



Simplycity — OEM / Operator API Integration Guide

Version 1.0-draft · 1 October 2026 · For aircraft manufacturers, fleet-management vendors and licensed operators



Everything in this document describes live endpoints on the running Simplycity platform, not a plan for a future API. Every request shape, validation range and error below is read directly from the server code and verified against <https://simplycity.nl/api/v1/spec>, which returns the same contract machine-readable. If this document and that URL ever disagree, the URL is correct — tell us and we'll fix the document.

1. What this is

Simplycity Flow is the mission-release and coordination layer for an urban drone corridor. It does not build or fly aircraft. Your system — the aircraft, its ground control station, or your fleet-management platform — remains the source of truth for everything about the aircraft: position, battery, health, faults. Simplycity's job is to take that data, check it against a named release policy (12 gates, detailed below), and turn a clinical transport request into an authorised, audited mission.

Integrating means your system does two things:

1. Sends Simplycity live telemetry while an aircraft is on the ground or flying a mission, and reports availability and post-flight data.
2. Reads the mission package Simplycity has authorised, so your aircraft knows the route, payload class and operating window it's cleared for.

Nothing in this contract asks you to accept a flight-release decision from Simplycity without visibility into why — every gate that passes or fails, and the exact data behind it, is visible in the platform console and in the `/api/v1/missions/{id}/package` response.

2. How your telemetry actually gets used

This isn't a generic telemetry sink. Every field you send feeds a specific, named release gate in Flow's policy engine — and if a gate fails, the mission does not launch. That's the whole point of the integration: your data has to be good enough, and current enough, for a human operator to safely sign a release on it.

Gate (policy engine name)	What it checks	Real threshold today	Fed by
communications_ready	Is telemetry current?	Telemetry older than 20 s fails the gate	every POST /telemetry call's ts
aircraft_healthy	Any faults? Battery in range? Correct config?	faults must be empty; batt_temp_c within 5–45 °C; config_version must match the approved build	faults, batt_temp_c, config_version
energy_sufficient	Enough charge for the route plus reserve?	30% contingency reserve above a 20% floor, compared against route distance	soc_pct, predicted_soc_dest_pct
maintenance_valid	Is the airframe in its maintenance window?	Reported via POST /assets/{id}/availability, not telemetry	availability state
route_available	Is there a flyable contingency plan?	>=2 contingency landing sites on the corridor (Simplycity-side data, not yours)	—

In flight, the same fields drive live safety monitoring, not just the pre-launch gate:

In-flight signal	Threshold	What happens
Link quality (link.quality)	below 0.60	LINK_DEGRADED alert raised
Cross-track deviation (deviation_m)	above 60 m	ROUTE_DEVIATION warning
Cross-track deviation (deviation_m)	above 120 m	GEOFENCE_BREACH — critical incident
State of charge (soc_pct) in flight	below 26%	ENERGY_LOW critical — operator is told to return or divert
Any entry in faults	any	AIRCRAFT_FAULT critical incident, immediately

If your telemetry stops arriving, Flow does not assume the aircraft is fine — the gate and the in-flight monitor both fail closed. This is the one piece of behaviour we will not make configurable for a partner: stale or missing data is always treated as a problem, never as "probably still okay."

3. Getting access

1. Request a key. Submit the form at simplycity.nl/platform/api.html (<https://simplycity.nl/platform/api.html>) ("Request

API access"), or email the contact in §10, with your organisation name, a technical contact, and what you intend to connect (aircraft telemetry, a fleet-management system, etc.).

2. We verify your organisation. A Simplycity administrator reviews the request — there is no automated approval. This is a deliberate, human step while the platform is pre-pilot.
3. We issue a scoped key. Format: `scsk_` followed by a random 30-byte token (e.g. `scsk_8pQ2v...`). The key is shown to us once at creation time — we send it to your technical contact over a channel we agree with you; we never have it again afterwards, because only its SHA-256 hash is stored.
4. You start sending telemetry. No separate "sandbox" environment exists today — the live platform is the integration target, scoped to the test aircraft asset ID we assign you during onboarding (see §9).

A key can be revoked at any time from the Simplycity admin console, immediately and without needing your cooperation — do the same on your side and tell us if a key is ever compromised.

4. Authentication

Every request below (except `GET /api/v1/spec`, which is public) requires:

```
X-Api-Key: scsk_your_key_here
```

Failure	HTTP status	Body
Missing header	401	<code>{"error":{"code":"invalid_api_key","message":"missing or invalid X-Api-Key"}}</code>
Unknown or revoked key	401	same as above
Key valid, but wrong scope for this endpoint	403	<code>{"error":{"code":"forbidden","message":"key scope does not allow telemetry"}}</code>

Keys are scoped (today, every issued key carries the single scope `oem:telemetry`, which covers every endpoint in this document) and every call is logged against the key's prefix in Simplycity's tamper-evident ledger — `OEM_AVAILABILITY_REPORTED`, `OEM_CAMERA_REGISTERED`, `OEM_POST_FLIGHT_REPORT` events all record which key made the call.

5. Base URL, transport and limits

Base URL	<code>https://simplycity.nl/api/v1</code>
Transport	HTTPS only (TLS via Let's Encrypt); plain HTTP is not served
Request format	JSON body, Content-Type: <code>application/json</code>
Max request size	4 MB (telemetry payloads are a few hundred bytes — this is not a practical limit)
Rate limit, all calls	25 requests/second sustained, burst 80, per source IP
Rate limit, writes (POST)	4 requests/second sustained, burst 30, per source IP
Recommended telemetry rate	1 Hz while airborne, ≥ 0.2 Hz on the ground

1 Hz telemetry is comfortably inside both limits. If you need a higher sustained rate for a fleet of aircraft behind one IP, tell us — the limit is per source address, not per key, so a shared NAT egress for a whole fleet is the one case worth flagging in advance.

6. Endpoints

6.1 POST /telemetry — aircraft state

The core of the integration. Send this on the ground (low rate) and continuously in flight (1 Hz recommended).

Request body

Field	Type	Range / values	Required
<code>asset_id</code>	string	must match an asset ID Simplycity has registered for you	yes

ts	number	epoch seconds; rejected if more than 120 s from server time (either direction)	no — defaults to server time
position.lat	number	-90 ... 90	yes
position.lon	number	-180 ... 180	yes
position.alt_m	number	-50 ... 500	yes
speed_ms	number	0 ... 60	yes
heading_deg	number	0 ... 360	yes
flight_mode	string	PAD · VERTICAL_CLIMB · TRANSITION_OUT · CRUISE · LOITER · TRANSITION_IN · VERTICAL_DESCENT · LANDED	yes
soc_pct	number	0 ... 100 (battery state of charge)	yes
batt_temp_c	number	-20 ... 90	yes
link.primary	string	up / down	no — default up
link.contingency	string	up / down	no — default up
link.quality	number	0 ... 1	yes
faults	array of strings	max 20 entries, each <=40 chars	no — default []
route_progress	number	0 ... 1	no
eta_s	number	0 ... 86400	no
predicted_soc_dest_pct	number	-50 ... 100	no
deviation_m	number	0 ... 20000 (lateral distance from the route centreline)	no
config_version	string	your airframe/software build identifier	no — keeps the last known value if omitted
sim_t	number	only meaningful inside Simplycity's own simulator — omit this	no

Example

```
curl -s -X POST https://simplycity.nl/api/v1/telemetry \
-H "X-API-Key: scsk_your_key_here" -H "Content-Type: application/json" \
-d '{
  "asset_id": "NL-041",
  "ts": 1759276800,
  "position": {"lat": 52.3676, "lon": 4.8360, "alt_m": 110},
  "speed_ms": 26.4,
  "heading_deg": 118,
  "flight_mode": "CRUISE",
  "route_progress": 0.42,
  "eta_s": 420,
  "soc_pct": 71.5,
  "predicted_soc_dest_pct": 52.0,
  "batt_temp_c": 33.1,
  "link": {"primary": "up", "contingency": "up", "quality": 0.93},
  "faults": [],
  "deviation_m": 8,
  "config_version": "SC17-CFG-0.1"
}'
```

Response — 200

```
{"accepted": true, "asset_id": "NL-041", "server_time": 1759276800.1, "quality": "good",
"mission_id": "SC-AMS-1046"}
```

quality is "degraded" whenever link.quality is below 0.6 or link.primary is "down" — this is computed and returned on every call so you can see, on your side, the moment Simplycity judges your link as unreliable, before it ever becomes a visible alert to an operator.

Validation errors (422, code invalid_telemetry unless noted) fire on: a ts more than 120 s off (code stale_sample), any numeric field outside its range above, an unrecognised flight_mode, or faults not being a list of <=20 items. Unknown asset_id returns 404 (code unknown_asset).

6.2 GET /missions/{mission_id}/package — what you're cleared to fly

Poll this to confirm what a mission is actually authorised for before or during a flight — route, payload constraints, and whether release has happened.

Response — 200

```
{
  "mission_id": "SC-AMS-1046",
  "state": "AIRBORNE",
  "route_configuration_version": "COR-AMS-01-v3",
  "corridor": "COR-AMS-01",
  "origin": "HUB-OLVG-W",
  "destination": "HUB-AMC",
  "waypoints": [{"lat": 52.3676, "lon": 4.8360}, {"lat": 52.3510, "lon": 4.8000}, "..."],
  "route_length_m": 18034,
  "operating_window": {"earliest": "06:00", "latest": "22:00"},
  "payload": {"class": "LAB_BLOOD_CHILLED", "max_mass_kg": 1.2, "handling": "temperature 2-8 °C, keep upright, sealed cassette"},
  "release": {"authorised": true, "snapshot": {"id": 214, "decision": "released", "hash": "..."},
  "policy_version": 7, "route_version": 3}},
  "escalation_contact": "ops@simplycity.nl",
  "assigned_asset": "NL-041"
}
```

release.authorised is the one field worth checking programmatically: it's true only while the mission is in RELEASED, LAUNCHING, AIRBORNE or APPROACHING. If it's false, you should not be flying this mission regardless of what your own system thinks — the release snapshot (release.snapshot) is the immutable, hashed record of the decision and the exact policy/route versions it was made against.

6.3 POST /assets/{asset_id}/availability — report fleet status

Request body

Field	Type	Values	Required
state	string	READY or MAINTENANCE	yes
reason	string	free text, shown to the operator	no

Rejected with 409 if the asset is currently MISSION or RESERVED — you cannot pull an aircraft out of service mid-assignment through this endpoint; that has to go through the console with an operator in the loop.

6.4 POST /missions/{mission_id}/post-flight — reconcile after landing

Send this once, after landing. Simplycity compares your reported energy use against what its own telemetry-derived model expected, and flags a discrepancy automatically if the two differ by more than 10%.

Request body

Field	Type
actual_takeoff	epoch seconds
actual_landing	epoch seconds
energy_kwh	number
route_adherence_pct	number
exception_codes	array of strings
noise_dba_lmax	number

Response

```
{"ok": true, "reconciliation": "consistent"}
```

— or "discrepancy flagged", which also raises a DATA_DISCREPANCY alert visible to the operator manager. This is intentional: it is evidence, not a penalty. A real difference between your post-flight report and in-flight telemetry is exactly the kind of thing Trust is supposed to surface, not hide.

6.5 POST /assets/{asset_id}/camera — register a live camera feed

Optional. If your aircraft or ground system can supply a live video feed, register its URL here and Simplycity's mission console will display it on the operator's screen for the duration of the mission.

Request body

Field	Type	Required
stream_url	string, <=2000 chars — a short-lived, signed yes HLS/WebRTC URL you control	
kind	string — nadir or forward	no — default nadir
expires_at	epoch seconds	no — defaults to 1 hour from registration

The console reads this back via GET /api/missions/{mission_id}/camera (session-authenticated, for the operator's own use — not part of your integration) as {"connected": true, "stream_url": "...", "kind": "...", "expires_at": ...}. Nothing is shown to an operator as a "live camera" unless a real feed has been registered through this exact call — Simplycity does not fabricate or simulate a feed on your behalf.

7. Error handling

Every error, on every endpoint, has the same shape:

```
{"error": {"code": "machine_readable_code", "message": "human-readable explanation", "...any extra context": "..."} }
```

HTTP status	Typical cause
401	Missing, unknown or revoked X-API-Key
403	Key is valid but its scope doesn't cover this call
404	Unknown asset_id or mission_id
409	Conflicting state (e.g. asset already on a mission)
422	A field failed validation — out of range, wrong type, or stale timestamp
429	Rate limit exceeded — back off and retry
500	Our fault, not yours — if this ever happens, it's logged server-side; tell us

8. Data handling

- Your telemetry is stored with its source and received time attached; Simplycity never overwrites or re-derives a value you sent — it records what you said, when, and compares it against policy.
- No patient-identifiable data should ever appear in any field of this API — mission payload references are opaque order IDs, not patient records, and Simplycity's own payload policy prohibits it platform-wide.
- Every write your key makes is appended to Simplycity's hash-chained, tamper-evident event ledger — the same ledger that records every human decision on the platform. Your integration's actions are auditable on exactly the same terms as a person's.
- API keys are stored as a SHA-256 hash, never in plaintext, and are immediately and permanently revocable.

9. Before you go live: integration checklist

- [] Received a scsk_... key and confirmed which asset_id(s) it's scoped to test against.
- [] Can authenticate: GET /api/v1/spec (no key needed) returns this same contract as JSON — useful as a connectivity smoke test before touching authenticated calls.

- [] Can send one valid POST /telemetry call and get back {"accepted": true, ...}.
- [] Confirmed your ts values are within 120 s of real time — a NTP-synced clock on your side is sufficient; this is the single most common integration failure.
- [] Sent an out-of-range value on purpose (e.g. soc_pct: 140) and confirmed you correctly handle the resulting 422.
- [] Can call GET /missions/{id}/package for a test mission and parse release.authorised.
- [] Reported availability via POST /assets/{id}/availability and seen it reflected (ask us to confirm on the console side during onboarding).
- [] Decided whether you'll send POST /post-flight reports and, if so, whether energy_kwh is derived the same way as the in-flight telemetry (to avoid spurious discrepancy flags).
- [] Reviewed §2 so your team understands which fields are safety-relevant, not just informational — a late or missing batt_temp_c is not a cosmetic gap, it fails a gate.

10. Contact

Simplycity B.V. — integration enquiries: ops@simplycity.nl Live platform: simplycity.nl/platform (<https://simplycity.nl/platform/>) · Live, machine-readable contract: simplycity.nl/api/v1/spec (<https://simplycity.nl/api/v1/spec>) · This document and all others: simplycity.nl/resources/deliverables (<https://simplycity.nl/resources/deliverables/>)

This is a v1-draft contract on a pre-pilot platform. No real aircraft is connected today — the endpoints above are live and already exercised by Simplycity's own simulated SC-17 traffic and automated test suite, but you would be among the first real integrations. Field names and validation ranges are considered stable; we will not remove or narrow a field without talking to every connected partner first.