Building on a Charging API

Integration patterns for operators with engineering teams

Six integrations that come up repeatedly, what each takes, and the failure modes to design for before you meet them.

FOR Engineering and product teams at charging operators

UPDATED 4 July 2026

Contents

WHAT YOU GET

- Why every webhook handler must be idempotent, with the failure it prevents
- Pushing sessions into an ERP without a nightly reconciliation job
- Joining charging data to telematics for cost per kilometre
- What to build against the API rather than wait for on a roadmap

CHAPTER 01

The shape of a charging API

Charging platforms differ enormously in what their API exposes, and the difference is usually structural rather than a matter of missing endpoints. The question worth asking first is whether the operator console is a client of the same API you are given, or whether it talks to a privileged internal surface you cannot reach.

Why that question matters

If the console has its own private interface, everything you build is second class: it will lag the product, it will be missing the field you need, and nobody at the vendor will notice when it breaks. If the console is a client of the public API, then anything the console can do you can do, and the API cannot rot without the product rotting with it.

THE TEST

Open the console, do something interesting, and watch the network tab. If you see the same endpoints your documentation describes, you can build with confidence. If you see something else, plan for the API to be a reporting surface rather than an integration surface.

The resources you will actually use

RESOURCE	USED FOR
Sessions	Everything financial and analytical
Meter values	Disputes, and per-vehicle energy
Stations and connectors	Provisioning, and live status
Drivers and tags	Onboarding, groups, limits
Tariffs	Pricing changes at scale
Partners and settlements	Revenue share into finance systems
Commands	Start, stop, reset, unlock, availability

Pagination

A year of sessions is not a single response. Cursor pagination is what you want: an opaque token that points at a position, stable while the underlying data changes. Offset pagination on a growing table silently skips and duplicates rows as new sessions arrive during your loop, which for financial data is not acceptable.

- Always filter by a time range as well as paginating, so a re-run is bounded.
- Store the cursor with your job state, so an interrupted export resumes rather than restarts.
- Expect a page size limit and do not fight it. Loop.

Idempotency

Any write you retry — and you will retry, because networks fail after the server has acted — needs an idempotency key: a value you generate, send with the request, and reuse on retry so the platform recognises it as the same operation.

Without it, a timeout on a refund request leaves you unable to tell whether the refund happened. With it, you retry and get the original result.

Authentication and scoping

01 Use a key scoped to what your integration needs, not an owner key. A reporting job does not need permission to issue refunds.

02 Keep keys out of source control and out of the frontend. A charging API key can start and stop physical equipment.

03 Know how to revoke and rotate before you need to.

04 Where station groups exist, scope the key to the sites the integration covers.

Errors

Distinguish three cases in your client and handle them differently: your request was wrong and retrying will not help; the platform had a transient problem and retrying will; and the request was refused for a permission or state reason that needs a human. Treating all three as "retry with backoff" produces a job that hammers an endpoint for an hour over a malformed field.

CHAPTER 02

Webhooks and duplicates

Webhooks are how you find out that a session finished without polling for it. Every practical webhook system delivers at least once, which means it will eventually deliver twice — a network blip during your acknowledgement is enough. A handler that is not idempotent will eventually do something twice that should have happened once.

The failure, concretely

Your handler receives session.completed and posts a revenue entry to your ERP. The acknowledgement is lost. The platform redelivers. You post a second revenue entry. Nobody notices until a month-end reconciliation is out by an amount nobody can explain.

THE FIX IS FOUR LINES

Every event carries an identifier. Before doing anything with side effects, record that identifier in a table with a unique constraint. If the insert conflicts, you have seen it — acknowledge and return. This is the whole of idempotent webhook handling and it is skipped constantly.

Acknowledge fast, work later

Do not do the work inside the webhook request. Validate, deduplicate, enqueue, return 200. Then process from the queue.

- Delivery systems time out. Slow handlers get retried, which produces more duplicates.

- Your ERP being slow should not cause the platform to consider delivery failed.
- A queue lets you replay after fixing a bug, which a synchronous handler does not.
- It separates "we received it" from "we processed it", which are different questions during an incident.

Verify the signature

A webhook endpoint is a public URL that causes financial records to be created. Verify the signature on every request, compare it in constant time, and reject anything that fails — without exception for testing, because the exception is what will be left enabled.

Ordering

Events for one charger arrive in order; events across chargers do not, and a retried event can arrive after a later one. Do not build logic that assumes a global sequence.

Where order matters, use the timestamps in the payload and make your handler tolerate arriving late — updating a record only if the incoming event is newer than what you have.

What to subscribe to

EVENT	TYPICAL USE
session.completed	Revenue, cost attribution, analytics
session.started	Live dashboards, fleet visibility
station.status_changed	Availability tracking
station.offline	Operations alerting
settlement.generated	Partner payouts into accounts payable

Subscribe narrowly. Every additional event type is more traffic, more duplicates and more code to keep idempotent, and most integrations need two of the five.

Reconcile anyway

Webhooks are a latency optimisation, not a source of truth. Run a daily job that pulls sessions for the previous day by API and compares them to what you received. It catches the events that never arrived, and it is the difference between believing your data is complete and knowing it.

CHAPTER 03

ERP and finance

The most common first integration, and the one where mistakes are most expensive because they surface as accounting discrepancies months later. The goal is that every rupee in your accounts can be traced to a session, and every session can be found in your accounts.

What to post, and when

Post on session completion, not on session start — a started session has no amount. Post the components separately rather than a single total.

- Net revenue, energy component.
- Net revenue, time or session-fee component, if you charge one.
- Idle fees, separately — they behave differently in a revenue-share calculation.
- Tax, as its own line.
- The session identifier, on every line, as the trace key.

CARRY THE SESSION IDENTIFIER EVERYWHERE

It is the join key between your platform and your accounts, and adding it later means backfilling. Every journal line, every invoice line, every settlement line should carry it.

Payments are not revenue

A session generates revenue at completion; the money arrives when the gateway settles, which is later and in aggregate. Modelling these as the same event produces a system that cannot explain the difference between what was earned and what was received — which is precisely what finance needs to see.

Post revenue against a receivable at session completion, and clear the receivable against gateway settlements. The residue is your reconciliation problem, and it should be small and explicable.

Refunds and credit notes

A refund is not a negative session. Post it as a credit against the original session, carrying the same identifier and a reason. Refund reasons aggregated over a quarter are a genuine quality signal, and they are only available if the reason travelled with the money.

Partner settlements

Settlement runs produce a payable per partner. Post it as such, with the statement identifier and the period, and reconcile the sum of statement lines against the sessions that produced them.

RECONCILIATION	CADENCE	CATCHES
Sessions posted vs sessions in platform	Daily	Missed or duplicated webhooks
Receivable cleared vs gateway settlements	Daily	Failed captures, chargebacks
Revenue vs energy delivered	Weekly	Tariff misconfiguration, unbilled sessions
Settlement statements vs session records	Monthly	Revenue-share errors

The unbilled session

Sessions that delivered energy and collected nothing are inevitable, and they are invisible in an integration that only posts what was collected. Post them too, at zero, with a flag. A monthly report of unbilled sessions is one of the most useful things a finance integration can produce, and almost nobody builds it.

CHAPTER 04

Fleet and telematics

A fleet does not want charging data. It wants cost per kilometre, and that requires joining charging data to telematics data. The integration is not difficult; the identifier mapping is where it fails.

The mapping is the project

Charging knows a tag or an authorisation identifier. Telematics knows a registration number or a VIN. Nothing connects them unless somebody maintains a table that does, and that table has to change when a vehicle changes hands.

01 Hold it as a first-class record: tag, vehicle identifier, registration, VIN, cost centre, valid from, valid to.

02 Date-bound it. A tag reassigned in March must attribute January's sessions to the old vehicle.

03 Make updating it part of the vehicle allocation process, not an occasional administrative task.

04 Report on sessions that could not be mapped. A rising count is the earliest sign the table has gone stale.

ATTRIBUTE TO THE VEHICLE, NOT THE DRIVER

Drivers change vehicles. A driver-keyed tag attributes a truck's energy to whoever was driving, which makes cost per kilometre meaningless. Keep the driver as a second field for accountability; key the cost on the vehicle.

What to pull, and how often

DATA	SOURCE	CADENCE
Session energy, cost, timestamps	Charging API	Daily, or on webhook
Distance by vehicle by day	Telematics	Daily
Public charging sessions	Charging API, via roaming	Daily
Site metered consumption	Meter or bill	Monthly

The numbers worth computing

- kWh per kilometre by vehicle. Compare identical vehicles on similar routes; a persistent gap is driving style or a developing fault.
- Cost per kilometre by route, which tells you which contracts are actually profitable.
- Depot versus public cost per kWh, which prices the convenience of opportunity charging.
- The trend in kWh per kilometre for one vehicle over months, which is where battery degradation becomes visible.

The reconciliation that keeps it honest

Sum of session energy for the depot against the site's metered consumption for the month. The gap is losses, non-charging site load and anything unattributed. A stable small gap is fine; a growing one means a tag is unmapped, a charger is not

reporting, or a meter is drifting.

Run it monthly and the whole analysis stays trustworthy. Skip it and the numbers slowly stop meaning anything, without ever visibly breaking.

CHAPTER 05

Warehouse and reporting

Beyond a certain size, an operator stops wanting reports from the charging platform and starts wanting the charging data next to everything else — finance, telematics, CRM — in a warehouse where it can be joined. That is a different integration from the transactional ones, with different failure modes.

Incremental, not full

Pull by a time window rather than re-pulling everything. But sessions can be updated after creation — a refund, a late meter value, a corrected tariff — so a naive "created after X" filter will miss changes to older rows.

01 Pull by updated-at where the API offers it, not created-at.

02 Overlap the window: re-pull the last few days each run, and upsert. Cheap, and it catches late corrections.

03 Upsert on the session identifier, never insert blindly, or a re-run duplicates a day.

04 Keep the raw payload alongside the parsed columns, so a field you did not model is recoverable without a backfill.

THE MISTAKE THAT PRODUCES SILENT GAPS

A nightly job that pulls "yesterday" and fails on Tuesday leaves a hole nobody sees. Record the watermark you actually reached, and resume from it, rather than computing the window from the current date.

Meter values are the volume

Sessions are modest in volume. Meter values, at a sample every minute for every session, are one to two orders of magnitude larger, and they are what make session-level analysis possible.

Decide deliberately whether you need them in the warehouse. For billing disputes, the platform is the right home. For analysing charging curves, derating and vehicle behaviour, you need them locally — and you should plan the storage rather than discovering it.

A workable schema

TABLE	GRAIN	NOTES
sessions	One row per session	The spine; carries every foreign key
meter_values	Session × timestamp × measurand	Partition by date
stations	Slowly changing	Keep history — sites get renamed and moved
tariffs	Version, with effective dates	Never overwrite a version
settlements	Statement line	Traceable to sessions
vehicle_map	Tag × vehicle × validity window	The join to telematics

Two things to keep as history

Stations and tariffs change, and reports about last year must reflect last year. Storing only the current state of either means a historical report silently reprices the past — the same class of error as editing a tariff in place, arriving through the warehouse instead.

Prove it is complete

Run one check after every load: session count and energy total per day in the warehouse against the same figures from the platform. Two numbers, and they either agree or you have a gap. Without it, a warehouse degrades quietly into a source of confident wrong answers.

CHAPTER 06

Provisioning at scale

At ten chargers, provisioning by hand in a console is fine. At two hundred a year it is a source of inconsistency: a tariff attached to the wrong group, a station named differently from every other station, a connector count that does not match the hardware. Programmatic provisioning is about consistency more than speed.

What a commissioning script does

01 Creates the location, if it does not exist, with coordinates and address from a single source rather than typed twice.

02 Creates the station with a name generated from a convention, not from whoever is on site.

03 Creates connectors matching the hardware specification, so a mismatch is caught before the unit arrives.

04 Attaches the correct tariff and, where relevant, the site’s load-management ceiling.

05 Assigns the station to the right station group, so field staff can see it immediately.

06 Generates the QR code payload for printing, so the sticker is produced from the record rather than transcribed.

NAMING CONVENTIONS ARE WORTH MORE THAN THEY SOUND

A network where station identities follow a rule can be filtered, grouped and reported on. One where they were typed by different people over three years cannot, and no amount of tooling fixes it afterwards. Encode the convention in the script and it enforces itself.

Idempotency again

A commissioning script is run, fails halfway, and is run again. Without idempotency the second run creates duplicate stations. Make every create an upsert keyed on your own identifier, and the script becomes safe to re-run — which is what makes it usable in the field.

The pre-arrival check

The most valuable thing provisioning-by-API enables is validating the plan before the hardware ships: does every station in the rollout have a location, a tariff, a group and the right connector configuration? A spreadsheet of two hundred rows checked against the platform catches errors while they are still text.

Decommissioning

The half nobody writes. A charger removed from a site should be marked as such, not deleted — its sessions and its history remain, and a deleted station orphans them.

- Set it inactive rather than removing it.
- Detach it from the site so availability reporting for that site stops counting it.
- Record where it went, if it is being redeployed.

- Revoke or reassign any QR codes, because the sticker outlives the installation.

Where the script should live

In version control, with the rollout data as a checked-in file rather than a spreadsheet on somebody's laptop. The provisioning of a network is a description of that network, and it is worth being able to see what changed and when.

About this document

Written by the team building zevOS, the charge point management platform. We publish this kind of thing because operators who understand utilisation, demand charges and settlement make better decisions — including about us.

Corrections and disagreements are genuinely welcome. If something here is wrong, or true in most states but not yours, write to support@zevos.ai and we will fix it in the next revision.

White Stripe Innovations Private Limited. Last updated 4 July 2026. This document is general information about running a charging business, not legal, tax or financial advice.