

ATI Straton Flex Rewrite

An independent replacement for the fixture's web and dimming application — what it does, how it is deployed, and how it measured against the original.

Technical overview · 10 October 2026 · Gerd Rooselaers · rooselaers.be/Acropora

Scope

This replaces one layer and one layer only: the Node.js application that serves the fixture's web interface and drives its LED dimming. The hardware, the Onion Omega2+ board, OpenWrt, the init system, SSH, networking, and the LED driver board underneath are all untouched and unmodified.

It runs as a drop-in substitute for the original application's own process — same init script, same restart mechanism, same `node --expose_gc` invocation the device already used. It speaks the same serial protocol to the LED driver, reverse-engineered from the original application's observed behaviour rather than guessed, and it keeps the same bearer-token HTTP API, so existing automation continues to work without changes.

|Built for two specific fixtures. This is not a general-purpose release and is not published as source. It was written against the author's own two out-of-warranty units and is documented here because the reliability notes it follows from are public.

Architecture

The original application is a single file of roughly 23 KB plus a bundled dependency tree. The rewrite is a set of small single-responsibility modules with no runtime dependencies at all — nothing outside the Node standard library, which is what lets it drop the bundled tree entirely.

Area

Responsibility

orchestrator, scheduler	Resolving the lighting schedule for the current moment and driving output
hardware, protocol, dimmer	Serial framing to the LED driver board, per-channel value computation
fixture_runtime	Schedule, spots and colour state; the single place that writes schedule files
web_server, templates, sessions, auth	HTTP routing, pages, login state, password and token handling
temperature, par_power	Live sensor sampling, smoothing, derating and shutoff
peer_sync	Optional schedule replication to a second fixture
memwatch, event_log	Memory ceiling enforcement, in-process event history
system_backup, support_bundle	Config export and diagnostics archive, with secrets redacted
setup_wizard, wifi_setup, timezone_setup, ssh_setup	First-boot and settings flows that previously needed shell access

It carries its own automated test suite — several hundred tests — run both on a current Node release and specifically on the Node 8.10.0 the fixture itself runs. That second target matters: a test passing on a modern engine proves nothing about a nine-year-old one on a 124 MB board with no swap, and more than one real incompatibility was caught that way before reaching hardware.

What it adds

- **Temperature-driven safety clamp.** The driver board's own temperature sensors are polled live, smoothed, and wired into a gradual derating curve plus a hard shutoff, rather than being a display-only number.
- **Showcase mode.** A time-boxed override for showing the tank to visitors that expires and reverts on its own, without disturbing the automated schedule underneath.
- **Diagnostics bundle.** Equivalent to the original's support-file download, but with the device's own API token redacted before the archive is written rather than left in plaintext in a file intended for sharing.
- **Settings pages for things that previously meant SSH** — timezone selection, and toggling log shipping to an off-device receiver.
- **Two-tier password recovery** — an SSH-based reset plus a physical-button fallback.
- **Optional peer sync.** Schedule replication to a second fixture, authenticated with the receiving fixture's own token and disabled by default, since two tanks can legitimately want different lighting.
- **A memory ceiling** matching the original application's own safety margin, added once a side-by-side comparison showed the rewrite had no equivalent of its own.

Deployment

Deployment is a single supervised script, run by hand — nothing deploys automatically. It is deliberately conservative, because the fixtures light a live reef tank:

- Confirm the fixture is currently healthy before touching it.
- Back up the running application on the device itself, compressed.
- Stage the new version in a separate directory, **seeded first from the live schedule data, the original device scripts the init system calls on every boot, and the per-fixture configuration** — so none of that is lost in the swap.

- Refuse to continue if that staging copy is incomplete, rather than proceeding to an empty-data swap.
- Swap atomically, restart, then health-check.
- **Roll back automatically** to the backed-up version if the health check fails after restart, and report clearly if the rollback itself does not come up.

An off-device backup is taken independently before each rollout and verified restorable — not merely created — by test-extracting it and comparing a checksum against the running version.

Side-by-side trial

One fixture ran the rewrite while the other stayed on the original application, on the same tank, same schedule and same automation, for six days of direct comparison before the second unit was switched over on 10 October 2026.

	Original application	Rewrite
Availability checks	1614 / 1614	1614 / 1614
Failures	0	0
Automatic recoveries	0	0
Memory, mean	119.2 MB	114.3 MB
Memory, range	114.4 – 122.1 MB	112.4 – 115.5 MB
Drift over the window	–1.4 MB	+1.6 MB

Neither showed any sign of a memory leak: both drifted by under 2 MB over 135 samples, and not consistently in one direction. The rewrite averaged about 5 MB lower with less than half the variance. Switching the second unit freed roughly 20 MB of RAM and some flash on it, since the bundled dependency tree goes away.

|What this does and does not show. It shows the rewrite introduces no regressions. It does *not* show that the rewrite fixes the original crash described in the reliability notes — because the original application did not crash during the window either. A memory-limit correction had already addressed that, and the unit still running the original passed thirteen days of uptime against roughly six-day failures beforehand. The case for the rewrite is that it is leaner and maintainable by its owner, not that it rescued a crash.

Verification after rollout

A fixture answering HTTP does not prove it is lighting anything. Each rollout is therefore confirmed on three independent signals: the fixture's own PAR sensor reading compared against the other unit's (they tracked within 1.5%), a successful scheduled push from the external automation *after* the swap, and the preserved schedule data read back through the API.

Limitations

- Not open source, and not offered as a download. Built for two specific units.
- No vendor involvement. This is owner-side work on out-of-warranty hardware.
- The peer-sync feature is implemented and tested but left disabled on these units, because external automation already pushes the same schedule to both every fifteen minutes, which makes it redundant here.
- Anyone considering similar work should assume their own unit differs, and should read the deployment section's rollback behaviour as the minimum bar rather than a nicety.

This hardware is out of warranty. The author considers inspecting, patching and rewriting equipment he has paid for and runs himself to be entirely his own call and his own risk to carry. If any of this is useful background for your own out-of-warranty unit, the same applies to you.

Companion writeups: **ATI Straton Flex Reliability Notes** and **ATI Straton Flex Firmware Rewrite** at rooselaers.be/Acropora.