



Simplycity, Role by Role — A New Joiner's Walkthrough

PURPOSE

get someone new to the project from "what even is this?" to "I get it" in one read, by following a single mission through every person who touches it. Audience: new team members, interns, advisors. Status: describes the working sandbox at /platform/ (/platform/) — simulated aircraft, real workflow logic.



If you only remember one sentence: Simplycity does not build or fly drones — it is the coordination and evidence layer that lets a hospital, an operator, a city and an aircraft partner all trust the same flight.

1. The cast: seven roles, one mission

The platform's access control (server/auth.py) defines exactly these roles. Nobody can do more than their role allows — that boundary is the product.

Role	Who they represent	What they're accountable for
Clinical requester	A hospital clinician or lab coordinator	Asking for a transfer; cancelling it if it's no longer needed
Hub attendant	Staff at the launch or receiving pad	Physically scanning the payload into and out of custody
Remote operator	The licensed drone operator's duty pilot	Running the release checklist, signing the release, flying and closing the mission
Operator manager	The operator's ops-desk lead	Configuring the fleet and release rules, reviewing exceptions, backing up the duty pilot
City / authority viewer	The municipality or regulator	Read-only oversight: ledger, holds, noise records, audit exports
Simplycity administrator	The platform owner (us)	User accounts, integrations, sandbox scenarios — least-used, most-logged role
OEM integration service	A machine account, not a person	The aircraft/operator system that pushes telemetry through the API

Two of these (operator manager, admin) are "privileged" — they need a sandbox passcode to sign in, on top of picking a name. Everyone else signs in with one click, because the sandbox holds no real people's data.

These seven roles aren't arbitrary — they mirror the three product modules from the pitch: Flow is mostly the clinical requester, hub attendant and remote operator moving a mission forward; Trust is the ledger, alerts and configuration control that every role writes to and the authority viewer/admin watch over; Impact is what operator manager, authority viewer and admin read back out once missions have happened.

2. One mission, start to close

This is the exact state machine the code enforces (server/missions.py), narrated as who does what. Open /platform/ -> "Watch a guided demo" -> The 17-minute corridor (/platform/) to see it happen live instead of reading about it.

Step 1 — Clinical requester opens a request

State: DRAFT -> REQUESTED

A clinician requests a blood-sample transfer between two hubs. The request carries an opaque order reference and a payload description (mass, volume, temperature band) — never patient data. This is the only role that can create or cancel a mission.

Why it matters: the demand signal for the whole system starts with a real clinical need, not a drone looking for a job.

Step 2 — Hub attendant prepares and scans the payload

State: REQUESTED -> PAYLOAD_READY

The attendant at the origin hub packs the sealed cassette, scans it in, and records mass, temperature and seal status. This opens custody — from this point, the platform can show exactly who has held the payload and when.

Why it matters: a courier company's biggest liability question is "where was it and who had it." This step is where that answer starts existing.

Step 3 — Remote operator runs the release gates

State: PAYLOAD_READY -> PREFLIGHT_CHECK -> RELEASED

The duty pilot opens the mission and the platform evaluates 12 release gates against live data — aircraft state, battery, weather, airspace, payload limits, and more. Any gate that's stale or missing data fails closed: it blocks the release rather than assuming the best. If every gate passes, the operator — a named individual, not "the system" — signs the release. That action freezes an immutable snapshot: every input value and the exact rule version used, so "why was this released?" always has a stored answer.

If a gate fails (e.g. insufficient battery reserve for the route), Flow refuses the release with a named reason and a suggested fix — the operator (or operator manager) resolves it and re-runs the check. Nothing launches on a stale pass.

Why it matters: this is the moment Simplycity is actually selling — not the flight, but the discipline of checking the right things, on live data, with one accountable name attached, every single time.

Step 4 — The aircraft flies

State: RELEASED -> LAUNCHING -> AIRBORNE -> APPROACHING

The operator launches. The aircraft (simulated SC-17 in the sandbox; a real OEM feed in production) reports telemetry through /api/v1 — the same contract a real aircraft or operator system would use, driven by the OEM integration service account. Energy use, altitude and route adherence are logged in real time. If anything changes — a storm arrives, a link drops — the operator or operator manager can hold, return or divert, and each of those is its own named, ledgered decision.

Why it matters: most of what a city or hospital actually wants to see isn't "the drone flew," it's "the drone did what it was supposed to, and someone was watching the whole time."

Step 5 — Receiving hub attendant closes custody

State: APPROACHING -> LANDED

The destination hub attendant scans the cassette out. If anything about the received condition doesn't match what left the origin (temperature, seal), that's recorded as an exception rather than silently accepted.

Step 6 — Remote operator reconciles and closes

State: LANDED -> HANDED_OVER -> CLOSED

The operator reconciles the aircraft data and the custody record and closes the mission. A mission that landed abnormally (returned, diverted, an incident) instead routes through incident review, owned by the operator manager or admin, which requires a recorded root cause before it can close.

Why it matters: "closed" isn't just a status flip — it's the point where the mission becomes a complete, queryable record that Impact can report on and Trust can prove was never altered.

Throughout — Trust is always watching, Impact is always counting

Every one of the six steps above writes to a SHA-256 hash-chained ledger — append-only at the database level, not just by convention. The authority viewer can watch any of it in real time without touching a control; the admin can verify the whole chain or run a live tamper demonstration to prove edits are detectable. Meanwhile Impact is computing this mission into the running scorecard: on-time rate, energy per kilometre, noise, interventions per 100 missions — the numbers the pitch deck's financials are built on.

3. When it doesn't go to plan

The happy path above is one of several scripted scenarios available from the guided demo. Briefly:

- A gate fails. Flow refuses the release with a named reason; the operator fixes it (e.g. swaps aircraft) and re-releases. The refusal itself is logged as evidence — a RELEASE_DENIED entry, not a deleted attempt.
- Weather turns mid-flight. A critical alert fires, the operator holds and orders a return, the payload is recovered at the origin hub, and an incident review closes the loop with a root cause.
- Telemetry drops. The last known position is kept and clearly flagged as stale rather than guessed at; the gap is reconciled once the link returns.

Full narrated versions of all four scenarios are in SIMPLYCITY-Demo-Script.md.

4. Why this whole loop is useful — not just how it works

- For the hospital: a faster, auditable alternative to a road courier for time-critical samples, with a named accountable person at every handoff.
- For the operator: a release process that makes "we checked everything, on live data, and can prove it" true by construction, not by paperwork after the fact.
- For the city / regulator: visibility into every flight and every refusal without needing to trust anyone's word for it — the ledger is independently verifiable.
- For Simplycity: every mission that completes this loop is simultaneously a delivered service (revenue), a safety record (trust), and a data point (the Impact metrics the business case depends on). The same six steps are the

product, the safety case and the evidence base at once — that's the whole bet behind "infrastructure that compounds."

5. Where to go next

Want to...	Go to
See this walkthrough happen live	/platform/ (/platform/) -> Watch a guided demo
Read what the public/city sees	/community/ (/community/)
See the exact API an aircraft/operator connects through	/platform/api.html (/platform/api.html)
Check what has to be true for the business to work	/platform/economics.html (/platform/economics.html)
Run the 10-minute version of this for a partner or investor	SIMPLYCITY-Demo-Script.md
Understand build status against the original design	SIMPLYCITY-PoC-Guide.md
See the full technical design	SIMPLYCITY-Platform-Design.md

What this is not

This document describes a working sandbox with a simulated aircraft, not an operational system. No real patient data, real aircraft, or regulatory approval is involved. Role names, permissions and the mission state machine are real code, running today; the organisations, sites and flight history around them are illustrative or synthetic, and clearly flagged as such inside the platform.