.g. "Cell Site 12", "Lab North"). Each card shows live status, battery, and operator. Select phones and a script to fan a test out across them; click a card to open its console.

Labelling a device. The pencil on a fleet card opens **Edit device metadata**, where you can give a device a friendly **name** and record its **operator**, **IMSI** and **MSISDN**. These are operator-assigned (modern Android hides the IMSI/MSISDN from ADB) and stick to the device even when it's offline — the name is what shows in the console headers and device pickers.

The sidebar's **Test Devices** entry expands to show each site (with a device count), the **Agents** view, the script library, the scheduler and the device tools, so you can jump straight where you need.

Running tests across devices

Select one or more phones, choose a script, and **Run on selected** fans it out in parallel. Results stream into a table — one row per device with a pass/fail badge, exit code, duration, and expandable output.

Agents

The **Agents** view (sidebar → Test Devices → Agents) lists every bench connected to this server — one row per site host — with its online status, IP, the devices it carries, when it registered, and its software version. It answers "which sites are actually reporting in, and from where?" at a glance, and is where you confirm a newly-installed bench has dialed in.

Device Console

Clicking a phone opens its **console**: a live view of the screen with full remote control.

*The console streams the phone's screen live. Click to tap, drag to swipe, type to enter text. A **network-quality** badge (top-right of the screen) shows green/amber/red at a glance — click it for the full stream stats (resolution, FPS, bitrate, 5-second packet-loss %, jitter, RTT). The hardware-key bar (back, home, recents, volume, power) and quick actions (Airplane, Dialer, Messages, Settings) sit below; a tabbed tool panel — Cellular validation, Scripts, Shell, Logcat, Files and Record — is alongside.*

> **One viewer at a time.** A device streams to one operator at a time. If someone else is already viewing it, the console shows *"In use by <user> (<IP>)"* — the IP lets you tell whether it's you on another machine or a different operator — with a **Take over** button to claim the stream. A dropped or reloaded console reconnects to its own session automatically rather than fighting for it.

Remote control

Action	How
Tap	Click anywhere on the screen
Swipe	Click and drag
Type	Click the screen, then type on your keyboard
Hardware keys	Back, Home, Recents, Volume ±, Power buttons
Quick actions	One-click Airplane toggle, or open the Dialer, Messages or Settings app. These fire in the background — they don't run a test or block the console.

The streamed screen is the live device — any app or web page renders and is fully drivable from the console. Here a test phone has a page open in the browser, controlled remotely:

Viewing more than one phone

To watch several handsets at once — both ends of a test call, say — open one phone's console and click **View second phone** (top-right). Pick another phone from the list and its live screen opens alongside the first, with its own full controls: tap/swipe/type, hardware keys, quick actions, listen-to-audio and stream stats. Add more the same way, up to **four screens** at once.

Each extra screen carries its own picker to **swap** which phone it shows and an × to close it. The arrangement is captured in the page address (`?with=…`), so a specific multi-phone layout can be bookmarked or shared.

The tabbed tool panel (Cellular, Scripts, Shell, …) always acts on the **primary** phone — the one you opened — which is named in the panel header so it's clear where a script or command will run.

Console tools

Alongside the screen, a **tabbed tool panel** keeps the per-device tools together — switch tabs without losing each tool's state:

Tab	What it does
Cellular	The call / text / ping / speedtest validation tools (below)
Scripts	Run a library script on this device and see its full run history, with a one-click re-run on any past run
Shell	An interactive command/response console for the device
Logcat	Live device log stream with text + level filtering
Files	Browse, download, upload, rename and delete files on the device
Record	Capture a tap/swipe sequence as replayable script lines

Cellular Validation panel

Built-in tools for the checks an operator runs by hand, each taking the inputs you'd expect:

Tool	Inputs	What it does
Call	Number, hold time	Dials, confirms the call goes off-hook, holds, hangs up
Text	Number, message	Sends an SMS and confirms it left the device
Ping	Host, count	Pings a host over the device's data connection
Speedtest	Test URL	Measures download throughput over the device's data link

The **Number** box remembers numbers you've recently called or texted (kept in your browser only, not on the server) and offers them as a dropdown, so re-dialling a test number is a click.

Scripts

The **Scripts** tab runs any library script against this one device and keeps its full run history — filterable by source, status and test — without leaving the console. Every past run has a **re-run** button, so repeating a check (or a script that just failed) is one click; scripts that take inputs re-prompt for them.

Shell

The **Shell** tab is an interactive command/response console for the device — type a command, see its output and exit code, with ↑/↓ to recall history. It's the device analogue of the modem AT console, for the times a quick command is faster than writing a script.

Logcat

The **Logcat** tab streams the device's live system log. Filter by free text and minimum level (Verbose → Fatal); levels are colour-coded. **Start**/**Stop** controls the stream, and stopping keeps what you've captured on screen so you can scroll back through it.

Files

The **Files** tab is a file manager for the device. Browse directories (starting at `/sdcard`), **download** a file to your machine, **upload** a file to the device, and **rename**, **delete** or create folders. Useful for pulling a capture or diagnostic file off a device, or pushing a config onto it.

Gesture recording

Toggle **Record**, perform a tap/swipe sequence on the screen, and the console emits the equivalent `adb input` lines. Paste them into a folder script to turn a manual UI flow into a repeatable test.

USB Modem console

A USB cellular modem (Quectel EP06 and similar) appears in the fleet as a **modem** and opens a cellular console instead of a screen. AT commands are relayed agent → modem over its serial (AT) port; every command and response is logged to the **Terminal** tab as an audit trail.

Tab	What it does
Status	One-shot read of model / firmware / IMEI / IMSI / SIM, signal (CSQ + per-RAT QCSQ), operator, EPS registration, attach, IMS, IP, APN, DNS/PCO
Actions	Set APN (preset or custom, with **PDP / IP type**), attach/detach data, show IP + DNS, ping, send SMS, radio (airplane / reboot)
Networks	Operator scan (`AT+COPS=?`) and a serving/neighbour **cell measurement** (`AT+QENG`) → EARFCN / PCI / band / RSRP / RSRQ / SINR table
Cell lock	Lock to a specific **EARFCN (+PCI)** (`AT+QNWLOCK`), band lock (`AT+QCFG="band"`), RAT preference, or release back to auto
Calls	Dial / answer / hang up, an `AT+CLCC` call table, and an **Enable / Disable VoLTE** control (`AT+QCFG="ims"`)
SMS	Read the inbox (`AT+CMGL`), delete, and send
Events	Enables the relevant URCs and polls them into a timestamped log (registration / call / signal / new-SMS changes)
Terminal	Raw AT entry plus the full request/response log

The full AT-command reference, with real EP06 responses, is in **modem-at-commands.md**.

> **VoLTE:** the EP06 is VoLTE/IMS-only for voice. *Enable VoLTE* sets `AT+QCFG="ims",1`; an `AT+CFUN=1,1` reboot may be needed to (de)register IMS, and it only registers if the network + SIM provision it and the correct carrier MBN is selected.

Old-Android dongles (screencap mode)

Cheap ~$10 LTE "MiFi" dongles (e.g. the UZ801, Qualcomm msm8916) run **Android 4.4 (API 19)**, which is too old for the live-screen view. These devices fall back to a **screencap** console: the agent polls `adb screencap` for frames (a few per second — `screencap` itself is the bottleneck on this hardware) and injects `adb input` for control. You get a click-to-tap, type-to-text view — not smooth video, but enough to drive it.

The agent smooths over the rough edges of these dongles on the host side:

- **ADB unlock** — they ship in RNDIS mode; a udev-triggered service `curl`s the dongle's `usbdebug.html` to switch it into ADB mode. See agent/host-setup/mifi-dongle.
- **Auto-English** — they ship in Chinese; the agent sets `persist.sys.language` to `en-US` the first time it sees an API-<21 device (disable with `OMNIWEB_DONGLE_AUTO_ENGLISH=0`).

Programmatic control (MCP)

The same device-control stack is exposed as an **MCP server**, so an MCP client (Claude, scripts) can drive the farm: `list_devices`, `screenshot`, `tap` / `swipe` / `input_text` / `key`, `adb_shell`, `list_scripts` / `run_script`, and modem `at_command` / `send_sms`. It wraps the `/api/android/*` API and is bearer-token gated. Setup and the `claude mcp add …` command are in **mcp/README.md**.

Scripts

Tests are bash scripts that run on the **agent host** (so host tools like `grep`/`awk` are available) and target a device via ADB. There are two kinds:

- **Folder scripts** — no-argument health checks, run with just the device. These appear in the library, the fan-out runner, and the scheduler.

Examples: Device Connected, VoLTE Registration, Report Signal Strength, Toggle Airplane Mode.

- **Built-in tools** — the parameterised UI tools (Call, Text, Ping, Speedtest) driven from the Cellular Validation panel. They take inputs and are not part of the library or scheduler.

The script library lists the no-argument folder scripts. Each shows its description and timeout; click one to view its source (read-only). Recent runs appear below.

A script reports its result by **exit code** — `0` is a pass, anything else a fail — and its captured output is shown in the results table.

Scheduled Tests

Scheduled tests are defined declaratively in `schedules.json` in the scripts directory. The file is re-read on every cycle, so edits take effect without a restart. Each entry runs a folder script against its target devices on an interval.

Configuration

```
{
  "schedules": [
    {
      "name": "VoLTE Registration",
      "script": "volte_registration.sh",
      "interval_seconds": 120,
      "serials": [],
      "enabled": true
    }
  ]
}
```

Parameter	Type	Required	Default	Description
`name`	String	Yes	-	Unique label for the schedule. Appears as the `schedule` label on the exported metric.
`script`	String	Yes	-	Filename of a folder script in the scripts directory to run.
`interval_seconds`	Integer	No	300	How often to run the script, in seconds. Minimum 30.
`serials`	List	No	`[]`	Device serials to target. An **empty list means all currently-connected devices**, so the schedule auto-follows whatever phones are plugged in.
`enabled`	Boolean	No	true	Whether the schedule is active. Set `false` to pause it without removing the entry.

Example: VoLTE check across a site

```
{
  "name": "VoLTE Registration",
  "script": "volte_registration.sh",
  "interval_seconds": 120,
  "serials": [],
  "enabled": true
}
```

How it works: Every 120 seconds the scheduler runs `volte_registration.sh` against every connected phone. The script checks the device is voice-registered (`IN_SERVICE`) on LTE with VoPS (Voice-over-PS) support, and exits `0` if VoLTE is registered. The result becomes a Prometheus gauge per device.

Use case: Continuously prove that real handsets at a site can register for VoLTE, and alert the moment one can't.

Metrics

Central OmniWeb exposes the latest result of each scheduled test, per device, in Prometheus format at `/api/android/metrics`. Prometheus scrapes it as the `android_device_tests` job (configured automatically by the monitoring role).

Metric: `omniweb_android_test_pass` **Type**: Gauge **Description**: Result of the most recent scheduled run — `1` for pass, `0` for fail **Labels**:

- `phone` — device model (e.g. `SM-A536E`)
- `serial` — device serial number
- `site` — the agent's site label (e.g. `Cell Site 12`)
- `agent` — the agent host the device is attached to
- `script` — the script that ran
- `schedule` — the schedule name

Companion gauges share the same labels:

Metric	Description
`omniweb_android_test_return_code`	Exit code of the last run
`omniweb_android_test_duration_ms`	Duration of the last run, in milliseconds
`omniweb_android_test_last_run_timestamp_seconds`	Unix time of the last run

Example queries:

```
# Phones currently failing VoLTE registration
omniweb_android_test_pass{script="VoLTE Registration Status"} == 0

# Failing tests grouped by site
count by (site) (omniweb_android_test_pass == 0)

# A test that hasn't run recently (stale agent or device offline)
time() - omniweb_android_test_last_run_timestamp_seconds > 600
```

Troubleshooting

A phone doesn't appear in the fleet

Symptoms: A device is plugged in at a site but isn't listed under Test Devices.

Possible causes:

- The phone isn't authorised for ADB on the agent host
- The agent isn't running or can't reach central
- The phone is in an offline/unauthorised ADB state
- The central Test Devices service isn't running (Test Devices is served by its own dedicated backend service, separate from the main request pool)

Resolution:

1. Confirm the phone shows as `device` (not `unauthorized`/`offline`) in ADB on the agent host, accepting the RSA prompt on the handset if needed.
2. Check the agent service is running and that its configured central URL and token are correct.
3. On central, confirm the Test Devices service is running and that the gateway routes `/api/android/*` to it. If the agent registered but the fleet is empty, this service is the usual cause.
4. Verify the phone's USB connection; re-seat the cable if it shows as offline.

The console screen stays on "Connecting"

Symptoms: The device console opens but the screen never appears, while the rest of the UI — fleet list, device details, input — works normally and the device shows **online**.

Possible causes:

- UDP is blocked between the browser's network and the agent host (the live video streams **directly** browser → agent host, not via the server — by far the most common cause)
- The agent can't start screen capture on that handset

This is **not** an SSL or MTU problem: TLS is proven working (the UI loaded), and with MTU issues the video would connect and *then* tear rather than never starting. The tell-tale sign is a device that shows **online** in the fleet yet whose live view alone won't start.

Resolution:

1. Confirm the browser's network can reach the agent host over **UDP** (the video path). This usually means a firewall/VLAN rule permitting the operator's subnet to the bench hosts over UDP — see **Network Requirements** for the exact rule to request and how to confirm the fix.
2. Check the agent log for screen-capture errors; some handsets can't have their screen captured over ADB.

VoLTE / signal checks report "not readable"

Symptoms: VoLTE or signal scripts fail with the telephony state unreadable.

Possible causes:

- The handset restricts telephony status to privileged access

Resolution:

1. Use test handsets that expose telephony status to ADB. Vendor-locked consumer devices may hide it without privileged access.

Test Devices — Network Requirements

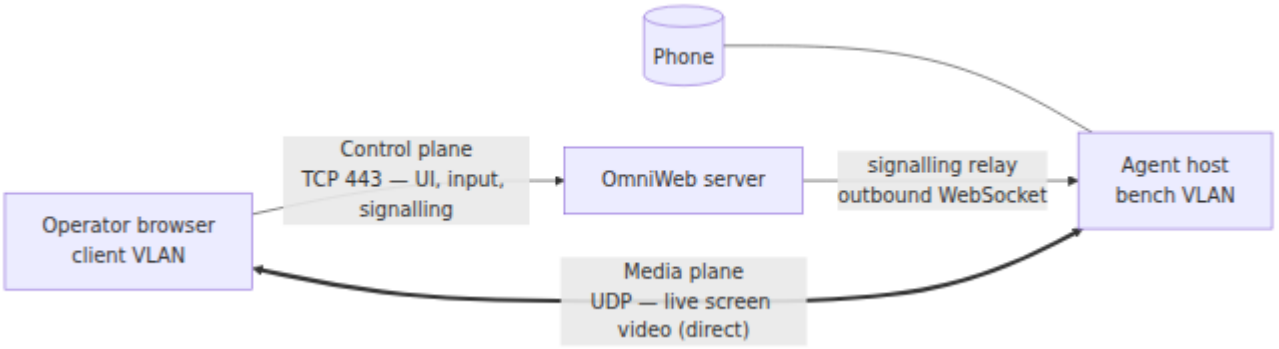
Live phone viewing in **Test Devices** uses **two separate network paths**. Understanding the split is the key to getting it working across firewalls and VLANs — and to diagnosing the single most common symptom, a console that sits on **"Connecting…"** forever.

← Back to Test Devices

The two paths

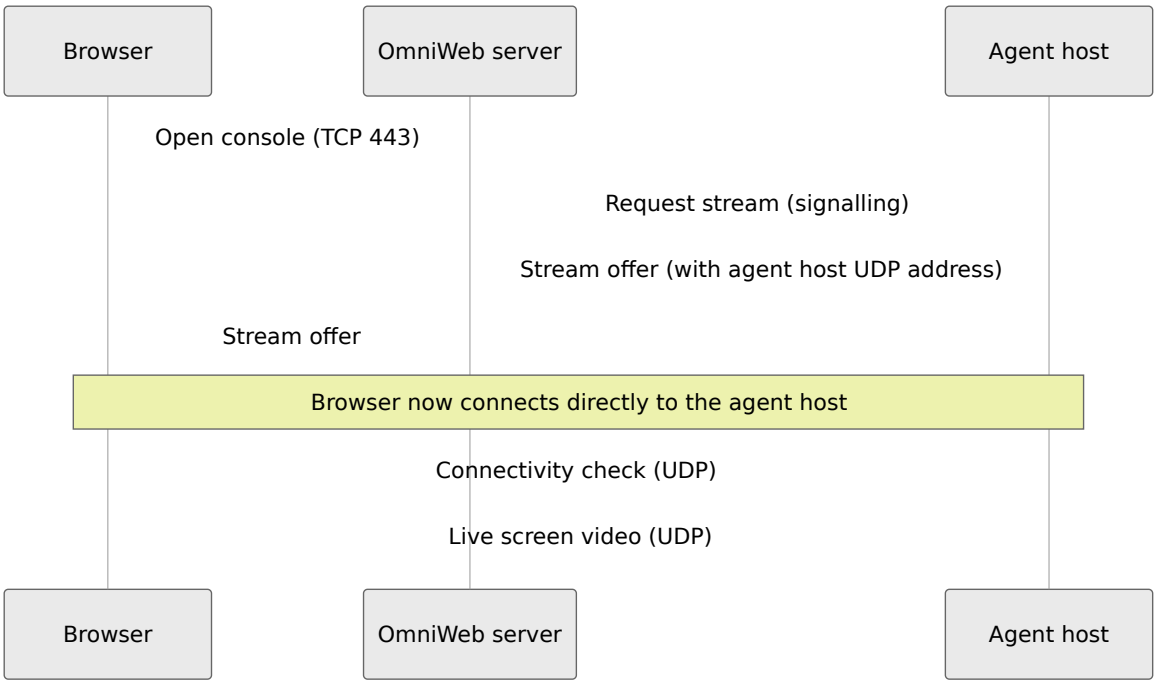
Path	Carries	Direction	Transport	Endpoint
Control plane	The web UI, login, the device list, remote input, test results, and the call setup ("signalling") for a live view	Browser → OmniWeb server	**TCP 443 (HTTPS)**	The OmniWeb server
Media plane	The live screen video itself	Browser ⇄ **agent host** (the bench the phone is plugged into)	**UDP** (WebRTC)	The agent host, **not** the server

The crucial point: **the live video does not flow through the OmniWeb server.** Once the server has brokered the connection, the video streams **directly between the operator's browser and the agent host** over UDP. A network that allows the browser to reach the *server* but not the *agent host* will load the UI, show the device list, and then hang on "Connecting…".



Why the media path is direct

Live screen video is sent peer-to-peer with **WebRTC over UDP** so it survives the lossy, high-latency links typical of remote cell sites far better than relaying it through a central server over TCP. The trade-off is that the operator's browser needs a **direct UDP path to the agent host** — there is no media relay in the middle to fall back on.



Firewall requirements

For an operator to view phones at a site, **both** paths must be permitted from the operator's network to the relevant hosts.

Control plane (usually already open)

Field	Value
Action	Permit (stateful)
Protocol	TCP
Source	Operator / client VLAN(s)
Destination	OmniWeb server
Destination port	`443`

If the OmniWeb UI loads at all, this path is already open — no change needed.

Media plane (the one that's usually missing)

Field	Value
Action	Permit (**stateful** — so return video is allowed automatically)
Protocol	**UDP**
Source	Operator / client VLAN(s) — where browsers run
Destination	The **device-bench host subnet** (the agent hosts), e.g. `10.0.20.0/24`
Destination ports	**`32768–60999`** (the default Linux ephemeral range the agent draws media ports from)

The browser always initiates the media connection, so a **stateful** permit on `client → agent host` UDP also allows the agent's return video automatically. No STUN/TURN or other ports are involved.

> **Do not change port 443 for this.** Signalling already rides the control plane; only the UDP media path to the agent hosts needs opening.

Recommended: pin the media port range

Opening the whole ephemeral range (`32768–60999`) is broad. The agent can instead be configured to draw its media ports from a small fixed range, shrinking the firewall rule to a tidy block:

Field	Value
Action	Permit (stateful)
Protocol	UDP
Source	Operator / client VLAN(s)
Destination	Device-bench host subnet
Destination ports	**`50000–50100`** (pinned range)

This is the preferred form to hand a network team — 100 ports to one subnet is far easier to approve and audit than the full ephemeral range. The pinned range is set on the agent host; coordinate the exact range with whoever maintains the benches so the firewall rule and the agent configuration match.

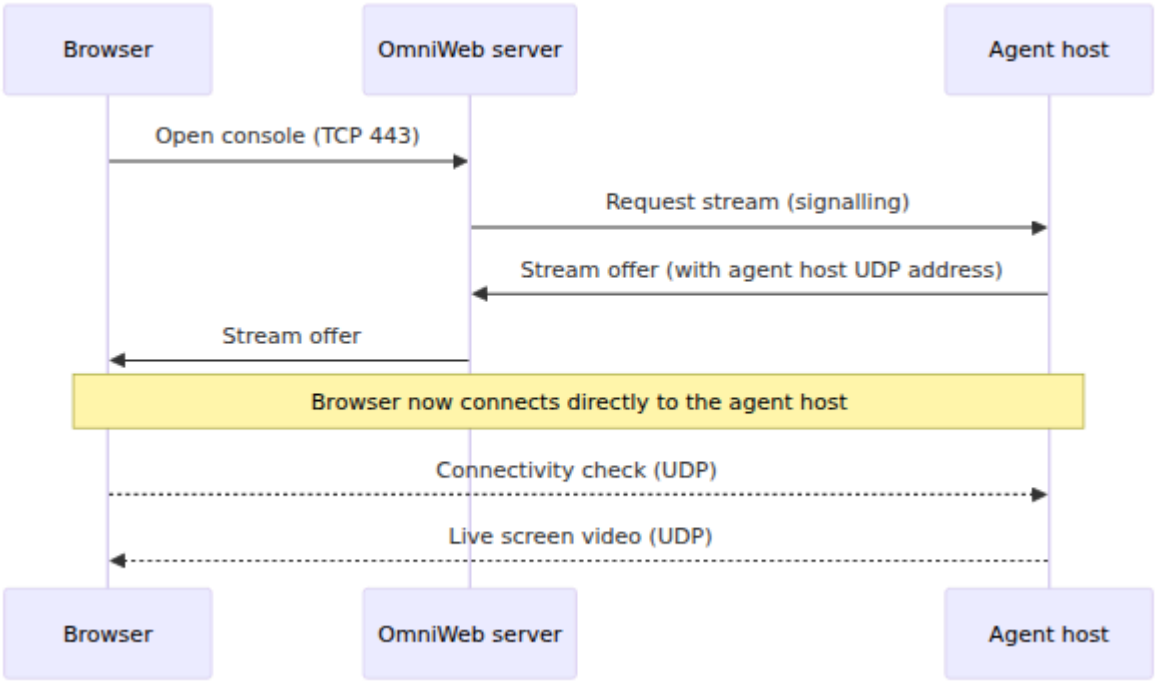
Recognising the problem

Symptom: the device console opens, but the screen never appears — it stays on **"Connecting…"** indefinitely. Everything else works: the fleet list, device details, even remote input setup.

This is almost always the media-plane UDP path being blocked. It is **not** a TLS/certificate problem and **not** an MTU problem:

Suspected cause	Why it's not this	What you'd actually see if it were
SSL / certificate	The UI, login, device list and signalling all succeed over HTTPS — TLS is clearly working	The page itself would fail to load, or show a certificate warning
MTU / fragmentation	MTU only matters once video packets flow; here **no** video packets ever arrive	Video would **connect**, then stall or tear — not sit on "Connecting…"
Agent / phone down	The device shows **online** in the fleet with live battery/operator data	The device would be missing from the fleet, or marked offline
UDP path blocked ✓	The browser can reach the server (UI loads) but not the agent host over UDP	Permanent "Connecting…" — exactly this symptom

A useful tell: the device tile shows the phone as **online** (so the agent and phone are healthy and the server can see them), yet the live view alone won't start. That combination points squarely at the media-plane firewall path.



Confirming the fix

After the firewall rule is applied, re-open the device console. If the screen appears within a few seconds and the **network-quality** badge (top-right of the live view) goes green, the media path is working. The badge's stream stats (resolution, FPS, bitrate, packet-loss %, jitter, RTT) confirm video is flowing end-to-end.

If it still hangs, verify with whoever maintains the benches that the firewall rule's **destination subnet and ports** match the agent hosts and their configured media-port range, and that the rule is applied to the operator's **actual** source VLAN (a laptop on guest Wi-Fi is a different source than the corporate LAN).

Multiple operator locations

Each operator network that needs to view phones must have the media-plane rule for its own source VLAN. The rule is per source subnet → bench subnet, so adding a new office, VPN pool, or site means adding its subnet as an additional **Source** on the same UDP permit. Sites whose operators only ever view phones from the bench LAN itself need no inter-VLAN rule at all.

Network Topology & Single Pane of Glass

OmniWeb's topology views are the "single pane of glass" entry point to the whole network. They show how the elements connect, which ones are deployed, and let you jump straight from a node on the map into that element's workspace.

← Back to Operations Guide

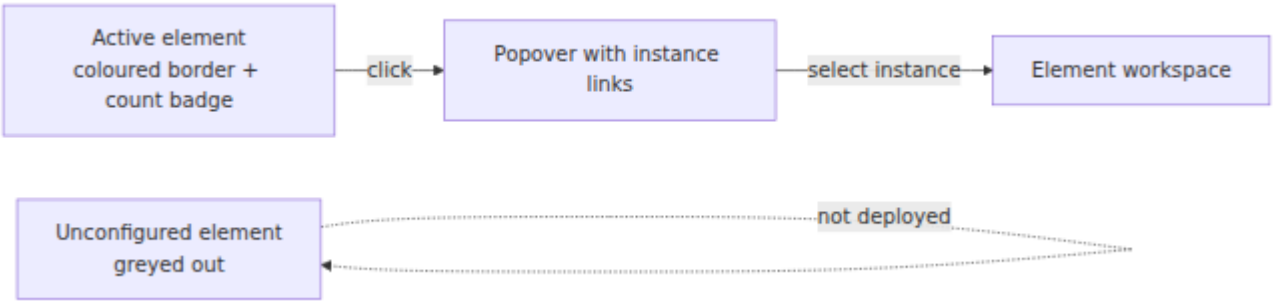
Topology Views

The landing/topology page presents the network across four domain tabs. Each is an interactive map of the relevant elements and their interfaces.

EPC topology: UE → eNB → MME → SGW-C → PGW-C → UPF, with Diameter signalling through the DRA to the HSS. Active elements show a coloured border and an instance-count badge.

Tab	Elements shown
Core (EPC)	UE → eNB → MME → SGW-C → PGW-C → UPF; DRA ↔ HSS; OCS; TWAG; RAN Monitor
IMS	P-CSCF → I-CSCF → S-CSCF; DRA ↔ HSS; TAS; IMS SMSc → SMSC → SMPP GW; OCS; Entitlement (OmniSEP)
CS Core	BTS → BSC → MSC; STP ↔ HLR / CAMEL GW / IP-SM-GW; HSS; SMSC; OCS
5G Core	gNB → AMF → SMF → UPF; NRF, SCP, AUSF, UDM, UDR, PCF; CHF → OCS; NSSF; BSF

Reading the Map



- **Active elements** — those with instances declared in `omniweb.json` — appear with a coloured border and a badge showing how many instances exist. Clicking one opens a popover with direct links to each instance, so you can navigate from the topology straight into an element's pages.
- **Unconfigured elements** appear greyed out, so the map doubles as a quick inventory of what is and isn't deployed.
- **Connections** indicate the interface between elements (e.g. SIP, Diameter, PFCP, GTP-C, SS7), distinguishing signalling types.

This makes the topology page the natural starting point: see the shape of the network, spot what's deployed, and drill in.

Element-Level Topology

Some elements also provide their own focused topology view. The **PGW-C → Topology** tab, for example, shows that node's relationships to its SGW-C, UPF, and Diameter peers, annotated with live session counts and utilisation — a localised single-pane view for that element's neighbourhood.

PGW-C topology: SGW-C, UPF, and Diameter peer relationships with per-peer session distribution.

Host List & Infrastructure

Beyond the network functions themselves, the **Host List** page is the single pane of glass for the *infrastructure* that runs them. It lists all infrastructure hosts by category, each with a direct link to its web UI:

Category	Examples
Monitoring	Grafana, Prometheus, Loki
SIP Capture	Homer Web UI
Charging	OCS Web UI
DNS	DNS servers
Virtualisation	Proxmox hypervisor consoles
Out-of-band	iDRAC server management
Routers	MikroTik WebFig, Peplink
Microwave	SAF, Ubiquiti AirFiber links
NAS	Synology DSM

The Host List — every infrastructure and element host by category, each with a direct link to its web UI (Grafana, Prometheus, Loki, element APIs, and more).

This gives operators one place to reach every supporting system — monitoring, hypervisors, out-of-band management, transport — without hunting for bookmarks.

TWAG — Trusted WLAN Access Gateway

The **TWAG** (Trusted WLAN Access Gateway) brings trusted Wi-Fi access into the mobile core. It authenticates Wi-Fi devices with EAP-AKA/AKA′ against the AAA/HSS over Diameter, hands each authenticated client an IP address by DHCP, and anchors its data back to the packet core over a GTP tunnel to the PGW — so a subscriber on Wi-Fi is served the same APN, policy and charging as on the cellular network. Its dashboard is the place to see who is connected, whether the Diameter and GTP paths are healthy, and to clear individual sessions.

← Operations Guide

The dashboard is a per-instance element page reached from the sidebar when a TWAG runs in the site's inventory. All traffic goes through the OmniWeb backend proxy to the TWAG's API, so it stays behind the single authenticated gateway. It is organised into tabs.

Overview

The landing tab is a single-screen health read: active and total **sessions**, connected/disconnected **Diameter peers**, accepted **subscribers**, the number of **access points**, **DHCP** lease count, the **GTP** tunnel status and its PGW address, the **authentication mode** in use (e.g. EAP-AKA), aggregate **traffic** in and out, uptime, and a live **recent-activity** feed of authentications and session starts/stops.

Sessions

Every authenticated Wi-Fi session, each row showing the subscriber **identity** (the network access identifier), **status**, assigned **IP**, client **MAC**, **SSID**, serving **NAS IP**, session **duration**, and **bytes in/out**. An operator can **delete** a session to force-disconnect a client — for example to clear a stuck or misbehaving device.

Access Points

The WLAN access points (NAS devices) currently connected to the gateway. Each can be **removed** individually when an access point is decommissioned or needs to be cleared.

Accounting

The accounting sessions the TWAG has recorded — the usage records raised for authenticated clients. Individual accounting sessions can be **deleted**.

Diameter

The Diameter peer connections to the AAA/HSS that carry authentication, with connected and disconnected counts. This tab also offers a **Diameter test** — it sends a test Multimedia-Auth-Request (MAR) to confirm the authentication path to the AAA/HSS is up, which is the quickest way to prove Diameter connectivity when authentications are failing.

DHCP

The DHCP leases the TWAG has issued to Wi-Fi clients — the address, client and lease state for each — so you can confirm clients are being addressed and see how much of the pool is in use.

Logs

The **Logs** tab tails the TWAG's live log output for troubleshooting an individual gateway without leaving OmniWeb.

Actions

Each action asks for confirmation first:

- **Delete a session** — force-disconnect one authenticated Wi-Fi session (Sessions tab, per row).
- **Delete an access point** — clear a single access point / NAS entry (Access Points tab, per row).

- **Delete an accounting session** — remove a single accounting record (Accounting tab, per row).
- **Diameter test** — send a test MAR to validate the Diameter path to the AAA/HSS (Diameter tab).

Related

- The **PGW** anchors the TWAG's user plane over GTP — see the packet-core elements.
- The **HSS** holds the subscriber authentication data the TWAG's Diameter requests resolve against.
- ePDG — the untrusted-Wi-Fi counterpart for Wi-Fi calling.

UPF — User Plane Function

The **UPF** (User Plane Function) is the packet-forwarding node of the mobile core. It carries every subscriber's actual traffic: it receives GTP-U tunnels from the access network on **N3** (S1-U in the EPC), forwards packets to and from the data network, applies per-flow QoS and volume accounting, and buffers downlink packets when a device is idle. It has no logic of its own — the control plane (an **SGW-C** in the EPC over **Sxa**, or an **SMF** in a 5G core over **N4**) programmes it entirely over **PFCP**, installing the packet-detection, forwarding, QoS and usage-reporting rules that describe how each session's traffic is handled. Omnitouch's **OmniUPF** implements the forwarding path as an **eBPF/XDP dataplane** in the kernel, so the rules the control plane installs become entries in eBPF maps that the XDP program consults for every packet. OmniWeb surfaces the UPF's PFCP associations and sessions, the live eBPF map contents, dataplane statistics, routing, downlink buffers, walled-garden state and configuration; throughput and trend charts are on the Grafana dashboards reached from the same page.

← Operations Guide

The UPF is a per-instance element page reached from the sidebar when one runs in the site's inventory, with an instance selector where several are deployed and a fleet view for aggregate status. All traffic goes through the OmniWeb backend proxy to the UPF's API, so it stays behind the single authenticated gateway. The documentation link in the page header (the in-context help button) opens this page. It is organised into tabs.

Fleet and instance selection

Where a site runs more than one UPF, the **fleet view** lists every instance with its health at a glance — the fastest way to confirm the whole user plane is up before drilling into one node.

From any tab, the **instance selector** in the page header switches which UPF the page is reading. Everything below — sessions, associations, eBPF maps, statistics — is scoped to the selected instance, so you pick the node once and the whole page follows it.

Overview

A single-screen health read of the UPF: its **health** status, the number of **PFCP associations** it holds with control-plane peers, the total **active sessions** across those peers, and the **XDP action counters** — how many packets the dataplane has passed, dropped, redirected, transmitted or aborted, which is the quickest confirmation that traffic is actually flowing. Beneath it, each **PFCP peer** is listed with its association state, session count and any missed heartbeats, and the **eBPF map sizes** show the capacity of the PDR, FAR, QER and URR maps and the maximum session count. A search box jumps straight to a session by **UE IP** or **TEID**, and a **Node Report** action sends a PFCP Node Report Request to the first active control-plane peer.

Sessions

Every **PFCP session** — the per-bearer / per-PDU-session user-plane state the control plane has installed. Each row is keyed by its **local and remote SEID** and shows the session's **IMSI**, **APN/DNN**, the **association** (which control-plane peer owns it), and the count of **packet-detection**, **forwarding-action**, **QoS-enforcement** and **usage-reporting** rules it carries. Search **by UE IP or TEID** to find the session serving a specific device or tunnel.

Session detail

Opening a session shows its full rule state. A **session rule map** draws how each **Packet Detection Rule (PDR)** references the **Forwarding Action Rule (FAR)**, **QoS Enforcement Rule (QER)** and **Usage Reporting Rules (URR)** it applies — clicking any node highlights just the rules connected to it, so a single flow's path through the dataplane is easy to trace. Beneath the map, four tables give the detail:

- **Packet Detection Rules (PDR)** — how incoming packets are classified: direction (uplink / downlink), the UE IP or inbound TEID they match, the SDF filters (protocol and port ranges), and which FAR, QER and URRs they resolve to.
- **Forwarding Action Rules (FAR)** — what happens to matched packets: **Forward**, **Drop**, **Buffer** or **Notify CP**, the outbound GTP-U tunnel details (remote IP and TEID) and network instance. FARs can be edited in place.
- **QoS Enforcement Rules (QER)** — the per-flow QoS: the **QFI**, the uplink and downlink **gate** status (open / closed) and the **maximum** and **guaranteed bitrates**. QERs can be edited in place.
- **Usage Reporting Rules (URR)** — the volume counters: current and last-reported uplink and downlink bytes and the report sequence number, which

is what the UPF feeds back to the control plane for charging and quota.

From the session header you can also **activate or remove a walled-garden redirect** for that specific session (see Walled Garden below).

Associations

The **PFCP associations** — the Sxa/N4 relationships the UPF holds with its control-plane peers (the SGW-C or SMF). Each row shows the peer's **remote IP and port**, its **session count**, whether it is currently **associated**, the number of **missed heartbeats**, and when it last **connected** or **disconnected**. This is where you confirm the user plane and control plane are in step and spot a peer whose heartbeats are lapsing.

Statistics

The dataplane's real-time numbers. **eBPF Map Usage** shows how full each map is — sessions, PDR, FAR, QER and URR — as used-versus-capacity bars that turn amber and red as they approach their limit, so you can see a UPF nearing its dimensioned capacity before it starts rejecting sessions. **XDP Statistics** breaks out the pass / drop / redirect / TX / aborted packet counters, and the **Metrics** section surfaces the UPF's Prometheus counters as individual figures.

The same used-versus-capacity model is what a capacity check reads: at a glance you can tell whether the constraining resource is the session table or one of the rule maps.

Routing

The routes that steer subscriber traffic to and from the UPF. The **Active UE IP Routes** table lists the routes the UPF currently holds for allocated UE IP pools, and a **Sync Routes** action reconciles them with the dataplane. Below it, an **FRR console** gives an in-context vtysh terminal against the UPF's routing daemon — type a `show` command (for example `show ip route` or `show ip ospf neighbor`) to inspect routing state, with tab-completion and command history, or switch to **configure mode** to make a change. This is how the UPF advertises its UE pools into the surrounding network without leaving OmniWeb.

Buffers

The **downlink packet buffers**. When downlink data arrives for a device that is idle (no active tunnel), the dataplane holds those packets against the relevant FAR and asks the control plane to page the device — the buffered packets are then flushed once the tunnel is re-established (this is the user-plane side of **Downlink Data Notification**). The tab shows the **total buffered packets** and **total FARs** holding packets, the **per-FAR** and **total** buffer limits, and a **lookup by FAR ID** to see how many packets a specific FAR is holding. Buffers can be **flushed** individually or **cleared** in bulk.

eBPF Maps

A direct browser onto the **eBPF maps** that are the UPF's dataplane. These are the raw kernel-side entries the XDP program consults for every packet, split across tabs:

- **Uplink PDR** — keyed by inbound TEID, mapping a tunnel to its PDR and FAR.
- **Downlink PDR** — keyed by UE IP, mapping a subscriber's downlink to its PDR and FAR.
- **FAR** — the forwarding actions, with decoded action, remote IP, outbound TEID and outer-header creation.
- **QER** — the QoS entries, with QFI, gate status and bitrate limits.
- **URR** — the usage counters, with current and last-reported uplink/downlink volumes.

Any row can be clicked to **edit the raw map entry**, which is a low-level troubleshooting and repair tool — the maps are normally driven entirely by PFCP, so hand-editing an entry is for diagnosis rather than routine operation.

Configuration

A read-only view of how the UPF is configured: its **network** identity (node ID, PFCP address and port, API port, N3 and N9 addresses), the **features** it has enabled (FTUP, UEIP, the route manager and its type, and the XDP attach mode — native, generic or offloaded), the **PFCP heartbeat** interval, timeout and retry settings, and the **dataplane** N3/N9 addresses. A **Raw JSON** toggle shows the full config document for support. This is where you confirm a UPF is dimensioned and addressed as expected.

Diagnostics

Runs a **ping from the UPF node** to a target host — enter a target and a packet count and the raw ping output comes back with a success/failure verdict. It is a quick way to confirm the UPF has data-network reachability (for example to a DNS server or upstream gateway) from where the packets actually leave, without shelling into the node.

Walled Garden

The **walled garden** — restricted forwarding for sessions that must reach only a captive portal before they are authenticated or entitled. Activating a redirect on a session's SEID rewrites its downlink so the device is steered to a **redirect URL / portal IP** instead of the open internet; the tab lists every **active redirect** with the UE IP, portal IP, redirect URL, gNB IP and the TEIDs involved, and a redirect can be removed to release the session to normal forwarding. A **whitelist** of IPs or CIDRs names the destinations a walled-garden session is still allowed to reach directly (for example the portal and its assets). Redirects can also be activated and removed per-session from the session detail page.

Logs

The **Logs** tab tails the UPF's live log output for troubleshooting a single instance without leaving OmniWeb.

Monitoring dashboards

Throughput, per-interface packet rates, session-count trends and other time-series for the UPF are presented as Grafana dashboards, reached from the element page — OmniWeb does not duplicate those charts natively.

Related

- **SGW-C** — the EPC control peer that programmes this UPF (as the SGW-U) over **Sxa/PFCP** and forwards traffic on **S1-U / N3**.
- **PGW-C / SMF** — anchors the PDN connection and, as the SMF, programmes the 5G user plane over **N4/PFCP**.
- **MME** — drives bearer setup that ultimately installs the sessions seen here.

- **SCEF** — carries small non-IP payloads over the control plane, deliberately bypassing this user plane.
- Operations Guide — back to the element index.