RING by Ringwood

Ring Reference

FOR PLANT ENGINEERS, IT, INTEGRATORS, AND ANYONE MAINTAINING THE SYSTEM

The lookup book: controller and firmware facts, network and endpoints, the PLC tag map, opcodes and ingredient codes, the data dictionary, cutover and rollback, compatibility, and every error code.

Generated 2026-09-05 from `docs/manual/reference`

Contents

Ring Reference

Who this is for

Plant engineers, plant IT, systems integrators, and anyone who has to keep a Ring installation running after the installer has driven away. It assumes you are comfortable with a controller and a network, and it does not assume you have read the other four books.

What this book is

The other four books are written to be *read*: they explain how Ring works and what to do. This one is written to be *looked up*. Every chapter is a table, a map, or an enumerated list, and every entry is expected to answer one question completely and then stop.

It exists because four questions kept being answered by reading source code:

1. *Which controller tag backs the thing on my screen?*
2. *What does my plant network have to allow, and what will Ring do to it?*
3. *What does this code mean?*
4. *What exactly happens on cutover day, and how do I go back?*

Status

This book is being assembled. Every chapter below currently carries a placeholder that names the material it will hold and points at the book that answers the question today. Nothing here is wrong; it is incomplete, and it says so on every page.

Chapters

Chapter	Answers
8. Compatibility matrix	What Ring runs on and what it talks to
9. Ring Remote View reference	The optional remote-viewing companion: pinned constants, listeners, files, accounts and rights, config keys, script parameters, bench evidence
90. Error and alarm code reference	Every code, with a stable anchor per entry

The other four books

Book	Read it for
Operator Manual	Running the plant from the screens
Installation Guide	Putting Ring on a plant PC and proving it works
Engineering Reference	How Ring is built and how to change it safely
Sales & Commercial Guide	The commercial story (internal)

Authoring rules specific to this book

The general contract in `docs/manual/README.md` applies. Two rules bind harder here:

1. **A reference entry with no verified source is worse than a missing entry.** If a value cannot be traced to code, an export, or a measurement, leave the row out and say the row is missing.
2. **Anchors are an API.** Anything another document, another book, or the app itself might deep-link to gets an explicit `{#id}` anchor on its heading. Reword the heading freely afterwards; never change the id.

1. PLC and controller facts

This is the "facts you'll otherwise get wrong" chapter. Everything on it has already cost somebody real time — a wrong assumption here has been the root cause of a shipped bug, a false bench-vs-live divergence, or a permanently dark indicator more than once in this project's history. Each fact below names the evidence it rests on; if you are about to act on a claim about the controller that is not on this page, verify it before you rely on it.

The controller

Ring talks to one controller: a **CompactLogix 1769-L36ERM**, firmware **v20.12**, at `172.22.103.10`. Three other EtherNet/IP-reachable devices sit on the same plant network — a PanelView at `.11`, a MicroLogix 1100 at `.12`, and the legacy RS-360 HMI PC at `.14` — and Ring has no relationship with any of them; they are listed here only so you don't mistake one for the box Ring writes to. A second **L36ERM** twin, used to prove write behaviour before it goes anywhere near the plant, sits on a bench network at `192.168.202.15`. Confusing the bench twin with the live controller is the single cheapest way to do real damage against this system — see reference/02 — Network and endpoints for the full address table and what plant IT needs to know about both.

(`docs/PLC_FACTS.md` §3; `docs/manual/engineering/01-system-overview.md`.)

The running program cannot be read as text

The program export taken directly off the live controller — `scripts/RS_3000_RUNNING_2026-06-09.L5K` — is **source-protected**. A count of `EncryptionConfig := 9` blocks against that file returns **88**, and every one of those 88 blocks is `EncodedType := ROUTINE`: 88 of the running program's 90 routines are opaque `ENCODED_DATA`, not readable ladder or Structured Text. Broken down by program:

- **`PROGRAM Tanks`** (the agitator and TVC program) — all 12 of 12 routines encrypted. There is no plaintext routine left in this program.
- **`PROGRAM Mixer`** — 37 of 39 routines encrypted. The two plaintext survivors are the physical-I/O cross-wire routines `IO_Inputs` and `IO_Outputs` — the rungs that connect a UDT bit to a physical output card, not the logic that decides the bit's value.
- The remaining 39 encoded routines are spread across `Alarm`, `fault_program`, `power_up_program`, `pre_run_main`, `PV_PC_Program` and `Simulate`.

The practical consequence: **no text-based conclusion about a ladder-side writer, or about whether a tag is live or dead, can be drawn from the running export** for anything inside `Tanks[]`, `TVC[]`, `TVC_Ctrl[]`, or `PC_Write_Integer` — most of the programs that touch them are encrypted, and the two plaintext I/O routines show only the last hop from a UDT bit to a physical point, not the logic upstream of it. Live tag observation on the controller (or the controls engineer's own ACD, which is source-available even where this L5K export is not) is the authority for what the running program actually does.

A second export, `scripts/RS_3000_2024_APR_22.L5K`, is **plaintext but two years stale** relative to the running controller. It is not proof of anything about today's behaviour, but it remains the **math authority for formula and band arithmetic** — `Formula_Calc`'s CASE logic is readable there and nowhere else — and it is what the repository's tag-audit tooling parses for rung-level provenance. Treat the two exports as different artifacts answering different questions: the running export for "is this tag reachable and readable today," the 2024 export for "what does the arithmetic actually do."

One consequence that follows directly: `TagManifestService`'s Live/Dead verdicts are sourced from the 2024 export, so they are **2024-sourced and non-authoritative** for the controller running today. And `scripts/code-tag-audit.md` classifying a tag as a "dead read" is a statement about the *parsed 2024 export having no writer for it* — not proof the tag is dead in the plant. A rising dead-read count still deserves attention (it usually means a new read was added against a tag nobody has confirmed is fed), but the word "dead" in that report is narrower than it sounds.

(`docs/PLC_FACTS.md` §§2, 9, 10 — each grep-verified against the named `.L5K` files, not carried from another document.)

A leading apostrophe in an L5K `ST_ROUTINE` is encoding, not a comment

This is the single most consequential misreading in this project's history, so it gets its own heading. Inside an `ST_ROUTINE` body, an L5K export prefixes **every line** with an apostrophe — that is file encoding, full stop, not a comment marker. A real comment in that Structured Text is `//`. The giveaway is a line that carries both, `'//JSR(Pump_To_Tank,0);`, which is what a genuinely disabled call actually looks like next to an encoded-but-live one.

Every claim of the shape "the ladder's `:=` writes are commented out," where the evidence offered is the leading apostrophe, is false. Two concrete things this got wrong, from the 2024 export where the routine is still plaintext: `ST_ROUTINE Agitator` drives `Tank_IO[N].O_Agitat := 1` / `:= 0` and forces it to 0 in several conditions — the ladder owns that bit, and it was never disabled, so "re-enable the ladder cycling" was never a task that existed. The same routine also clamps the agitator presets unconditionally, every scan: `Agt_on_pre` to 2..60 and `Agt_off_pre` to 5..60. An out-of-band value the HMI writes is silently corrected on the controller on the next scan — so a seeding test that reports the write as having DIVERGED from what it sent is normal behaviour, not a defect.

Separately, and still true: whether a task is *scheduled* is a different question from whether its source is *commented*. `MainTask InhibitTask := Yes` does appear in the 2024 export. The live plant runs batches, so its `MainTask` plainly is not inhibited in production — but "task inhibited" and "code commented out" are independent claims, and documents in this project's history repeatedly conflated them. Read `InhibitTask` as a program-load-time switch, not as evidence about what any individual routine's source does.

(`docs/PLC_FACTS.md` §1, citing `docs/production-readiness/WRITE_ENABLE_READINESS_2026-07-26.md` open questions OQ-5/OQ-8/OQ-13, each verified at rung level.)

The ladder-versus-Ring ownership boundary

`Tank_IO[N].O_Agitat` is **ladder-owned**. Ring never writes it, and the codebase's own write-surface census — `Ring.Tests/WriteSurfaceRegisterTests.cs` — enumerates every file under `Ring/` that can put a byte on the wire; nowhere in that register does Ring construct a write to `O_Agitat`. A code comment anywhere in this project suggesting Ring toggles the agitator is a phantom; trust the register, not the comment.

There is a second, sharper reason to leave that bit alone on the tanks Ring never actually touches for it: on the doser tanks, `ST_ROUTINE UD_Ctrl` round-robins four doser slots and **overwrites** `O_Agitat` from a remote-I/O status word every pass. On those tanks the bit is a status mirror, not a command — a write there is clobbered within about four PLC scans. The lookup table the routine uses to resolve which tank it is currently addressing (`N7_integers[20..23]`) is likewise not a write target: writing it re-points the routine at the wrong tank entirely. (`docs/PLC_FACTS.md` §5, from the same OQ-16 evidence chain.)

Two writes are **hard-closed on purpose** at the commissioning-hold layer, independent of the agitator question: `MainTvcWriteAuthorization.IsAuthorized` is a hard-coded `false`, and `TankTempModeWriteAuthorization.AuthorizedTankTempModeTanks` is a hard-coded empty list. Neither is populated anywhere in the codebase today. Do not populate either without a controls decision — see engineering/04 — The Write Path and Safety Model, §4.3 for the gate stack these two sit inside, and engineering/04's write-surface register for what any new writer — UDT member or bit — owes before it is allowed to exist.

More generally: what Ring *can* write is a short, explicit, tested list, not an inference from "the PLC has a tag for it." `Ring.Tests/WriteSurfaceRegisterTests.cs` is the authoritative enumeration — read it, don't guess from tag names.

UDT members are read by byte offset, and some hide behind alias names

libplctag cannot address a UDT member by its declared name in every case. Ring reads a UDT instance whole and extracts ordinary INT/REAL fields by offset — `Tanks[N].Formula_number`, `TVC[N].Preset_Temp`, and similar dotted paths resolve fine. But **bit-packed SINT members are exposed by the controller under a generated alias name** of the form `ZZZZZZZZZZ<truncated-suffix>` — for example `Tank_IO[N].ZZZZZZZZZZTank_I_O9` for the byte that packs `O_Agitat`, or `Tanks_c[N].ZZZZZZZZZZTank_c35` for the "tank installed" configuration byte. Reading the human-readable member name instead of the generated alias returns `ErrorBadParam` — a read that silently fails and ships a permanently dark indicator, or a write that silently no-ops. This has happened more than once in this project.

Twenty-two UDT types in the controller's `DATATYPE` block carry at least one `ZZZZZZZZZZ*` alias; as of the last full sweep only seven of those types have any reader or writer in the WPF code at all (`Tank`, `Tank_c`, `Tank_I_O`, `Pump_LA_I_O`, `TVC`, `TVC_Control`, `TVC_I_O`) — the remaining ~17 UDTs exist on the controller but are currently dark from Ring's side. A follow-on bench probe against every alias then in use came back clean: **zero new bugs found** — every aliased tag string in `Ring/Services/PLC/**` already used the correct generated name. If you add a reader or writer against any of the still-dark UDTs, the alias table in

`docs/reference/plc/UDT_ALIAS_SWEEP.md` gives the verbatim names to use — and the rule that generalizes past that table: **grep the L5K `DATATYPE` block for the verbatim alias string before trusting any UDT member name you haven't used before.**

(`docs/PLC_FACTS.md` §7; `docs/reference/plc/UDT_ALIAS_SWEEP.md` §§1–3, bench-verified 2026-06-03 against the `192.168.202.15` twin.)

Scan-rate and polling facts that shape how Ring reads

Ring's own poll cadences are a software design choice, not a controller constraint, but they are worth stating here because they explain the latency you'll see between a PLC-side change and its appearance on a Ring screen: background pollers read PLC tags on their own timers (roughly 250–500 ms for the fastest snapshot poller, 2000 ms for batch steps, storage tanks and batch start), and the UI consumes those snapshots on a separate, slower timer (roughly 500–1000 ms) — it never reads the PLC directly on the UI thread. A value can therefore be up to the sum of both intervals old on screen. See `docs/reference/plc/DESIGN_PATTERNS.md` for the full interval table and the per-poller/per-snapshot breakdown if you're adding a new PLC-driven screen.

Separately, the controller itself has a hard session ceiling worth knowing before you script anything against it: roughly **four to eight concurrent EtherNet/IP sessions per source IP**, with Ring's own pollers already consuming several of them — full detail, including exhaustion symptoms and recovery time, is in reference/02 — Network and endpoints.

Bench evidence does not automatically transfer to the plant

Tag parity between the bench L36ERM and the live one is an open question that has to be checked **per tank**, not per tag name — `enabl_agt` / `enabl_tvc` read true for tanks 1, 2, 8 and 9 only, so a tag can exist and behave correctly on both controllers for one tank and be inert on another. A 2026-06 survey that appeared to show `tank_split_number_*` tags absent from the live controller turned out to be a probe artifact: those tags are `Mixer` PROGRAM-scoped, the probe asked for the bare (unprefixed) names, and every controller-scoped name in the same sweep read fine. A later capture read the program-scoped names successfully straight off `172.22.103.10`. Nothing was ever actually shown to differ between the L5K and the live program on that point — the lesson that survives is procedural: **before recording a tag as absent, confirm whether it is program-scoped and re-probe with the `Program:<name>.` prefix.**

(`docs/PLC_FACTS.md` §4, retracted-and-corrected 2026-08-26.)

What Ring never writes — the short version

This chapter's version is the "meet the controller cold" summary; the authoritative, tested enumeration lives in the engineering book:

- **The agitator output**, `Tank_IO[N].O_Agitat` — ladder-owned; see above.

- `MainTvcWriteAuthorization` and `TankTempModeWriteAuthorization` — hard-closed commissioning holds; see above.
- **Anything not named in the write-surface register** — `Ring.Tests/WriteSurfaceRegisterTests.cs` enumerates every file that can put a byte on the wire and fails the build if an unregistered one appears. If a tag isn't reachable through a file on that register, Ring does not write it, no matter what a comment nearby claims.

For the full gate stack a write actually has to pass through — read-only guard, per-feature `Enable*` flags, the two commissioning holds, the endpoint guard, the heartbeat check, and the live demo re-check at the wire — see engineering/04 — The Write Path and Safety Model.

2. Network and endpoints

Who this is for. Plant IT, a network engineer standing up a new station, or anyone troubleshooting a dead PLC link from the network side rather than the application side.

What you'll learn. Every device on the reference site's control network and Ring's relationship to each; the ports and protocols Ring actually opens; the EtherNet/IP session budget and what exhausting it looks like; what a plant network must allow before Ring's core function works at all; what additionally needs internet access, and only if a site turns it on; and where to look when the link dies.

2.1 The reference site's network

Four devices share the plant control network Ring lives on, plus a bench twin used to prove write behaviour off the live box first:

Address	Device	Ring's relationship
`172.22.103.10`	CompactLogix **1769-L36ERM**, firmware **v20.12**	**The only controller Ring reads and writes.**
`172.22.103.11`	PanelView	None. Ring does not talk to it.
`172.22.103.12`	MicroLogix 1100	None. Ring does not talk to it.
`172.22.103.14`	Legacy RS-360 HMI PC	None in normal operation — it is the source machine for legacy data captures during migration work, not a device Ring reads live.
`172.22.103.99`	Engineering laptop (field-capture NIC)	Not a plant-permanent device; the address a laptop is set to when it is plugged into the control network for a field session.
`192.168.202.15`	Bench CompactLogix 1769-L36ERM twin	A separate physical controller on its own subnet where write behaviour is proven before it goes anywhere near `172.22.103.10`.

The plant subnet observed at the reference site is a `/25` (`172.22.103.0` – `172.22.103.127`, mask `255.255.255.128`) with gateway `172.22.103.1`. Treat that as this site's shape, not a universal Ring requirement — a new installation is whatever subnet the plant's own network design puts the controller on.

Ring's CIP path to the controller is `1,0` (backplane port 1, slot 0), resolved at runtime by `PlcConnectionConfig.GetPlcPath()` reading `PlcSettings.DefaultPath` from configuration on every call — not a build-time constant. A different path only matters if the controller sits behind a different CPU slot or a routed CIP hop; the reference site does not.

Sources: `docs/PLC_FACTS.md` §3 (the device table, four plant addresses plus the bench twin); `docs/production-readiness/FIELD_CHEATSHEET.md` and `docs/production-readiness/FIELD_SESSION_RUNBOOK_2026-06-29.md` (the laptop's `172.22.103.99/25` field NIC address and gateway); `docs/manual/engineering/03-plc-communications.md` §3.1 (the `Path` / `GetPlcPath()` behaviour).

The superseded IP

An early cutover-preparation document used `10.170.35.35` as a placeholder production IP before the site's real network was surveyed. **That address is wrong.** It was removed from the cutover documents once the actual plant network (`172.22.103.0/25`) was confirmed on-site, and it never corresponded to a real device at the reference plant. If you find `10.170.35.35` in an older document or a stale config file, it is a leftover placeholder, not a controller to reach. (`docs/reference/plc/PLC_BENCH_FINDINGS.md`, verdict table: "Cutover docs — Annotated with 2026-06-02 findings; placeholder `10.170.35.35` removed.")

2.2 Ports and protocols

Protocol	Port	Direction	Required?	What it's for
EtherNet/IP explicit messaging (CIP)	**TCP 44818**	Outbound, Ring → controller	**Yes — core function.**	Every tag read and write Ring performs. `PlcSettings.DefaultPort` is `44818` and is described in the appsettings reference as "only change if the plant has a non-standard NAT in front of the PLC — very rare."
EtherNet/IP List Identity discovery	**UDP 44818**	Outbound broadcast, Ring's field tooling → subnet	Only for discovery tooling, not for normal operation	The zero-config way to find a controller's IP when it's unknown: a single connectionless UDP broadcast (CIP command `0x0063`) that every Logix controller answers with its IP, product name, serial and revision. Ring itself connects by configured IP; discovery is a setup-time tool (`plc-discovery/Find-EipDevices.ps1`), read-only by design — it never opens a CIP session or writes a tag.

Protocol	Port	Direction	Required?	What it's for
SMTP	Site-configured (typically TCP 587/465/25)	Outbound, Ring → the plant's mail relay	Only if alarm email is turned on	`AlarmEscalation.Mode` ships as `"None"` (log-only) in `Ring/Config/appsettings.json`. Nothing is sent, and no outbound mail port is needed, until a site deliberately configures an SMTP host and switches the mode to `"Smtp"`.
HTTPS (MeshCentral / Cloudflare Tunnel)	TCP 443, optionally UDP 7844	Outbound, the HMI PC → Cloudflare's edge	Only if the optional remote-view package is installed	See §2.4 below. Not installed, not required, and touches neither the PLC network nor Ring's own process when absent.
Intel AMT MPS (MeshCentral remote-view v1)	TCP 4433	—	Not required	Bench-verified 2026-09-04: in the shipped **LAN-only** mode, MeshCentral never starts this listener at all — it is not merely "bound to loopback," it is **not listening on any interface**. Only relevant if a site turned LAN-only off, which the shipped install does not do.
UDP	Any	Inbound, created automatically on all three firewall profiles	Not required — safe to disable or remove	The Mesh Agent's own installer, not `Install-RemoteView.ps1`, creates this rule ("Mesh Agent WebRTC Traffic") for `MeshAgent.exe` on all three firewall profiles. The server runs with `webRTC: false`, so nothing ever uses it — disable it (`Disable-NetFirewallRule -DisplayName 'Mesh Agent WebRTC Traffic'`) or remove it; `Uninstall-RemoteView.ps1` removes it along with the rest of the agent.

Everything else Ring needs is local to the PC: the SQLite database, the config files, the log directory. There is no inbound port a plant firewall has to open **for Ring itself** — the connections above are all Ring or its optional companion package initiating outbound.

The install runbook states this as a firewall instruction for whoever is provisioning the machine: **"Whitelist outbound EtherNet/IP UDP/44818 + TCP/44818 on the local firewall"** (`docs/production-readiness/12_INSTALL_RUNBOOK.md`, responsibilities table). The first-start troubleshooting page gives the

same fact from the failure side: a link that pings but won't connect on 44818 means "either Ring's own Windows firewall exception is missing, or plant IT needs to open TCP port 44818 to the PLC" (`docs/manual/installer/03-first-start.md`).

Sources: `docs/reference/plc/APPSETTINGS_REFERENCE.md` (`PlcSettings.DefaultPort` row); `docs/production-readiness/12_INSTALL_RUNBOOK.md` (firewall responsibility, `DefaultPort: 44818` sample config); `docs/manual/installer/03-first-start.md` (TCP 44818 troubleshooting line); `plc-discovery/Find-EipDevices.ps1` (UDP 44818 List Identity discovery, read-only); `docs/manual/engineering/09-build-deploy-release.md` §9.6 (SMTP and remote-view as the only two internet-requiring features, both opt-in).

2.3 The EtherNet/IP session budget

A CompactLogix accepts only a small number of concurrent EtherNet/IP sessions from any one source IP — observed at roughly **four to eight sessions per client IP** on the bench controller. Ring's own polling coordinator already runs seven pollers against the live controller (`engineering/03 — PLC Communications` §3.2), each opening and disposing a fresh `Tag` per read rather than holding a session pool — session reuse is entirely libplctag's business, not something Ring's own code manages. That is why poller startup is deliberately staggered (0 ms, 150 ms, 300 ms, 450 ms, 600 ms, 750 ms delays across the seven pollers) rather than firing all at once.

Exhaustion symptom: running an additional tool against the same controller from the **same source IP** — a second probe script, RSLinx browsing, a Studio 5000 "Who Active" session left open — can starve Ring's own pollers of sessions. The visible result is EtherNet/IP timeouts (`ErrorTimeout` bursts) that look like a comms problem but are really session contention, and on the UI side it can present as the blanket "stale everywhere at once" pattern described in `engineering/11 — Diagnostics and Field Triage` §11.3. A *different* source IP hitting the controller does not compete for the same budget — the cap is per-IP, so Ring on the HMI PC and a laptop running a field script on its own IP do not starve each other.

Recovery is automatic and takes about 30 seconds. Once the extra tool closes its sessions, the controller frees them and Ring's pollers resume without any restart. Every field script in this repository (`Find-EipDevices.ps1`, `Probe-TagInventory.ps1`, `Test-PlcWrite.ps1`) is documented as read-only or short-lived specifically because of this budget.

Coexistence advice for a live station: Studio 5000 "Who Active" browsing, a PanelView on its own IP, and Ring can all run against the same controller simultaneously — they are different source IPs (or, for Studio 5000 on the same PC as Ring, still count against the shared budget for that IP). The practical rule for anyone opening a second connection to `172.22.103.10` from the same machine Ring runs on: close it when you're done, and expect a transient staleness window while sessions rebalance, not a hard fault.

Sources: `docs/PLC_FACTS.md` §8 ("The controller has very few EtherNet/IP sessions" — 4–8 per source IP, self-clears in ~30 s, bench-observed); `docs/manual/engineering/03-plc-communications.md` §3.1–§3.2 (no C#-level session pool; per-tag `Tag` construct-and-dispose; the seven pollers and their staggered startup

delays); `docs/manual/engineering/11-diagnostics-and-field-triage.md` §11.3 and its symptom-index row for "PLC comms degrade while a capture/probe script is running" (self-heals in ~30 s once the extra sessions close).

2.4 Remote access posture

Remote access is **entirely optional** and ships as a separate companion package (`remote-view/`), not part of Ring itself. Two tiers exist:

v1 — on-site / LAN viewing. A MeshCentral-based viewer runs on the HMI PC and is reachable from other machines on the same LAN at `https://<hmi-pc>:8443/`. Gates: per-user accounts, mandatory two-factor (`force2factor`), a view-only-by-default rights mask with control granted by named exception, a consent prompt at the panel before a session starts, and session recording. Governed by `docs/production-readiness/REMOTE_ACCESS_RUNBOOK.md`.

v2 — off-site viewing, additive on top of v1. A `cloudflared` service on the HMI PC dials **outbound only** to Cloudflare (TCP 443, optionally UDP 7844 for QUIC) and Cloudflare's edge (Cloudflare Access) authenticates every visitor — with MFA — before a single byte reaches the plant. **No inbound firewall rule is created on any NIC.** Every v1 gate still applies behind Access, so an off-site user authenticates twice: once at Access, once at MeshCentral. Governed by `docs/production-readiness/REMOTE_ACCESS_V2_RUNBOOK.md` and the design rationale in `REMOTE_ACCESS_V2_CLOUDFLARE.md`.

What plant IT is actually being asked to approve, if a site wants this:

- Nothing inbound, ever — the model is deliberately outbound-only in both tiers (the design document names port-forwarding an HMI to the internet as the pattern behind real published OT incidents, and builds v2 specifically to avoid it).
- One outbound HTTPS path from the HMI PC (v2 only) — TCP 443 to Cloudflare, optionally UDP 7844.
- A second Windows session sharing the same physical PC as Ring. The documented engineering hazard here is **not** networking — it's that Ring's single-instance protection is per-session, not per-machine, so a second logged-in session (RDP, MeshCentral, or a physically present engineering laptop) could, in principle, launch a second write-enabled Ring. The standing mitigation is procedural: check Task Manager → Users for a second `Ring.exe` before assuming only one write path is live.

Two more things worth knowing before you install it: **browse to it by the hostname the installer's shortcut points at, not a bare IP address** — the server checks the browser's Origin against a fixed list of accepted names, and an address that isn't on that list is refused with "Invalid origin in HTTP request." And **the TLS certificate name it's installed under must contain a dot** (a real hostname, not a bare computer name) — MeshCentral only offers a way to enrol two-factor authentication when the certificate name has a dot in it, and two-factor is mandatory, so a dotless name locks everyone out of every device. For how to install it and what these two rules mean in practice, see the installer's Ring Remote View chapter and its off-site companion for the Cloudflare Tunnel option; for what the person at the panel sees, see the operator manual's Remote viewing chapter.

Status as of this writing (2026-09-04): on a developer laptop, the v1 installer and uninstaller were both run for real, end to end. The first attempt hit two blocking defects in the installer itself — an uppercase certificate name that made the browser's Origin check fail with `invalidorigin`, and a dotless certificate name that left no way to enrol two-factor authentication — both fixed the same day before the walkthrough below could complete; see engineering §9.5 for what changed. With that fix in place the installer completed (`INSTALL OK`), the server's own per-account rights masks were proven server-side rather than merely hidden in the UI — a restricted account was denied file transfer, remote command execution, and an interactive shell, while the site administrator account was allowed, as expected — and the uninstaller removed everything it had created. The v2 staged quick-tunnel smoke test also passed. None of that is the same as being ready for a site: it has **not been installed at any plant**, and the full penetration test on a clean VM, the live Cloudflare Access policy, and an off-site smoke test against a real Zero Trust tunnel are all still outstanding hard gates. Treat "bench-verified on a developer laptop" and "installed and signed off at a site" as two different questions, and check the runbook's own checklist rather than this summary for current status.

Sources: `docs/production-readiness/REMOTE_ACCESS_RUNBOOK.md` (v1: LAN scope, account/2FA/consent/recording gates); `docs/production-readiness/REMOTE_ACCESS_V2_RUNBOOK.md` (v2: outbound-only tunnel, Cloudflare Access, dual authentication, sign-off checklist and outstanding-gates status); `docs/manual/engineering/09-build-deploy-release.md` §9.5–§9.6 (the single-writer hazard, the two internet-requiring features, and the "check Task Manager → Users for a second `Ring.exe`" mitigation).

2.5 What Ring needs from your network — the sign-off list

For core operation (batching, screens, history — everything except email alerts and remote viewing), a plant network must provide:

- ☐ **A reachable IP for the controller**, static or a fixed DHCP reservation. Ring is configured with one IP (`PlcSettings.DefaultIpAddress`) and does not re-discover it at runtime if it changes.
- ☐ **A path from the HMI PC to that IP** — same subnet, or routed with nothing in between that strips or blocks EtherNet/IP.
- ☐ **Outbound TCP 44818 and UDP 44818 allowed** from the HMI PC toward the controller (or the whole control subnet, if using discovery tooling during setup).
- ☐ **No internet access required** for core operation. A plant PC with the PLC path working and no other features configured has no reason to reach the internet at all.
- ☐ **If email alerts will be used:** outbound access from the HMI PC to whatever SMTP host the plant chooses, on the port that host requires. Not needed until a site sets `AlarmEscalation.Mode` to `"Smtp"`.
- ☐ **If remote viewing will be used:** outbound TCP 443 (and ideally UDP 7844. from the HMI PC to Cloudflare, decided and signed off per the v2 runbook's own checklist — this is a deliberate, documented decision, not a default.

For IT approval — summary table

Question	Answer
What does Ring connect to?	One CompactLogix controller, by configured IP, over EtherNet/IP (CIP).
What port(s)?	TCP 44818 (reads/writes); UDP 44818 only for setup-time discovery tooling.
Inbound ports required?	None.
Internet required for core function?	No.
What needs internet, and is it on by default?	Email alerts (off by default) and the optional remote-view package (not installed by default) — both opt-in.
What's the failure mode if the network is wrong?	Ring shows a disconnected/stale state on screen and in its logs; it does not silently show wrong data (see §2.6).
Anything to whitelist beyond the PLC path?	Only the SMTP host (if email is enabled) and Cloudflare's edge (if remote-view v2 is enabled).

2.6 What to check when the link dies

In rough order of likelihood:

1. **Cable and link light.** Physical layer first, always.
2. **Subnet and IP.** Confirm the HMI PC's own address, mask and gateway still match the plant network's design, and that `PlcSettings.DefaultIpAddress` still points at `172.22.103.10` (or the site's equivalent) rather than a stale bench or field-capture value.
3. **Firewall.** Confirm TCP 44818 outbound to the controller is still allowed, both on the Windows firewall on the HMI PC and on any managed switch or plant firewall between them — this is the single most common cause the installer-level troubleshooting page names.
4. **Session exhaustion.** If the link was fine moments ago and several people or scripts are touching the same controller from the same source IP, wait roughly 30 seconds and check again before escalating — see §2.3.
5. **The right diagnosis for "stale."** A single dimmed value is ordinary freshness gating on one tag; every screen going stale at once is the heartbeat/session layer, not a per-tag problem. The engineering diagnostics chapter's symptom index is the deeper reference: [`engineering/11 — Diagnostics and Field Triage` §11.1–§11.3`](../engineering/11-diagnostics-and-field-triage.md#111-symptom-index).

2.7 For IT: antivirus and security notes

Exclusion paths. Antivirus/EDR real-time scanning and any file-lock-on-read behaviour can slow or trip over Ring's own writes. Exclude:

- The Ring install directory itself, and inside it `RingwoodDatabase.db` along with its `-wal`/`-shm` sidecars, which live beside `Ring.exe` (`AppDomain.CurrentDomain.BaseDirectory`, not a data folder).
- `%ProgramData%\Ring\` — logs, database backups, support bundles, readiness-run history, and the rolling config-history snapshots (`Logger.cs`, `DatabaseHealthService.cs`, `SupportBundleService.cs`, `Ring/Services/Readiness/ReadinessRunLog.cs`, `Ring/Infrastructure/Configuration/ConfigSnapshotService.cs`).
- `%ProgramData%\Ringwood\Ring\` — the two hash-chained journals, `ui-audit\ui-audit.jsonl` and `plc-write-evidence.jsonl` (`Ring/Services/Audit/UiAuditJournal.cs`, `Ring/Services/PLC/PlcWriteEvidenceJournal.cs`). Note the different vendor folder name from the paths above — both are real and both matter.

How passwords are stored at rest. The Supervisor and Administrator passwords are written in **plain text** into `Config\appsettings.json` (or its per-machine `appsettings.local.json` overlay) — first by the commissioning wizard's Passwords step, and again every time Setup → Quick Setup or the credential-rotation dialog changes them (`CredentialRotationService.PersistAtomically` writes the new value straight into the `Authentication` JSON node; nothing in that path encrypts or DPAPI-protects it). The one credential Ring does DPAPI-protect at rest is the SMTP mail password, via `SmtpPasswordProtector`. Practical recommendation: treat `appsettings*.json` as sensitive — restrict its NTFS permissions to the account(s) that run Ring and to the IT staff who administer it, the same way you would a file holding a plaintext secret, and rotate the Supervisor/Admin passwords through Setup rather than hand- editing the file.

Changing the PLC's IP address.

1. Open **Setup** → **Quick Setup** (or the commissioning wizard's PLC Connection step) and enter the new address in the PLC IP field.
2. Save. The change is written to `PlcSettings.DefaultIpAddress` in `appsettings.json` immediately, but Ring keeps talking to the *old* address for the rest of this session — the screen's own save confirmation says so: **"Restart Ring for the PLC address … to take effect."** Restart Ring to reconnect at the new address.
3. An in-flight batch is not affected by any of this on the controller side — batch sequencing runs in the PLC's own ladder logic, not in Ring, so restarting Ring (or having pointed it at the wrong controller in the meantime) does not pause, resume, or lose the batch. It does mean Ring is blind to that batch's progress until it is restarted pointed at the right controller — do this between batches, not while one you care about is running, so nobody loses visibility mid-run.

3. PLC tag reference

This chapter maps controller tags to screen elements: which tag backs which number or indicator on which screen, which tags Ring writes and under what gate, and the array/alias conventions you need to read any of it correctly. Read `1. PLC and controller facts` first — in particular the fact that a "commented-out" ladder line in the 2024 L5K export is **not proof a tag is dead**, and that the running program's tanks-and-mixer ladder is source-protected and cannot be read as text at all. Every "is this live" claim below is qualified accordingly; where the underlying investigation predates that correction, this chapter says so rather than repeating it.

3.0 How to read this chapter honestly

Two independent per-screen investigations in this repository (`docs/reference/plc/STORAGE_TANK_PLC_TAGS.md`, `TVC_PLC_TAGS.md`, `USE_TANK_PLC_TAGS.md`) were written **before** `docs/PLC_FACTS.md` fact 1 was established, and they reason from "the assignment to this tag is prefixed with `'` in the L5K, therefore it is commented out and dead." Fact 1 shows that a leading apostrophe on every line of an `ST_ROUTINE` body is the export's own line encoding, not a comment marker — so every "dead" or "starved" verdict in those documents that rests on that reading is unproven, not refuted. This chapter draws the **tag names, UDT shapes, and bit positions** from those documents (that part is a straight read of the L5K `DATATYPE` block and is not affected by the correction) but does **not** repeat their liveness verdicts. Where current code has since been built and shipped against a tag (there is a reader/writer service and it is registered with a poller), that is the strongest evidence available that the tag is live enough to be useful, and this chapter says so. Where a tag is still only described in a scoping document with no reader built, this chapter says that plainly instead of asserting a verdict either way.

3.1 Two layers of controller-exposed data

The controller exposes process data to the HMI two ways, and different screens use different ones.

Per-tank UDT fields — `Tanks[N]` (UDT `Tank`), `Tank_IO[N]` (UDT `Tank_I_O`), `Tanks_c[N]` (UDT `Tank_c`, supervisor configuration), `TVC[N]`/`TVC_Ctrl[N]`/`TVC_IO[N]` and the pump UDTs `Pump_LA_IO[N]` / `Pump_Tank_I_O[N]`. Reads are per-field (libplctag cannot address a UDT member directly — Ring reads the whole UDT and extracts fields by offset; `docs/PLC_FACTS.md` fact 7), canonical, and slightly more expensive per read than the bulk mirrors below.

Bulk-mapped HMI mirror arrays — four flat block tags kept for one-shot HMI consumption: `PC_Read_Integer` (INT[480]), `PC_Read_Float` (REAL[180]), `PC_Display_Outputs` (BOOL[224]), `PC_Write_Integer` (INT[480], PC→PLC direction). Ring's fast block (`PcReadFloatSnapshot`, `Ring/Services/PLC/PcReadFloatSnapshot.cs`) reads `PC_Read_Float[0..34]` in one call: indices 0–29 are batch actuals, 30–32 are the mixer/Make-Ready weight/temperature/borax-caustic channels, and 33–34

are the mixer viscometer channels (§3.4). The heartbeat lives at `PC_Read_Integer[0]`, polled every 1500 ms by its own dedicated timer (`Ring/Services/PLC/PcReadIntegerPoller.cs`) — kept short and separate from the bulk poll specifically so a disconnect is felt quickly.

`docs/reference/plc/PLC_TAG_REFERENCE.md` documents further offsets in `PC_Read_Float` (50–55 TVC temps, 60–71 tank temperature mirrors, 72–83 level mirrors, 84–95 starch-weight mirrors) as a cheaper alternative to the per-UDT reads for a future perf pass; Ring does not read those offsets today, so treat them as documented-but-unused, not as a second live source.

3.2 Array and index conventions

Tank roster indices are 1-based and roster-driven, not a fixed column mapping. Two older per-screen documents (`STORAGE_TANK_PLC_TAGS.md`) describe a hardcoded "UI column I reads `Tanks[I-1]`" mapping. That has been superseded: `StorageTankGroupPoller` and `UseTankTagReaderService` both resolve which `Tanks[N]` slot backs which screen slot through the **Tank Roster** (`Ring/Services/Roster/`), which discovers a tank's role by reading `Tanks_c[1..12].ZZZZZZZZZZTank_c35` — a bit-packed SINT alias where bit 0/1 mean "storage" and bits 2–5 mean "one of four small-tank types" (i.e. a use tank). `PlcIndex` values handed to the readers are the controller's own 1-based `Tanks[N]` slot number (1..12); a slot with `PlcIndex` 0 or disabled in the roster is not polled at all (`StorageTankGroupPoller.cs`). Roster changes require a restart to take effect on the pollers. Do not assume storage occupies slots 1–4 and use tanks occupy 5–8 (or any other fixed split) on a given plant — read the roster.

TVC slots are 1:1 with storage-tank slots. `TVC_PLC_TAGS.md` resolved this (its open question #4): `TVC[N]`, `TVC_Ctrl[N]`, `TVC_IO[N]` for the same `N` as `Tanks[N]`, confirmed via the `Tanks_c[N].TVC_Installed` bit and the `TVC_c[9]` initialiser alignment. `TVCTagReaderService.cs` and `TVCControlPlcService.cs` both use this mapping.

UDT arrays generally reserve index 0. The `TVC`/`TVC_c`/`TVC_Control`/ `TVC_I_O` arrays are declared with 9 elements; indices 1–8 are used and 0 is reserved. `Tanks[13]` similarly leaves slot 0 unused in the roster model — valid slots are 1–12 (the `Tank_c` UDT commissioning bits are read across that same 1–12 range in `UseTankTagReaderService.DiscoverUseTankSlotsAsync`).

3.3 Bit-packed types and the alias convention

Several UDT members are bit fields inside a single SINT, and the controller exposes that SINT under a **generated alias name**, not the member name you would guess from the `DATATYPE` block — `ZZZZZZZZZZ<parent-tag-suffix>`. Reading the individual bit-named member directly (e.g. `Tank_c35` instead of `Tanks_c[N].ZZZZZZZZZZTank_c35`) returns `ErrorBadParam` on every controller this has been checked against (`UseTankTagReaderService.cs`, bench-verified 2026-06-02). `1. PLC and controller facts` carries the full alias-naming rule and points at `docs/reference/plc/UDT_ALIAS_SWEEP.md` for the complete sweep; this chapter only lists the aliases actually in use on the read/write surfaces below.

Alias	Parent UDT array element	Bits Ring uses
`Tank_IO[N].ZZZZZZZZZZTank_I_O9`	`Tank_I_O[N]`, output byte	0 `O_Fill_valv`, 1 `O_Agitat`, 2 `O_Heat`, 3 `O_Cool`, 4/5/6 `O_Resin_{1,2,3}_dischrg_valv`, 7 `O_Resin_1_fill_valv`
`Tank_IO[N].ZZZZZZZZZZTank_I_O18`	`Tank_I_O[N]`, output byte	0/1 `O_Resin_{2,3}_fill_valv`
`Tanks_c[N].ZZZZZZZZZZTank_c35`	`Tank_c[N]` config	0/1 `Storage_tank`/`Storage_tank2`, 2–5 `Small_tank_type_{A..D}` (role discovery)
`Tanks_c[N].ZZZZZZZZZZTank_c17`	`Tank_c[N]` config	0 `tank_installed` (write-gate commissioning input)
`Tanks_c[N].ZZZZZZZZZZTank_c44`	`Tank_c[N]` config	5/6/7 `Liquid_{1,2,3}_installed` (write-gate commissioning input)
`TVC_Ctrl[N].ZZZZZZZZZZTVC_Contro 0`	`TVC_Control[N]`	0 `enabled`, 3 `heat_over_temp`, 4 `heat_under_temp`, 5 `cool_over_temp`, 6 `cool_under_temp`
`TVC_IO[N].ZZZZZZZZZZTVC_I_O0`	`TVC_I_O[N]`	0 `I_Low_lev`, 1 `I_Heat_flt`, 2 `O_Fill_valv`, 3 `O_Heat`, 4 `O_Cool`
`Pump_LA_IO[N].ZZZZZZZZZZPump_LA_ I_0`	`Pump_LA_I_O[N]`	1 `O_Pump`

Two gotchas the code comments call out explicitly: `Tanks_c[N].LA1_pump` (the pointer from a use-tank slot into `Pump_LA_IO[]`) is declared **INT**, not SINT — a prior SINT read happened to work only because the values were small enough to decode as the low byte, and was corrected (`UseTankTagReaderService.cs`). And the alias for the pump output byte ends in `_0` (`ZZZZZZZZZZPump_LA_I_0`), not `_O9`/`_O18` like the tank output bytes — the `Pump_LA_I_O` and `Tank_I_O` UDTs are different shapes and the naming looks similar enough to transpose by accident.

3.4 Read surface, by screen

Every row below names the reader/poller service that is actually built and registered, so this table describes what Ring reads today, not a proposal.

Storage Tank Group (Main Screen → Storage Tank Group, and the Dashboard tank tiles)

Screen element	PLC tag	Type	Reader
Tank volume	`Tanks[N].Level`, fallback `LA_Weight`	REAL	`StorageTankTagReaderSer vice.ReadLevelAsync`

Screen element	PLC tag	Type	Reader
Fill % denominator	`Tanks_c[N].nominal_Size`	REAL, cached (long TTL)	`StorageTankTagReaderService` (fill % = Level ÷ nominal size — legacy parity, `PARITY_AUDIT_2026-06-10` §3.1)
Formula number	`Tanks[N].Formula_number`	INT	`StorageTankTagReaderService.ReadFormulaNumberAsync`
Current temperature	`Tanks[N].Current_Temp`	INT (not REAL)	`StorageTankTagReaderService.ReadCurrentTempAsync`
Requested level	`Tanks[N].Request_level`	INT	`StorageTankTagReaderService.ReadRequestLevelAsync`
Preset temperature	`Tanks[N].Preset_Temp`	INT	`StorageTankTagReaderService.ReadPresetTempAsync`
Is filling	`Tank_IO[N].ZZZZZZZZZZTank_I_O9` bit 0 (`O_Fill_valv`)	bit	`StorageTankTagReaderService` (one SINT read, masked for all three booleans below)
Is agitating	same SINT, bit 1 (`O_Agitat`)	bit	″
Is transferring	same SINT, bits 4|5|6 (`O_Resin_{1,2,3}_dischrg_valv`, OR'd)	bit	″

Poll cadence: 2 s (`StorageTankGroupPoller`), driven by the roster's storage slots (§3.2), not a fixed 1–3/1–4 range.

Use Tank Group (Main Screen → Use Tank Group, and the per-tank MF/drill-down windows)

Slot discovery and pump-index resolution happen once at startup/roster change (§3.2); the per-tick reads are the rows below, once per discovered slot.

Screen element	PLC tag	Type	Reader
Level	`Tanks[useIdx].Level`, fallback `LA_Weight`	REAL	`UseTankTagReaderService.ReadLevelAsync`
Formula number	`Tanks[useIdx].Formula_number`	INT	`ReadFormulaNumberAsync`
Current temp	`Tanks[useIdx].Current_Temp`	INT	`ReadCurrentTempAsync`
Requested level	`Tanks[useIdx].Request_level`	INT	`ReadRequestLevelAsync`

Screen element	PLC tag	Type	Reader
Status code	`Tanks[useIdx].Status`	INT	read; the L5K `Description` gives `0=none 1=idle 2=in progress 3=in queue 4=controlled remotely` but the legacy string table behind those numbers was never recovered from the source tree available (`USE_TANK_PLC_TAGS.md` open question #2, unresolved)
Source storage tank	`Tanks[useIdx].Fill_from_tk`	INT	`ReadFillFromTankAsync`-style read, → "Storage Tank N"
Is adding additive	`Tanks[useIdx].Liq_1_status` (`Liq_2`/`Liq_3` also read; `Liq_3_status` is a confirmed **orphan** — no L5K writer — so that call short-circuits to null)	INT, `1 = adding resin`	code comment, `UseTankTagReaderService.cs:429,613`
Is inlet valve open	`Tank_IO[useIdx].ZZZZZZZZZZTank_I_09` bit 0	bit	one SINT read, masked for the four bits below too
Is agitating	same SINT, bit 1	bit	″
Is heating	same SINT, bit 2 (`O_Heat`)	bit	″ — see the write-surface note below: this bit has **no confirmed L5K writer** per the 2026-05-26 orphan audit cited in the code comment; treat the indicator as best-effort until re-confirmed
Is transferring (discharge)	same SINT, bits 4|5|6	bit	″
Is adhesive-supply valve open	same SINT, bit 7 (`O_Resin_1_fill_valv`)	bit	″
Resin pump	`Pump_LA_IO[pumpIdx].ZZZZZZZZZPump_LA_I_0` bit 1 (`O_Pump`), `pumpIdx` from `Tanks_c[useIdx].LA1_pump`	bit	`UseTankTagReaderService.ReadPumpStatusAsync`-style read; bench-confirmed alias 2026-06-02

The resin type labels (`ResinTypeSelected{N}`) are **not** read from the PLC at all — they are a supervisor-configured value stored in Ring, not controller state.

TVC (Main Screen → TVC card, and the four per-tank TVC control windows)

`TVCTagReaderService.cs` exists and is registered (`TVCPoller.cs`), which is stronger evidence of liveness than the scoping document `TVC_PLC_TAGS.md` alone — but the reader's own doc comment is explicit that whether these tags are live on the **running, source-protected** program is still **unconfirmed**, because the routine that would settle it is one of the 88 encrypted blocks (`docs/PLC_FACTS.md` fact 10; `TVCTagReaderService.cs` class comment). A failed/null read leaves the ViewModel on its hardcoded defaults rather than showing a false zero.

Screen element	PLC tag	Type	Notes
Actual temperature	`TVC[N].Current_Temp`	INT	
Preset temperature	`TVC[N].Preset_Temp`	INT	Live 2026-08-20 field capture recorded 90/120/125/125 across slots 1–4 against a 125/120/125/125 export initialiser — slot 1 has been retuned in the field; quote a fresh capture, not the export (`TVC_WRITE_TAG_MAPPING.md`).
Temperature mode	`TVC[N].Temp_mode`	INT	0 = off, 1 = heat, 2 = cool (per the write-side mapping, §3.5)
TVC enabled	`TVC_Ctrl[N].ZZZZZZZZZZT` `VC_Contro0` bit 0	bit	The same 2026-08-20 capture recorded this bit **0 on every slot, all 713 samples**, while `Temp_mode` was non-zero and the plant was running — i.e. this bit does not currently track whether the sequencer is doing anything. Treat it as unreliable for an "actively cycling" indicator until re-confirmed.
Heater fault flags	same SINT, bits 3–6	bit	over/under-temp for heat and cool

Screen element	PLC tag	Type	Notes
Is heating (`TVC_heat.png` animation)	`TVC_IO[N].ZZZZZZZZZZTVC_I_O0` bit 3 (`O_Heat`)	bit	Relabelled from an originally-proposed "circulating pump" indicator to this honest heat-call signal — there is no dedicated pump-run tag in the export, and a continuously-running circulation pump would make a pump-derived indicator read "Standby" while actually running (`TVC_PLC_TAGS.md` open question #3 resolution note).
Low-level / heat-fault input, fill valve	same SINT, bits 0/1/2	bit	read but not currently surfaced in the UI

Make Ready Tank / batch formula steps

Screen element	PLC tag	Type	Reader
Current step	`current_step`	INT (0 = no batch running)	`BatchTagReaderService.ReadCurrentStep`
Preset operation per step	`Batch_Current_Preset_Operation[0..30]`	INT array	`BatchTagReaderService.TryReadAllStepArrays` (whole-array read, not per-index)
Preset amount per step	`Batch_Current_Preset_Amount[0..30]`	REAL array	″
Actual amount per step	`Batch_Current_Actual_Amount[0..30]`	REAL array	″
Preset mix time per step	`Batch_Current_Preset_Mix[0..30]`	INT array	″

Operation codes are mapped to display text via `BatchOperationMapper` — the full opcode table with the L5K/Dutch-legacy cross-reference lives in `4. Opcode and ingredient map`, not here. A single failed array read keeps the previous array's values rather than blanking the whole grid (`BatchTagReaderService.cs`, "last-good" cache, timestamped and dropped at batch start/end so one batch's data can never bleed into the next). Older analysis in this same source tree (`BATCH_TAGS_ANALYSIS.md`) predates this implementation and describes these tags as unread; that finding is superseded by the code above. The `Batch_Previous_{1,2,3}_*` tags documented alongside `Batch_Current_*` in that same analysis are **not read anywhere in the current codebase** — a grep for "Batch_Previous" against `Ring/` returns no hits — so treat their existence in the PLC configuration as plausible (the tag names are documented) but their consumption as not built.

Heartbeat and viscometer (cross-screen)

Screen element	PLC tag	Type	Notes
Connection heartbeat	`PC_Read_Integer[0]`	INT	Polled every 1500 ms on its own timer, separately from the bulk data poll, specifically so a disconnect registers quickly (`PcReadIntegerPoller.cs`). Drives `PlcHeartbeatConnectionTracker`, which in turn gates every PLC write (§3.5).
Mixer/Make-Ready weight	`PC_Read_Float[30]`	REAL	fast block, one read for [0..34]
Mixer temperature	`PC_Read_Float[31]`	REAL	″
Borax/caustic scale weight	`PC_Read_Float[32]`	REAL	″
Viscosity result	`PC_Read_Float[33]`	REAL	The **PLC-computed** viscosity result — this is the number an operator reads and records. There is no dedicated viscometer tag on the TVC; see the operator book for the honesty note on acceptance bands.
Raw Cambridge viscometer input	`PC_Read_Float[34]`	REAL	Raw analog input, kept for provenance alongside the computed result.

3.5 Write surface

Full gate-stack mechanics (the six-layer funnel, per-feature flags, commissioning holds, the demo re-check) live in `engineering/04 — The Write Path and Safety Model`; this table is the tag-level view of the same, enumerable-write-surface register (`Ring.Tests/WriteSurfaceRegisterTests.cs`) that chapter describes. Every writer file below has its own register row, and the register's "no file leaves the register unnoticed / no new writer goes undiscovered" tests are the thing that keeps this table honest over time — if you add a writer without a register row, the test suite fails, not this document.

Writer (file)	Tag(s) it writes	Gate(s) beyond the shared funnel
`BatchStartTankPlcWriter.cs`	`Tanks[N].Formula_number`, `Request_level`, `Start_Type`, `Splitting_A`, `Splitting_B`	roster-change lock, roster-load-failure lock, `StoragePlcIndex/NoWrite` check
`BatchStartPlcWriteHelper.cs`	the ordered batch-start sequence (`Start_Type` written **last**, deliberately)	shares the word-30 batch-control lock

Writer (file)	Tag(s) it writes	Gate(s) beyond the shared funnel
`UseTankControlPlcWriter.cs`	`Tanks[N].Agitator_mode`, `Agt_on_pre`, `Agt_off_pre`, `Liq_add_{1,2,3}_mode`	`TankInstalledWriteGate.AllowsAdditiveWrite`; demo suppression and endpoint refusal both return `false` here (not the "fake success" idiom some other writers use — §4.1 of the engineering chapter)
`UseTankFormulaPlcWriter.cs`	`Tanks[N].Formula_number` (from the Use Tank screens)	`EnableUseTankFormulaWrite` flag, gated before ReadOnly is even checked
`TankAgitatorPlcWriter.cs`	agitator cycle bits, via `RosterWriteIndex.StorageWriteIndexForLegacyWindow`	`EnableHmiAgitatorCycle` flag (the one `Enable*` flag that ships `true` in the shipped JSON — see the engineering chapter §4.2). **Never writes `Tank_IO[N].O_Agitat` itself** — see the note below.
`TVCControlPlcService.cs`	`Tanks[N].Temp_mode` (per-tank selector, `SetTankTempModeAsync`); `TVC[N].Temp_mode`, `TVC[N].Preset_Temp` (shared unit, `SetMainTVCControlModeAsync`)	`MainTvcWriteAuthorization` (**hard-closed**, always `false`), `TvcCoolingWriteAuthorization`, `TankInstalledWriteGate`
`InventoryAmountPlcWriter.cs`	`Inventory_Amount[0..13]` (REAL array, atomic whole-array write)	`EnableInventoryAmountWrite` flag; even enabled, still needs a healthy changing heartbeat, non-demo session, and full read-back verification before the commissioning gate is considered signed
`BatchFormulaPresetPlcWriter.cs`	the recipe bank tags for a formula number	`EnableFormulaBankWrite`, formula-number range check, payload validation, active-batch interlock
`ShiftControlPlcService.cs`	shift-control tags	`EnableShiftControlPlcWrites`
`ProcessorTimeDatePlcService.cs`	processor time/date sync tags	`EnableProcessorTimeDateSync` (the one flag that defaults **true**)
`AlarmSilencePlcService.cs`	`PC_Write_Integer[30].3` by default (the alarm silence pulse)	`EnableCustomSilenceLatchTag` (`false` by default) gates retargeting; **`B3_bit[15]` is explicitly not an acceptable alternative target** — it is the ladder's own OTL latch, not a legacy PC write, and pointing the silence pulse at it produces a partial, horn-still-ringing silence, not parity (`AlarmSilencePlcService/WriteSurfaceRegisterTests.cs`)
`InventoryEventCaptureService.cs`	inventory-snapshot acknowledge, `PC_Write_Integer[27]` bit 4	`EnableInventorySnapshotPlcAcknowledge`; documented as having **no outer heartbeat gate**

Writer (file)	Tag(s) it writes	Gate(s) beyond the shared funnel
`TankNamePlcWriter.cs`	tank name sync tags	`EnableTankNamePlcNameSync`

What Ring never writes, restated for this chapter's read/write map: `Tank_IO[N].O_Agitat` (the physical agitator output) is ladder-owned; Ring reads it (§3.4) but no writer in the register above, and none in the wider codebase, writes it. `MainTvcWriteAuthorization` and `TankTempModeWriteAuthorization` are the two commissioning holds that ship hard-closed on purpose — see the engineering chapter for why. The two-step "write `TVC_Ctrl[N].enabled` then the mode" that an earlier version of the TVC writer used was deleted on 2026-08-26 after the field capture showed the enabled bit reading 0 on every slot regardless of whether the plant was actively heating or cooling — writing it would have been asserting a bit nothing downstream demonstrably reads.

3.6 UDT notes

UDT	Array	Purpose	Notes
`Tank`	`Tanks[13]` (slots 1–12 used)	Per-tank process state: level, temps, formula number, setpoints	`Status` field has its own enum, unrelated to the C# `TankStatus` enum despite the shared name (`PLC_TAG_REFERENCE.md`)
`Tank_I_O`	`Tank_IO[13]`	Per-tank discrete I/O	Authoritative source for agitator/fill/discharge-valve state; see the bit table in §3.3
`Tank_c`	`Tanks_c[13]`	Per-tank supervisor configuration	Role bits (§3.2), `nominal_Size`, `tank_installed/Liquid_{1,2,3}_installed` commissioning inputs to the write gate
`Pump_LA_I_O`	`Pump_LA_IO[N]`	Liquid-additive pump output	`O_Pump` at bit 1 of the `_0`-suffixed alias — do not confuse with the `Tank_I_O` alias suffixes
`Pump_Tank_I_O`	—	Tank pump UDT, `O_Pump` at bit 1	Referenced in the use-tank open-questions doc; not read by any service found in this pass
`TVC` / `TVC_c` / `TVC_Control` / `TVC_I_O`	`TVC[9]` / `TVC_c[9]` / `TVC_Ctrl[9]` / `TVC_IO[9]`	Water-jacket runtime values, supervisor config, runtime control flags, discrete I/O	1:1 slot mapping with `Tanks[N]`, index 0 reserved (§3.2)

4. Opcode and ingredient map

Every formula step is, at the controller, a single small integer: the value `Batch_Current_Preset_Operation` holds while that step is active. This chapter is the rosetta for that integer — what it meant to the legacy Borland system, what the running L5K's own comments say it means, what Ring shows for it today, and where the number turns into money and inventory counts.

4.1 Where the authority lives

Three generations of documentation exist for these codes, and they do not all agree:

- **The running controller's own L5K export** (`scripts/RS_3000_RUNNING_2026-06-09.L5K`, `Batch_Current_Preset_Operation`'s `COMMENT[]` metadata) is the authority Ring's code was written against for the codes that differ.
- **`legacy-extract/opcode-rosetta.csv`** (and the identical `.json` / `-review.md`) is a read-only extract generated 2026-06-11 that cross-joins the *2024* L5K export's `Formula_Calc` CASE comments, the VersaView `Ingredients.ini`, the MultiPanel Dutch ingredient file, and the 1500-series Dutch string dictionary. It predates and is **not** sourced from the running controller's own program.
- **`Ring/Services/PLC/BatchOperationMapper.cs`** is what actually ships — the `OperationMap` dictionary the app displays from.

Where these three disagree, `BatchOperationMapper.cs` documents the resolution inline, and this chapter follows it. The one place they diverge in fact (not just label wording) is codes 29–31 — see §4.3.

4.2 The opcode table (0–28)

Source: `legacy-extract/opcode-rosetta.csv` cross-checked against `BatchOperationMapper.cs`'s `OperationMap`. **Ring label** is what the app displays by default; if a supervisor has renamed the plant's ingredient vocabulary (Setup → Make Ready Tank names), the ingredient rows (not the action rows) display that configured name instead — see §4.4. **Solids** and **weight-excluded** come from the L5K's `Weight_for_Solids` set and the legacy weight-loop exclusion set respectively, both corroborated in the rosetta review.

Op	Ring label (default)	2024 L5K label	Dutch (MultiPanel)	Contributes to solids?	Excluded from batch weight?
0	No Step or Pass	No step	*(none — see note)*	No	Yes
1	Water (Fast)	Water	Water	No	No
2	Heat (Steam)	Steam	Verwarming	No	Yes
3	Finish Water (Slow)	Finish Water	Eindwater	No	No
4	Domestic Starch	Starch 0 - Domestic Starch	Natief Zetmeel	**Yes**	No

Op	Ring label (default)	2024 L5K label	Dutch (MultiPanel)	Contributes to solids?	Excluded from batch weight?
5	Modified Starch 1	Starch 1 - Modified Starch 1	Gemodificeerd Zetmeel	**Yes**	No
6	Concentrated Caustic	Caustic	NaOH	No	No
7	Primary Borax	Primary Borax	Borax Primair	No	No
8	Waste Water	Wate Water *(sic, L5K typo)*	Gebruikt Water	No	No
9	Secondary Borax	Secondary Borax	Borax Secundair	No	No
10	Liquid Additive 1	Liquid Additive 1	Vloeibare Additief 1	No	No
11	Liquid Additive 2	Liquid Additive 2	Vloeibare Additief 2	No	No
12	Liquid Additive 3	Liquid Additive 3	Vloeibare Additief 3	No	No
13	Biocide (Liq-4)	Liquid Additive 4	Anti Bacterie Middel	No	Yes
14	Pump to Storage	Pump to Storage	Pomp naar Voorraad	No	Yes
15	Caustic Dilution	Dilute Caustic	Verdunde NaOH	No	No
16	Modified Starch 2	Starch 2 - Modified Starch 2	Gemodificeerd Zetmeel 2	**Yes**	No
17	Slurry 1	Slurry Starch 1	Slurry 1	**Yes**	No
18	Slurry 2	Slurry Starch 2	Slurry 2	**Yes**	No
19	Turn On Agitator	Agitator ON	Zet Roerwerk Aan	No	Yes
20	Turn Off Agitator	Agitator OFF	Zet Roerwerk Uit	No	Yes
21	Measure Viscosity	Viscosity	Meet Viscositeit	No	Yes
22	Batch Inspection	Batch Inspection	Batch Inspectie	No	flagged — see below
23	Batch Done	Batch Done	Batch Klaar	No	Yes
24	Ultramixer On	Ultramixer ON	Ultramixer Aan	No	Yes
25	Ultramixer Off	Ultramixer OFF	Ultramixer Uit	No	Yes
26	Manual Addition	Manual Addition	Handtoevoeging	**Yes**	No
27	Liquid Additive 5	Liquid Additive 5 *(legacy VersaView label: "Calciban")*	Vloeibare Additief 5	No	No
28	Liquid Additive 6	Liquid Additive 6	Vloeibare Additief 6	No	No

Op 0 note: the MultiPanel Dutch file is 1-based against these opcodes and has no row for op 0 at all (the rosetta review flags this as `NL_SLOT_BASE`); there is no missing translation to chase here, the source file simply never had one.

Op 22 flag (`WEIGHT_OP22`): the rosetta review notes the legacy C++ weight loop (`DatabaseFormula.cpp`) excludes op 22 (Batch Inspection) from the batch weight total, while the *2024* L5K's `Formula_Calc` CASE comments include it. This is a genuine disagreement between two legacy source artifacts, not resolved by any file read for this chapter — treat any total that hinges on whether op 22 is weighted as **unverified** until someone checks it against the running controller's own math.

English label drift: nineteen of the codes in this table carry a different English wording between the 2024 L5K, the VersaView ingredient list, and Ring's own label (full detail in `legacy-extract/opcode-rosetta-review.md`'s "Flagged rows" section). These are wording differences only — the underlying operation is the same code in every source. The most operationally relevant one: op 8 is "Wate Water" in the L5K itself (a typo baked into the running program), "Processed Water" on VersaView, and "Waste Water" in Ring.

4.3 Codes 29–31: controller service steps, not ingredients

`BatchOperationMapper.cs` documents this divergence directly, and it is worth restating because it is the one place the rosetta extract is **superseded** by better evidence found afterward:

Op	Ring label	Source of truth
29	Rinse Mixer	Running controller's own L5K `COMMENT[29]` (`scripts/RS_3000_RUNNING_2026-06-09.L5K`, ~line 2272)
30	Viscometer Adjust	Running controller's own L5K `COMMENT[30]`
31	Viscometer Water	Running controller's own L5K `COMMENT[31]`

`legacy-extract/opcode-rosetta.csv` — built from the 2024 export, not the running program — lists these differently (29 as "Rinse Tank" / "Liquid Borax", 30/31 as "Prefill Caustic" / "Prefill Dilute Caustic"). The code comment in `BatchOperationMapper.cs` calls this out explicitly: the running controller's own metadata is the authority, and the 2024-export rosetta is stale for this range specifically. Codes 29–31 are also excluded from the `GetSupervisorIngredientName` lookup (§4.4) — they are controller *actions* (mixer rinse, viscometer service steps), not ingredient charges, so a supervisor renaming an ingredient can never affect their label.

Codes 32–33 appear only in the VersaView ingredient list (never in an L5K comment Ring's code was checked against) and are not in Ring's `OperationMap` at all — unassigned as far as this repository's evidence goes.

4.4 Ingredient names are supervisor-configurable

For the ingredient charge codes only (1, 3–13, 15–18, 27, 28 — the plant's actual raw-material list), `BatchOperationMapper.GetOperationDescription` first checks whether a supervisor has applied a make-ready-tank name snapshot (`MakeReadyTankSupervisorNamesState`); if so, the configured name (e.g. a plant-specific name for op 6's caustic charge) wins over the built-in label. This mirrors the legacy RS-360 `OperationDescription` behavior. Action codes (0, 2, 14, 19–26, 29–31 — steps that don't name a material) are never subject to this override and always resolve through the localized action-vocabulary keys (`S.BatchOperation.*`), independent of display language.

`GetOperationCode` (the reverse lookup used when importing formulas/reports) accepts the canonical English label, the localized action string, *or* the currently configured supervisor ingredient name — so an export taken under one plant vocabulary still imports correctly after a rename.

4.5 Ingredient cost — how a code becomes a price

`IngredientCost` (`Ring/Models/IngredientCostRecord.cs`, `Ring/Database/Repositories/IngredientCostRepository.cs`) is keyed directly on the same `OperationCode` integer, deliberately — pricing an ingredient never depends on name-matching a label string. Effective dating is the point of the table: changing a price inserts a **new** row with a later `EffectiveFromUtc` rather than overwriting the old one, so a historical batch always prices at what was true on the day it ran. The price in force for a batch is resolved as the row with the greatest `EffectiveFromUtc` that is `<=` the batch's start time (`IngredientCostRepository.cs`'s `GetEffectivePrices`/`GetPriceAsOf`-style query, lines ~204–213). `UnitCost` and `UnitName` are informational/label fields on the row — the unit is whatever the owner entered it against, most naturally pounds for the solids-scale ingredients.

`IngredientUsage` (`Ring/Models/IngredientUsageRecord.cs`, `IngredientUsageRepository.cs`) is the per-batch consumption row this cost table prices against: `OperationCode` (the join key), a denormalized `OperationName` snapshot captured at write time (informational — a rename afterward does not rewrite history), `ActAmount` (what the controller actually delivered) and `CalcAmount` (the formula's preset/calculated amount). Both the live batch recorder (`BatchLifecycleRecorder.cs`) and the legacy-history backfill path (`BatchHistoryBackfillService.cs`) populate `OperationName` from the same `BatchOperationMapper` resolution described above, so a Usage Report row's label matches what the operator saw on screen for that op code at the time.

4.6 The 14 inventory slots

`InventorySnapshotRecord` (`Ring/Models/InventorySnapshotRecord.cs`) mirrors the controller's `Inventory_Amount[0..13]` array directly as fourteen columns, `Amount0` through `Amount13` — one persisted row per inventory event, alongside `Carryover`, `Counter`, `InitiatorCode` and `FormulaNumber` stamps. `IngredientStockThresholdRecord` (`Ring/Models/IngredientStockThresholdRecord.cs`, `IngredientStockThresholdRepository.cs`) is the owner-configured low-stock table for the same fourteen slots: `SlotIndex` (0–13, primary key) lines up positionally with `Amount{SlotIndex}`, plus `Enabled`, `LowThreshold`, and a **nullable** `CriticalThreshold`. The nullability is deliberate and documented in the model: when an owner has not set a critical level, the low-stock forecaster reports "below low" only and never fabricates a critical band.

This chapter did not find, in any file read for it, a fixed mapping table stating "slot index N is always ingredient op-code M" — the inventory slots and the batch opcodes are two separate zero-based integer spaces that happen to share a similar shape (14 inventory slots; ~28 ingredient op codes), and nothing read here asserts they line up 1:1. Treat any claim of "slot 4 is always Domestic Starch" as **unverified** unless checked directly against the controller's own inventory-slot configuration.

4.7 Units: pounds are native

Every stored ingredient amount — `IngredientUsage.ActAmount/CalcAmount`, formula step presets, `InventorySnapshot.Amount0..13` — is a **pound** value at the controller and in the database. Ring's display-unit toggle (kg, if enabled) is a presentation-layer conversion only; it changes nothing about what is written to the controller or stored in SQLite. See `operator/05 — Formulas and recipes` for the operator-facing statement of the same rule and a worked example.

The `resize_class` column in the opcode rosetta (`scale-int` for most ingredients, `scale-real-caustic-borax` for caustic/borax/borax-dilution codes 6, 7, 9, 15, `verbatim` for action codes) describes how the *legacy* Borland/L5K batch CASE logic scaled the raw controller integer into an engineering-unit amount — it is a fact about the legacy scaling path, not a statement about Ring's own read/write scaling, which is not covered by the files read for this chapter.

4.8 No acceptance bands exist for ingredient amounts

The three formula-guideline PDFs extracted from the legacy plant drop (`legacy-extract/formula-guidelines-extract-notes.md`, covering `APS_Formula Guidelines.pdf`, `APS_FormulaGuidelines.pdf`, and `RS360FormulaGuidelines.pdf`) were read in full for this project. **None of the three states a minimum/maximum acceptable range for any ingredient input amount.** What they do contain:

- A single nominal recipe (the "Corn Starch Sample Formula") given as one fixed weight per ingredient per step, scaled by which Make Ready Tank size is loaded (340 gal or 800 gal) — a point value, not a band.
- Genuine min/max ranges only for **post-batch QC outcomes**, not input amounts: viscosity 35–40 Stein-Hall, temperature 100–105°F, gel point 145–148°F, and a single-point 30% solids target.
- Corrective tuning deltas (e.g. "increase caustic by 0.2 gal / 2.52 lb increments") for adjusting the *next* batch after an out-of-range QC result — step sizes for manual tuning, not a stated tolerance on any one batch.

Do not present any formula amount as having an acceptance band in Ring or in any document derived from this one — the source manuals do not supply that data, and per this repository's evidence-discipline rule, no band should be invented to fill the gap.

5. Legacy RS360 migration

Ring's predecessor at this site is a Borland C++ application referred to throughout this documentation as legacy `~RS360` (the source tree's own top-level directory name). It is a **supervisory PC application** — the direct functional ancestor of Ring, not the controller: batch start, doser tanks, formula edit, inventory edit, names edit, password edit, reports, setup, startup, TVC control and use-tank edit are the same module list on both sides (`RS-APS Computer*` chapter of the plant's own APS-360 manual binder describes this ancestry; see `legacy-extract/pdf-leverage-map-notes.md`). The controller and ladder logic underneath are unaffected by which supervisory application is running — that is what makes a guarded rollback possible at all (see Chapter 7 — Cutover and rollback).

This chapter is written for whoever has to answer "what happens to our old data" at a site still running `~RS360`. It covers three things, in the order a site actually needs them:

1. **What imports automatically** — the in-app importer, field by field.
2. **What was extracted from a legacy tree but does not feed the importer** — batch history, shifts, formula numbers, alarm cross-reference, Dutch dictionaries — described by shape, not reproduced here.
3. **What stays behind** — the legacy application itself, kept installed and working as the rollback path.

Where legacy data lives on a plant PC

On this reference site the legacy tree was copied off the LCP to `C:\PlantDrop\Ringwood\RS-360\` for extraction work (`legacy-extract/pdf-leverage-map-notes.md`, `docs/production-readiness/LEGACY_GOLDMINE_2026-06-11.md`). That is a field-capture convention for this engagement, not a Ring or `~RS360` install-path constant — a different site's copy will land wherever its own IT team puts it. What is structural, and what the in-app importer actually looks for, is the subtree layout: an `Ini\MultiPanel\` directory holding plain slot-numbered text files, and — one level up — an `Ini\Language\` directory that may hold `Name.dec_ini`. `LegacyRs360ImportService.ResolveMultiPanelDirectory` (`Ring/Services/LegacyRs360ImportService.cs:364-385`) accepts either the legacy root or the `MultiPanel` folder itself, and falls back to a recursive search for a directory literally named `MultiPanel` — so the importer does not depend on the `C:\PlantDrop\...` convention.

What the guided import maps, field by field

The production import path is an in-app screen (Setup → **Import Legacy RS-360 Data**, `Ring/Views/Setup/LegacyImportScreen.xaml.cs`), not a script run by an engineer. It is **preview-first**: `LegacyRs360ImportService.BuildPlan` builds a row-by-row plan — legacy value, current Ring value, and a NEW/CHANGED/UNCHANGED/NO MATCH status — before anything is written, and `Commit` only runs the rows the operator approved (`Ring/Services/LegacyRs360ImportService.cs:394-518`). Every write goes

through Ring's own repositories, so open screens refresh live rather than requiring a restart. A PowerShell equivalent (`scripts/Import-LegacyRS360.ps1`) exists for bench/dev use, but the in-app screen is the one a plant runs (class doc comment, `LegacyRs360ImportService.cs:11-25`).

The importer reads four source files, all decoded with an explicit ANSI codepage (default 1252, selectable in the UI) — deliberately not from label text, because the mapping is keyed on the legacy **slot number** convention so it works regardless of what language the legacy install is in:

Source file	What it holds	Lands in
`Ini\MultiPanel\tanks.txt`	Slot-numbered tank/mixer names	`TankAndGroupName` (tank rows) and `MakeReadyTankNames` (mixer row), unless `Name.dec_ini` supersedes it — see below
`Ini\MultiPanel\ingredients.txt`	Slot-numbered ingredient/make-ready names	`MakeReadyTankNames` fields
`Ini\MultiPanel\formulas.txt`	Slot-numbered custom formula names	`FormulaName`, one row per formula number
`Ini\MultiPanel\alarms_1-100.txt`, `alarms_101-200.txt`, `alarms_901-999.txt`	Slot-numbered alarm description text	`AlarmDefinitions.DescriptionTemplate`, when the operator opts in to alarm localization

`tanks.txt`'s tank rows are a **positional guess**: slot 1 is the mixer (high confidence), slots 2–7 map to storage tanks 1–6 (high confidence), but slots 8–13 → tanks 7–12 are marked `REVIEW` in the importer's own mapping table (`LegacyRs360ImportService.TankMap`, `LegacyRs360ImportService.cs:110-125`) because the offset is inferred, not stated. Where a `Name.dec_ini` file is present (`Ini\Language\Name.dec_ini`, resolved relative to the `MultiPanel` directory — `ResolveNameDecIniPath`, `LegacyRs360ImportService.cs:308-336`), its tank rows **replace** the `tanks.txt` guess entirely and are treated as high confidence, because `Name.dec_ini` is what the legacy HMI itself reads and keys tanks 1–12 directly with no offset to infer. On this reference site the two sources actually disagree — `tanks.txt` still carries 2011-era static names while `Name.dec_ini` carries the names in service (code comment names an example: tank 7 = "MF 2", tank 9 = "Double Backer", `LegacyRs360ImportService.cs:199-211, 430-436`). `Name.dec_ini` also carries six `[GROUP 1]` storage-group names that `tanks.txt` has no equivalent for at all — before this importer, Ring had no import path for group names and they stayed on Ring's own defaults.

`Name.dec_ini` values are stored in an HTML-numeric-entity form (`&#77;&#70;` = "MF") that the legacy application itself writes and reads (`Exchange\Main.cpp AnsiToAnsiHTMLDec/AnsiHTMLDecToAnsi`, per the code comment at `LegacyRs360ImportService.cs:213-223`). The importer's decoder (`DecodeAnsiHtmlDec`, `LegacyRs360ImportService.cs:225-246`) differs from legacy's on purpose in one place: legacy assumes every token is well-formed entity syntax and can throw or garble on one that is not; the importer falls back to returning the value trimmed as-is rather than mangling or dropping it, because a hand-edited `Name.dec_ini` is a real possibility in the field and the preview screen is where an operator would notice.

Import behavior worth stating plainly:

- **Blank legacy values are never imported.** A blank name slot in the legacy table does not overwrite a name already set in Ring — clearing a name stays a deliberate Setup action, never a side effect of import (`LegacyRs360ImportService.cs:252-254`).
- **Factory-default formula names are skipped.** A `formulas.txt` slot still reading `Formula N` (the untouched default) is counted but not imported — only formulas the plant actually renamed come across (`IsDefaultFormulaName`, `LegacyRs360ImportService.cs:192-196`, `487-498`).
- **Names are length-capped** to whatever `SetupNameValidator.PlcNameMaxLength` allows, so nothing imports that Ring's own Setup screens would then refuse to save back (`LegacyRs360ImportService.cs:256-257, 298-299`).
- **Alarm import updates description text only**, and only for alarm numbers Ring's own catalog already has a row for; a legacy alarm number with no match in Ring is reported as skipped, not created (`LegacyRs360ImportService.cs:586-593`).

What the import deliberately does not carry

The importer is a **names-and-labels** bridge, not a data migration. It does not move formula ingredient amounts, batch history, shift history, or tank/PLC configuration of any kind — those either stay in the controller (where they always lived) or are a manual, evidence-checked exercise, not an automated one. Nothing below is invented; each was confirmed extractable from a legacy tree during field work, described here by shape only — per the evidence rules in this book, no raw batch data, timestamps or volumes from that extraction are reproduced on this page.

- **Batch history and batch-step detail.** The legacy tree carries a batch-level history table and a much larger step-level detail table (formula step, actual amount, timing) — real extraction artifacts exist at this scale but are not part of the guided import; a site wanting historical batches inside Ring needs a separate, purpose-built migration, not this screen.
- **Shift history.** Legacy shift/production records exist at comparable scale to batch history. Same story: extractable, not imported by this screen.
- **Formula guideline numbers.** The plant's own formula-guideline manuals (three PDF sources, cross-checked against each other — see `legacy-extract/formula-guidelines-extract-notes.md`) describe **one** documented formula chemistry as single point amounts scaled by Make-Ready Tank size (340 gal / 800 gal) — not tolerance bands. The manuals give no acceptance range for an ingredient input amount anywhere; where a number looks like a range it is actually two alternate fixed doses picked by product grade (5-mol vs. 10-mol borax), not a min/max. The only genuine ranges in those manuals are **post-batch QC outcome checks** (viscosity, temperature, gel point, percent solids) with corrective deltas for the *next* batch — not a statement that a batch outside the delta is invalid. None of this is a Ring formula amount "band"; do not treat it as one (see Chapter 4 — Opcode and ingredient map).
- **Alarm cause/remedy text beyond the description template.** A separate extraction cross-referenced three manual sources (the live plant's `RS360WarningAlarms.pdf`, the `APS_WarningAlarms.pdf` variant, and the "Alarms and Warnings" chapter of the RS-3000 all-in-one manual that ships with Ring) by alarm number, and found manual-backed text for 132 of 170 legend alarm numbers (`legacy-`

`extract/alarm-manual-extract-notes.md`). None of the three manuals separates "cause" from "remedy" as distinct fields — most entries give causes only, and the extraction deliberately did not invent remedies the manual does not state. This cross-reference is a documentation source, not something the importer writes into Ring's database.

- **The Dutch-string dictionaries themselves**, covered next.

The Dutch-strings story

The plant runs Ring in English, but the legacy application's own data and internal names are Dutch — the string tables, the alarm decode logic, and some on-disk file content all speak Dutch under the hood (`legacy-extract/nld-strings.csv` is an indexed extraction of that string table; it is a lookup resource, not something reproduced here). Two facts matter to anyone reconciling the two systems' alarm numbering:

- Legacy alarm text is resolved through a **non-linear** lookup, not a fixed offset from the alarm number. `DatabaseString.cpp`'s decode `switch` (referenced at `docs/production-readiness/FIELD_SESSION_RUNBOOK_2026-06-29.md:158` and `docs/production-readiness/LEGACY_GOLDMINE_2026-06-11.md:101`) maps alarm number to string index arbitrarily — for example alarm 3 lands on index 1112, while alarms 11 *and* 12 both land on index 1006. Do not assume `index = 1000 + alarm_number`, or that two different legacy alarm numbers necessarily produce two different legacy strings. `legacy-extract/nld-strings.csv` deliberately preserves the raw string indices rather than pre-resolving them, so anyone reconciling a legacy alarm number against this dictionary has to go through the same switch logic legacy did, not a shortcut.
- Ring's own alarm decode is **PLC-tag-faithful, not legacy-text-faithful.** Where the two disagree on wording for the same underlying condition — a documented example is Ring's SILENCED/RESOLVED banding versus legacy's Active/Ack-Active/Ack-Inactive labels (`docs/production-readiness/PARITY_AND_ONBOARDING_AUDIT_2026-06-18.md:207`) — that is a deliberate design choice in Ring, not a translation bug to fix.

Nothing about the Dutch/English split affects import mechanics: the importer is keyed on legacy slot numbers, never on label text, specifically so it works the same way regardless of which language a given legacy install's own labels are in (class doc comment, `LegacyRs360ImportService.cs:11-25`).

Reconciling batch state across a fallback

If a site falls back to the legacy application mid-shift — planned rollback or an unplanned Ring outage — the two applications are two independent supervisory views over the same controller; the controller keeps running the batch either way, and legacy does **not** read Ring's own database (`CUTOVER_RUNBOOK.md:459`, `459-467`). The step-by-step procedure for running that fallback, verifying legacy is operating, and reconciling afterwards is `CUTOVER_RUNBOOK.md` at the repository root and its rollback-decision matrix; the operator-facing version of "how do I fall back right now" is operator/07 — Troubleshooting §Falling back to the legacy application. This page does not restate either procedure — see Chapter 7 — Cutover and rollback for the summarized version and where the full runbooks live.

What stays behind, and for how long

The legacy application is not removed at cutover. The runbook's own rollback procedure depends on it still being present and runnable on the LCP (`CUTOVER_RUNBOOK.md` steps R2–R4, `445-500`) — a snapshot of the legacy data directory is taken before cutover specifically so legacy can be restored and restarted if a rollback trigger fires. How long a given site keeps legacy installed after a successful, hypercare-cleared cutover is a site and contract decision, not a fact this repository's evidence establishes — this page does not state a retirement date because none is verified here. Chapter 7 — Cutover and rollback carries the staged model (read-only ship → evidence-gated write enable → hypercare) that governs when that decision gets made.

6. Data dictionary

Ring keeps one database. This chapter names every table in it, says what each one is for and what unit its numbers are in, and is honest about the tables that self-trim, the columns that are unrecorded on purpose, and the gap between what an operator can export today and what actually exists.

The storage model

Everything lives in a single SQLite file, `RingwoodDatabase.db`, next to `Ring.exe` (`Data Source=RingwoodDatabase.db;Version=3;` in `Ring/Config/appsettings.json`, resolved against the install directory — `Ring/Database/DatabaseInitializer.cs`). Every connection applies `PRAGMA busy_timeout=15000`, `PRAGMA journal_mode=WAL` and `PRAGMA synchronous=FULL` (`Ring/Database/RingwoodDbAccess.cs`) — `FULL` rather than WAL's default `NORMAL`, deliberately, so a plant power cut cannot lose the last committed writes. `.db-wal` / `.db-shm` sidecar files live beside the main file while the app is running; they are not optional extras, they hold committed data that has not yet been folded into the main file (see Getting data out safely below).

Schema is versioned with `PRAGMA user_version`. `DatabaseInitializer.Initialize()` runs a baseline pass of table-creation steps, then an ordered, savepoint-atomic migration runner — each step's DDL and its version stamp commit inside one `SAVEPOINT`, so a crash mid-migration cannot leave the schema half-upgraded. `CurrentSchemaVersion = 33` (`Ring/Database/DatabaseInitializer.cs:765`, verified in this worktree). `DatabaseBackupService.ExpectedTables` — the list a restore validates a full-version backup against — enumerates **46 tables** (`Ring/Services/DatabaseBackupService.cs:43-109`, counted directly from the array).

Three other places numbers live, deliberately outside the database:

- **`appsettings.json`** (and its `bin\*\Config` copies) — configuration and secrets: connection string, `PlcSettings`, alarm/email routing, SMTP credentials (protected at rest — see Scrubbing an export). Not exported by anything under `Ring/Services/Export`.
- **`%ProgramData%\Ring\backups\`** — the online-backup and pre-migration snapshot destinations `DatabaseHealthService` and `DatabaseBackupService` write to by default (`DatabaseMaintenance.BackupDirectory`, `null` = this default).
- **`%ProgramData%\Ringwood\Ring\ui-audit\ui-audit.jsonl`** — the tamper-evident operator audit journal. It is a hash-chained append-only file, not a table (`Ring/Services/Audit/UiAuditJournal.cs`), and no export routine in `Ring/Services/Export` touches it — see Getting data out safely.

Tables, grouped by what they're for

Grouped as the codebase groups them by comment and by the class of write that owns them, not alphabetically. "Unit" is the unit the raw column holds — storage is always native; see Units, verbatim for the authoritative list. Every table below was read directly from its repository class under `Ring/Database/Repositories/` unless noted.

History — batches, ingredients, alarms, trends

Table	Purpose	Key columns	Notes
`Batch`	Primary batch history	`StorageTankNumber` (roster slot, deliberately not the PLC index), `FormulaNumber/Name`, `RequestLevel` (gallons), `StartUtc/EndUtc`, `Status` (1 Running / 2 Complete / 3 Cancelled / 4 Interrupted / 5 Aborted — `Ring/Models/BatchRecord.cs`), `ActualVolume` (gallons, **NULL on every row before migration v26** — see honesty notes), `PlantBatchNumber/ControllerOutcome` (v28, sourced from the controller's own batch stamps)	The join key back to the controller, the PanelView and legacy Paradox history is `PlantBatchNumber`
`IngredientUsage`	Per-ingredient amounts used in a batch	`BatchId`, `OperationCode` (PLC op code — see chapter 4), `ActAmount/CalcAmount` (pounds), `RecordedUtc`	Mirrors the legacy `USE` table's `RAWPLCCALL/ACTAMOUNT`
`BatchStepEvent`	Per-step timeline within a batch	`BatchId`, `StepNumber`, `OperationCode/Name`, `EnterUtc/ExitUtc`, `DurationSeconds`	
`InventorySnapshots`	Point-in-time ingredient inventory readings	`CapturedAtUtc`, `Carryover`, `Counter` (PLC INT, wraps at 32767), `Amount0`..`Amount13` (14 slots, pounds)	Dedup key is `(Carryover, Counter, PlcEventLocalText)`, not `(Carryover, Counter)` alone — a v7 migration fixed a dedup rule that silently dropped snapshots taken right after the counter wrapped

Table	Purpose	Key columns	Notes
`AlarmRecords`	Legacy-shaped alarm history	`Almid`, `Almnumber`, split `Almdate/Almtime` + `Ackdate/Acktime` text columns	Mirrors the RS-360 Paradox layout column-for-column
`AlarmLifecycle`	Modern alarm history — the MTTR/downtime source	`SlotIndex`, `Almnumber`, `ActiveAtUtc`, `AckAtUtc`, `ResolvedAtUtc`, `LastUpdatedAtUtc`	**Self-trims** — see retention
`PlcAlarmEvents`	Raw alarm transition log	`SlotIndex`, `PreviousValue/NewValue`, `BaseAlarmNumber`, `EventKind`	**Self-trims**
`ViscometerReading`	Recorded viscosity readings	`FormulaNumber`, `RecordedAtUtc`, `RawValue/CalibratedValue`, `StorageTankNumber`, `TemperatureF` (Fahrenheit)	See operator 03a's viscometer section for what these readings do and do not tell you
`ProcessHistory`	Trend/historian samples	`TankNumber`, `CapturedUtc`, `CurrentTempF` (Fahrenheit), `LevelGal` (gallons), `CurrentStep`, `FormulaNumber`	**Self-trims** — 30-day rolling window, see retention
`DowntimeEvent`	Downtime/OEE log	`StartUtc`, `EndUtc`, `ReasonCode`, `OperatorName`	
`ReceivingEvent`	Bulk-material receiving log	`Supplier`, `BolNumber`, `ExpectedLbs/BlownInLbs/VarianceLbs` (every weight column names its own unit)	
`ShiftStartEvent`	Shift start/end log	`ShiftNumber`, `StartUtc/EndUtc`, `OperatorName`	
`ShiftHandoffNote`	Free-text shift handoff log	`ShiftNumber`, `Notes`	
`ShiftProductionLog`	Per-shift production figures	`ShiftDateLocal` (local date, not UTC), `SquareFeet`	Feeds dry-lbs-per-square-foot math
`DataEntryEntries`	Free-form operator data-entry log	`ScreenNumber`, `Field1`..`Field11` (TEXT)	Untyped and unitless by design — meaning comes only from `DataEntryFieldLabels`; export the two together or the numbers are meaningless
`FormulaHistory`	Versioned recipe change history	`FormulaNumber`, `Version`, `AppliedAtUtc`, `StepsJson` (whole step-set snapshot)	

Table	Purpose	Key columns	Notes
`EmailLog`	Sent-email audit	`SentUtc`, `Category`, `Recipients` (PII), `Success`, `Status` (`Pending`/`Sent`/`Failed`, migration v33)	`Status` lets a durable send be recovered as `Failed` if Ring crashed mid-send
`MaintenanceCompletion`	Maintenance sign-off ledger	`TaskId`, `Operator`, `Outcome`, `PerformedUtc`	
`ScaleCalibrationLog`	Scale calibration history	`ScaleId`, `AsFoundZero`/`Span`, `AsLeftZero`/`Span`, `Passed`, `Unit` (nullable, migration v29)	See honesty notes — rows before v29 have `Unit` NULL and genuinely do not record whether a value was pounds or kilograms
`InventoryAdjustments`	Audit trail of operator inventory edits	`MaterialIndex`, `PreviousAmount`/`NewAmount` (pounds), `ChangedBy`, `CommandId` (v27)	Written **before** the PLC write it accompanies
`InventoryAdjustmentCommands`	Durable write-intent journal for inventory edits	`CommandId` (PK), `IntentTimeUtc`, `Outcome` (written only after PLC readback)	The intent row commits before the PLC call, so a crash between intent and outcome is visible rather than silent
`PlcCommunicationLog`	PLC comms audit history (migration v32)	`UtcTimestamp`, `EventType`, `Endpoint` (PLC IP address), `Detail`	Distinct from the per-day CSV files `Ring/Services/PLC/PlcCommunicationLogService.cs` writes to disk; both retention-prune independently (see retention)

Configuration — recipes, catalogs, naming, routing

Table	Purpose	Key columns	Notes
`FormulaSteps`	Recipe steps	`(FormulaNumber, Step)` PK, `Operation` (name, not a PLC code), `PresetAmount` (pounds), `PresetTime` (**seconds**)	Seeded verbatim from legacy pound values (`Ring/Database/FormulaStepSeed.cs`)
`FormulaName`	Custom display names per formula number	`FormulaNumber` PK, `Name`	
`AlarmDefinitions`	Alarm catalog	`AlarmNumber` PK, `DescriptionTemplate`, `Severity` (`A`/`W`)	Seeded only when empty; templates an operator has edited survive a re-seed
`AlarmPlaybook`	First-response guidance per alarm	`AlarmNumber` PK, `Meaning`, `FirstCheck`, `WhoToCall`	Operator-edited

Table	Purpose	Key columns	Notes
`AlarmShelf`	Alarm shelving (ISA-18.2) audit	`AlarmNumber`, `ShelvedUntilUtc`, `ByOperator`, `Reason`	Separate from acknowledgement
`AlarmEmailRule`	Alarm email routing rules	`EventKinds`, `MinSeverity`, `Recipients` (PII)	
`EmailRecipient`	Email address directory	`Name`, `Address` (PII)	
`ReportSubscription`	Scheduled report subscriptions	`ReportType`, `Cadence`, `Recipients` (PII)	
`ViscometerSettings`	Per-formula viscosity adjustment settings	`FormulaNumber` PK, adjustment factors/percentages, alarm high/low	See operator 03a
`TrendingPreset`	Saved trend-chart configurations	`Name` (unique), `TagsCsv`, `WindowMinutes`	UI preference
`ReportFilterPreset`	Saved report filter sets	`(ReportName, PresetName)` unique	UI preference
`MaintenanceTask`	Maintenance task definitions	`AssetKey`, `IntervalKind` (`days`/`hours`/`cycles`), `BaselineRuntimeHours`/`BaselineCycleCount` (v25)	Without the v25 baseline columns, an hours/cycles task measures the asset's lifetime counter and reads permanently overdue
`EquipmentRuntime`	Lifetime runtime counters	`AssetKey` PK, `RuntimeHours`, `CycleCount`	Monotonic accumulators
`TankCipState`	Tank cleaning state	`TankNumber` PK, `State`, `LastCleanedUtc`, `CipIntervalHours`	
`DataEntryFieldLabels`	Names the generic data-entry columns	`(ScreenNumber, FieldNumber)` PK, `Label`	Pairs with `DataEntryEntries`
`BatchQueue`	Supervisor planning board	`FormulaNumber`, `TargetSizeGal`, `Status`	Ephemeral, DB-only; never written to the controller
`IngredientCost`	Effective-dated per-ingredient cost	`OperationCode`, `UnitCost`, `UnitName` (label only — `'lb'`/`'gal'`, no conversion applied), `EffectiveFromUtc`	Old rows are kept so a past batch keeps the price that was in force on its run date
`IngredientStockThreshold`	Low/critical inventory thresholds	`SlotIndex` PK (0-13, matches `InventorySnapshots.Amount0..13`), `LowThreshold`/`CriticalThreshold` (pounds)	

Table	Purpose	Key columns	Notes
`Operators`	Operator name directory	`DisplayName` (unique, case-insensitive)	Names only — no credentials; authentication was deprioritized for this release
`TankAndGroupName`	Tank/group display names	`(SlotKind, SlotNumber)` PK, `Name`	
`MakeReadyTankNames`	Make-ready area naming, single row	29 named columns (ingredient/mixer display names)	
`GroupHardwareSetup`	Plant hardware configuration per tank group	`GroupKind` PK, `InstalledTankCount`, `CoolingExchangerInstalled`	
`ShiftDefinition`	Shift schedule definitions	`ShiftNumber` PK, `StartTime` (local clock time), `DurationMinutes`	
`AppState`	Key-value application state	`Key` PK — `wizard.completed.utc`, `ui.language`, `ui.units.volume/weight/temperature`	Lives in the database specifically so a rebuild, which wipes `appsettings.local.json`, never resets it

What Ring never records with a unit label

Amounts are bare `REAL` columns — nothing in the schema stores a unit alongside a number. The unit is a property of the column, fixed by the code that writes it, and is the same regardless of what the operator's display toggle shows on screen (`Ring/Services/PlantNativeUnits.cs`, `Ring/Services/Display/DisplayUnitService.cs`). This table is the authoritative list — verbatim from the columns above, cross-checked against the writing code:

Column(s)	Unit	Source
`FormulaSteps.PresetAmount`	Pounds	`PlantNativeUnits.RecipeWeightUnit` — see the pounds-native rule
`IngredientUsage.ActAmount` / `CalcAmount`	Pounds	Same rule
`InventorySnapshots.Amount0`..`Amount13`	Pounds	Same rule
`IngredientStockThreshold.LowThreshold` / `CriticalThreshold`	Pounds	Same rule
`InventoryAdjustments.PreviousAmount` / `NewAmount`	Pounds	Same rule

Column(s)	Unit	Source
`ReceivingEvent.ExpectedLbs` / `BlownInLbs` / `VarianceLbs` / `ToleranceLbs`	Pounds	Named in the column itself
`Batch.RequestLevel` / `ActualVolume`	US gallons	`Views/BatchStart/StorageTank1.xaml.cs:1252` passes a gallons value in; `ActualVolume` is NULL on all pre-v26 rows — see honesty notes
`ProcessHistory.LevelGal`	US gallons	Column name
`BatchQueue.TargetSizeGal`	US gallons	Column name
`ProcessHistory.CurrentTempF`	Fahrenheit	Column name
`ViscometerReading.TemperatureF`	Fahrenheit	Column name
`FormulaSteps.PresetTime`	Seconds	`Ring/Database/FormulaStepSeed.cs:122`; the formula report prints `"{PresetTime} Seconds"`
`EquipmentRuntime.RuntimeHours` / `TankCipState.CipIntervalHours`	Hours	Column name
`ShiftProductionLog.SquareFeet`	Square feet	Column name
`ScaleCalibrationLog.AsFoundZero/Span`, `AsLeftZero/Span`	Whatever `Unit` says on that row — **NULL before migration v29**	See honesty notes
`IngredientCost.UnitCost`	Currency per `UnitName`	`UnitName` is a display label only; no conversion is applied to `UnitCost` based on it

`AppState.ui.units.volume/weight/temperature` records the operator's chosen **display** unit. Changing it changes what is painted on screen, and nothing else — see the pounds-native rule and `PlantNativeUnits.cs`'s own safety-rule comment: storage, PLC I/O, clamps and cost math stay raw native values; conversion happens only at paint time.

Time zone conventions

Most timestamp columns are ISO-8601 `"o"`-format UTC text (`StartUtc`, `RecordedUtc`, `CapturedAtUtc`, and so on) — a fixed-width format chosen deliberately so a plain lexicographic text comparison is also a chronological one. A few tables are local-time by design and will not compare correctly against a UTC column:

- `ShiftDefinition.StartTime` and `ShiftProductionLog.ShiftDateLocal` (local clock time / local date — a shift is defined by wall-clock time, not UTC).
- `InventoryAdjustments` carries both `EventTimeUtc` and `EventTimeLocal`.
- `InventorySnapshots.PlcEventLocalText` is the controller's own local-time text, not converted.

- `AlarmRecords` splits `Almdate`/`Almtime` and `Ackdate`/`Acktime` as separate text columns, mirroring the legacy Paradox layout rather than a single UTC instant.

`Ring/Services/Export/DatabaseExportService.cs`'s date-range filter is a `@start`/`@end` text comparison against whichever column a table names as its `DateColumnName` — meaningful on the UTC columns, meaningless if pointed at a local-time one.

Retention — what self-trims

Most tables are permanent history: nothing in the codebase deletes rows from `Batch`, `IngredientUsage`, `AlarmRecords`, `FormulaHistory` and the rest short of an operator-driven database action. Four things prune themselves on a timer, so "everything since install" is not true for them:

What	Window	Mechanism
`ProcessHistory`	30 days (`ProcessHistorianService.DefaultRetentionDays`)	Pruned roughly hourly (every 60 ticks of the 60-second sample)
`AlarmLifecycle`, `PlcAlarmEvents`	`AlarmSettings.AlarmHistoryDays` — **365 days shipped** (`Ring/Infrastructure/Configuration/AppSettings.cs:663`; this worktree's `Ring/Config/appsettings.json` also has `365`)	`ApplyRetentionDays()` on each repository
`PlcCommunicationLog` (table)	Retention-pruned by `UtcTimestamp`; window follows the same `AlarmHistoryDays`-derived configuration	`Ring/Database/Repositories/PlcCommunicationLogRepository.cs`
Per-day PLC comm-log CSVs (disk, not this table)	30 days (`PlcCommunicationLogService.RetentionDays`)	Deleted at most once per local day

> **A contradiction worth flagging.** One fallback read site (`AlarmAlarmNumberPlcService.cs:914`) still falls back to `?? 30` if the setting is unset, and the production-template appsettings ships `30` (`Ring/appsettings.production.template.json:80`) — but the `AppSettings.AlarmHistoryDays` C# property default and this worktree's live `Ring/Config/appsettings.json` are both `365`. A code comment near the fallback records the change explicitly ("this used to read `AlarmSettings.AlarmHistoryDays`... shipped default 30 -> 365"). Treat **365 days** as the current shipped default for a station running today's code, and confirm the actual value on a given station from its own `Config/appsettings.json` before relying on it — this is exactly the kind of number that drifts between a template, a fallback and a running station.

Honesty notes — gaps that are not bugs

- **`Batch.ActualVolume` is NULL on every row before migration v26.** The pre-v26 code summed native-pound ingredient actuals and stored that dimensionally-invalid number in a column every consumer treated as US gallons. v26 NULLed the column outright rather than leave a wrong number in place; v28 added a controller-proven volume signal (`Batch_*_Stamps[25]`) so volumes recorded from that point forward are the plant's own reading, not a Ring computation. A NULL here means "no controller-proven volume was ever obtained for this batch," not zero.
- **`ScaleCalibrationLog.Unit` is NULL on every row before migration v29.** `AsFoundZero`/`Span` and `AsLeftZero`/`Span` are metrology reference values stored without conversion; a pre-v29 row genuinely does not record whether its number meant pounds or kilograms, and this was a deliberate choice not to back-fill a guess. Do not assume a unit for a NULL row.
- **Three tables are rolling windows, not full history** — see Retention. A "complete" export that includes `ProcessHistory` or `AlarmLifecycle` is complete only back to their retention horizon, not back to install.
- **Display units never change what's stored.** Every value in the tables above is native regardless of what `ui.units.*` says the operator sees — see Units, verbatim.

Getting data out safely

The front-door answer is Reports → Export Center. It is the operator-facing successor to the 7-table CSV screen described below: `DatabaseExportSchema.Tables` now enumerates **all 46 tables** `DatabaseInitializer` creates, grouped as History / Configuration / Diagnostics, each with a `Purpose` sentence, `IsSensitive` flag and per-column unit/timezone notes (`Ring/Services/Export/DatabaseExportSchema.cs`). "Export Everything" writes a dated folder (`Ring-Export-yyyyMMdd-HHmm`, local time) containing every dataset in CSV, JSON and/or SQLite, plus a **complete database snapshot** (the same online-backup path as above), a **SQL dump**, the **operator audit journal** with its chain-verification result, this station's **settings with secrets redacted**, and the application/communication logs — everything this section otherwise tells a reader to gather by hand, in one place (`Ring/Services/Export/ExportJobService.cs`).

Two files make the bundle self-describing without Ring installed:

- **`manifest.json`** — machine-readable provenance: app version, the database schema version (from `DatabaseInitializer.CurrentSchemaVersion`), the date range applied, every dataset's row count, and `skipped[]` with a reason for anything the run could not produce (`Ring/Services/Export/ExportManifest.cs`).
- **`README.txt`** — the data dictionary above, generated from the same catalog so it cannot drift from what was actually exported: provenance, format notes, the units table, the local-time-column list, the four rolling-window datasets and their honesty notes, and a per-dataset section (`Ring/Services/Export/ExportDataDictionaryBuilder.cs`).

The screen remembers the destination folder (seeded on every Save dialog across the app, not only here — `Ring/Services/Export/ExportDestinationService.cs`) and offers an **automatic nightly export**: off by default, a configurable local time-of-day and how many dated exports to keep before the oldest are pruned (`AppState` keys `export.schedule.*`, `Ring/Services/Export/ExportScheduleSettings.cs`, `Ring/Services/Export/ExportScheduleService.cs`). The last-run timestamp (`export.schedule.lastRunUtc`) is written **only after the job reports success** — a night the folder was unreachable is a skip, not a stamp, so it is never mistaken for a completed export. Because four tables in this chapter self-trim (see Retention), the nightly export is the practical way to keep a full history of them past their rolling window.

When a selected dataset carries email addresses (`EmailRecipient`, `EmailLog`, `AlarmEmailRule`, `ReportSubscription`) the screen shows a privacy notice before export — the same PII this chapter's scrubbing section lists is called out at the point someone is about to hand the folder to someone else.

The Export Center is additive: it does not replace the two lower-level paths below, and `SqlDumpService`'s whole-database dump remains the only path that is not filtered through a fixed table catalog at all (relevant if a 47th table is ever added before the catalog is updated to match).

Never file-copy the live `.db` file. With `journal_mode=WAL`, recent commits can sit in the `.db-wal` sidecar rather than the main file; a plain copy of `RingwoodDatabase.db` alone can be missing data the app already considers committed, and `DatabaseBackupService.RestoreFromFile` deletes stray `.db-wal`/`.db-shm` files it finds beside a restored copy for exactly this reason. Two paths are actually safe:

1. **`DatabaseBackupService.BackupToFile`** — opens the source read-only and calls SQLite's own online backup API (`BackupDatabase`), which is transactionally consistent even while the live app holds open connections and folds WAL-resident pages into the copy. This is what the startup auto-backup and the Setup → Database Export screen's backup path both use.
2. **`SqlDumpService.DumpDatabase`** — a whole-database `.sql` dump, reading every user table inside one read transaction for a single consistent snapshot, streamed to a `.partial` file and atomically renamed on completion so a power cut never leaves a plausible-looking partial file behind. This is the only export path that reaches every table in the schema, not just the operator-facing export catalog below.

`DatabaseExportSchema.Tables` now whitelists all 46 tables, not a subset. It used to enumerate exactly 7 (`Batch`, `IngredientUsage`, `AlarmLifecycle`, `ShiftDefinition`, `InventorySnapshots`, `ViscometerSettings`, `PlcCommunicationLog`) — an older claim in this chapter's history that this worktree's `Ring/Services/Export/DatabaseExportSchema.cs` no longer bears out. The catalog was widened to cover every table `DatabaseInitializer` creates, each entry now also carrying a `Category` (History/Configuration/Diagnostics), a `Purpose` sentence and per-column unit/timezone notes; every table name and column is still checked against this list before it reaches a `SELECT`, so neither screen that reads from it can name an arbitrary table. `DatabaseExportSchema.ExcludedTables` exists and is deliberately empty — a future entry there is a decision to withhold data and must carry a reason. Two screens read the same catalog: the original **Setup → Database Export** screen (`Ring/Views/Setup/DatabaseExportForm.xaml.cs`) still exports one table at a time from a combo box, while **Reports → Export Center** (see Getting data out safely below) exposes the same 46 tables as a

checklist with formats, date filtering and a scheduled run. Reach for `SqlDumpService`'s whole-database dump, still the only path outside this fixed catalog entirely, when the request is "everything, including a table this catalog hasn't caught up to yet."

The audit journal is a file the exports above do not touch. `%ProgramData%\Ringwood\Ring\ui-audit\ui-audit.jsonl` (plus its rotated archives, retained 30 deep by default) is the hash-chained, tamper-evident record of operator UI actions. Neither `SqlDumpService` nor the CSV export screen reads it — it must be gathered separately, and `UiAuditJournal.VerifyChain` can confirm the chain was intact at the moment of export.

Station backup covers what the database backup does not. Setup → Plant Profile → Station backup writes a `*.ringstation.zip` (`Ring/Services/Setup/StationBackupService.cs`) that bundles the three `%LocalAppData%\Ring\*.json` files — `tank_roster.json` (which PLC array index each storage tank write targets), `operator-session.json`, and `group_hardware_setup.json` — plus the `AppState` key/value rows (chosen language, per-quantity display units, the wizard completion marker). None of this travels with the plain database backup above or with the plant-profile export it extends. It deliberately excludes two things: secrets (the Supervisor/Admin/Demo passwords and the SMTP password never appear in the zip — restoring one never sets a password on the target machine, per `StationBackupService.SecretsExcluded`) and the history database itself, which stays the job of `DatabaseBackupService`/the Export Center above, not of a station backup.

Scrubbing an export

Nothing under `Ring/Services/Export` redacts anything — both export paths stream columns verbatim. Before any export or settings dump leaves the plant, strip:

- From `appsettings*.json`: `AlarmEscalation.Smtp.Password` (the stored value is a protected blob via `Ring/Infrastructure/Configuration/SmtpPasswordProtector.cs`, but a protected secret is still a secret) and `.Username`, `AlarmEscalation.Webhook.Url` (may carry a token), `Authentication.DemoPassword`, `PlcSettings.DefaultIpAddress`, and `Database.ConnectionString` (a filesystem-path disclosure).
- From the database: every `Recipients/Address` column (`EmailRecipient`, `EmailLog`, `AlarmEmailRule`, `ReportSubscription` — all email PII), every operator-name column (`Batch.Operator`, `InventoryAdjustments.ChangedBy`, `DowntimeEvent/ShiftStartEvent/ViscometerReading.OperatorName`, `AlarmShelf.ByOperator/UnshelvedByOperator`, `MaintenanceCompletion.Operator`, `ScaleCalibrationLog.Operator`, `ReceivingEvent.EnteredByUser`, `ShiftHandoffNote.CreatedByUser`, `DataEntryEntries.OperatorName`), `ReceivingEvent.Supplier/BolNumber` (commercial), `IngredientCost.UnitCost/Batch.ComputedCost` (commercially sensitive), and `PlcCommunicationLog.Endpoint` (a plant network address — network topology is otherwise approved for publication per this book's ground rules, but a specific customer's export is a different question from documenting the reference site).

Related reading

- `engineering/05 — Data and Persistence` covers the migration architecture and backup/restore mechanics in more depth than a lookup chapter needs.
- Chapter 4 — Opcode and ingredient map for the pounds-native rule and how `OperationCode` maps to an ingredient.
- `operator/03a — Main screen reference` for what the viscometer readings mean to an operator on shift.

7. Cutover and rollback

This chapter is the public summary of how Ring moves from installed-and-reading to writing to a live controller, and how a site gets back to the legacy `~RS360` system if that goes wrong. It distills the internal runbooks — it does not replace them. The runbooks themselves ship inside the release package (see below) and are the only documents that should be followed step-by-step during an actual cutover or rollback window.

The staged model

Ring reaches a plant in three stages, never skipping one:

1. **Read-only first.** Ring ships with `PlcSettings.ReadOnlyMode = true`. Every controller write is suppressed and logged at the point it would have happened; every read is unaffected. A site can install Ring, connect it to a live, producing controller, and watch it track the real process for as long as it takes to trust the reads — without any possibility of it commanding anything.
2. **Evidence-gated write enable.** Flipping to writes is a deliberate, supervised step, gated on a written readiness record (per-item bench evidence for every PLC-touching feature, controller identity confirmed, the legacy application proven off the wire, no simulator on the path), not on a calendar date. The flip itself is one configuration change plus a restart — see below — but the gate in front of it is the larger part of the work.
3. **Hypercare.** A fixed, high-touch support window that starts the moment cutover sign-off completes and runs until the plant has demonstrated stable production operation. See severity tiers below.

Flipping the global gate does not, by itself, arm any single feature. Each write-capable feature (batch start, hold/resume/reset, agitator, formula bank, inventory adjustments, and the rest) carries its own `Enable*` flag, its own commissioning holds, and the live heartbeat gate underneath all of it — the same gate stack described in engineering/04 — Write path and safety. `ReadOnlyMode = false` arms the outermost layer only; the site chooses which individual features to turn on, and when, on their own evidence.

The flip

The one configuration change that moves a station from read-only to write-enabled: set `"PlcSettings": { "ReadOnlyMode": false }` in the deployed `Config\appsettings.local.json` — the copy sitting next to the running `Ring.exe`, not the source checkout — and restart Ring. The startup log stops emitting `[ReadOnly] … SUPPRESSED` lines once the gate is off; that absence, not a UI banner alone, is the thing to confirm.

Two things worth being explicit about because they are easy to get backwards:

- The flip is a **plant-wide safety default**, not a per-feature switch. Ring ships read-only everywhere and every controller write anywhere in the app is suppressed until this key is off — there is no per-screen or per-tank read-only toggle underneath it.

- **At most one write-enabled Ring station may exist on the plant network at a time.** Ring's single-instance guard is a named mutex scoped to a single Windows logon session, not to the machine — a second interactive session on the same PC (a different login, a remote-seat viewer session) gets its own mutex namespace and would not be stopped by the guard, and there is no cross-machine arbitration at all. Confirming "only one Ring is running" means asking everyone who could have an install, not just checking one session on one machine.

The first-start wizard's read-only page — the green/red banner an installer sees at commissioning — covers this same setting from the installer's side; see installer/03 — First start, page 5.

What changes on the first write-enabled shift

Nothing about the screens, the navigation, or what an operator can see changes at the moment of the flip. What changes is that buttons which previously logged a suppressed write now actually reach the controller — batch start, hold/resume/reset, formula-bank edits, inventory adjustments, and whichever other feature families the site has separately enabled. What does **not** change, ever, regardless of `ReadOnlyMode`: Ring never drives the agitator output — that stays ladder-owned — and the two deliberately fail-closed write authorizations are never populated by a configuration flip. See engineering/04 for the full "what Ring never writes" list.

Verification after the flip

The internal cutover runbook runs a fixed sequence after the flip, checked in ascending order of consequence: the lowest-risk write first (processor time/date, if that feature is separately unlocked), then alarm silence, then hold/resume/reset, then a batch member write, then the agitator decision, and only last — with the equipment isolated — the split-batch fill valve, the single highest-consequence write in the system. Each step has an explicit pass/fail check against a specific tag, not a general "it looked right." That full step-by-step sequence, its preconditions, and the tag-level detail behind each check live in the internal runbooks (see "Where the full runbooks live" below) — this chapter names the shape of it, not the procedure itself, because a write-path verification step is a safety procedure and belongs in exactly one maintained document.

The rollback net

Rollback exists because the legacy `~RS360` system is never uninstalled during cutover — only stopped. The cutover sequence closes the legacy application and snapshots its data directory to a removable backup drive before Ring is ever launched; the Borland binaries and shortcut stay on the machine. Rolling back is, in shape: stop `Ring.exe`, restore the legacy data-directory snapshot (or a full disk image, if that was the backup form used), restart `~RS360`, and confirm it is talking to the PLC again. `RingwoodDatabase.db` is left in place for forensics — the legacy application never reads it, so nothing is lost by leaving it there.

Two versions of that sequence exist, for two different situations:

- A **ten-minute operator-facing checklist**, designed so an on-shift operator plus one engineer can drive an emergency fallback without waiting for anyone else, for the "something is visibly broken right now" case (Ring won't start, the PLC heartbeat is red, a mandatory screen throws an exception, an unsafe state is declared). It assumes the snapshot and backup drive from cutover are already in place; if either precondition is missing, it explicitly degrades into "follow the longer engineer runbook instead."
- A **longer engineer-side procedure** covering the same restore in more detail, plus a decision matrix for judging whether a given failure calls for rollback at all or is better handled as a hotfix-in-place (a single broken screen with everything else healthy, for example, is a hotfix; a batch that fails to record its own completion is a rollback).

Both explicitly reserve the harder cases — a full disk-image restore, or a PLC-side controller restore from a pre-cutover ACD backup — as slower paths outside the ten-minute budget, run by an engineer, not the operator checklist.

Severity tiers, from hypercare

Once cutover sign-off completes, incidents are triaged into four tiers with response and fix-or-rollback targets, not left to individual judgement each time:

Tier	Meaning	Response SLA	Fix-or-rollback SLA
P0 — Plant down	Production is fully blocked: PLC heartbeat red more than 10 minutes, Ring won't launch, the database is corrupt or unreadable, every operator is blocked	< 15 min	< 2 hr (hotfix or invoke rollback)
P1 — Degraded production	Production continues but a real capability is broken or unreliable: one screen won't render, a report is missing or wrong data, intermittent PLC read errors	< 1 hr	< 8 hr
P2 — Annoyance	Cosmetic or low-impact, with a workable manual workaround	< 4 hr	Batched into the next routine release
P3 — Backlog	Feature requests and polish	Acknowledged at daily standup	Deferred to a future release

These are the SLAs a deployment's hypercare plan defines by default; the on-call rotation, the daily monitoring checklist, and the exit criteria (a run of consecutive incident-free days, by default 14) are set per deployment at scheduling time and live in the internal hypercare plan.

Where the full runbooks live

The runbooks this chapter summarizes **ship inside every release package**, next to the application itself, so the machine that has the software also has the procedure for installing it, cutting it over, and rolling it back:

Document	Covers
`CUTOVER_RUNBOOK.md`	The per-LCP cutover sequence for this plant: prerequisites, numbered steps with verification, the rollback mechanism, and the decision matrix for hotfix-vs-rollback.
`STARCH_SYSTEM_CUTOVER_RUNBOOK.md`	The reusable read-only-first bring-up pattern this cutover model is built from — written to be repeatable for any new starch system, not specific to one plant.
`13_RS3000_ROLLBACK.md`	The ten-minute operator-facing rollback checklist described above.
`12_INSTALL_RUNBOOK.md`	Installation, including the per-LCP configuration overlay the cutover depends on.

A site running Ring finds these files at the root of the extracted release package. They are also indexed, alongside a written write-enable readiness gate and the hypercare on-call plan, from engineering/09 — Build, deploy, release, which is the authoritative index for anyone who needs to trace a claim in this chapter back to its source document.

8. Compatibility matrix

This chapter separates what Ring has actually been proven against, at the one site it currently runs, from what it should work with on paper because of the library it's built on. Where the line falls is stated plainly in every row — this project's history has enough phantom claims already; a compatibility page is exactly the wrong place to add one.

The controller

Item	Status
CompactLogix 1769-L36ERM, firmware v20.12	**Production-proven.** This is the controller at the reference site (`172.22.103.10`) and the family of its bench twin (`192.168.202.15`). Every read and write path Ring ships has been exercised against this exact controller family and firmware line. See reference/01 — PLC and controller facts.
Other ControlLogix / CompactLogix firmware revisions	**Expected to work, not site-proven.** Ring's PLC layer talks EtherNet/IP through `libplctag`'s `ab-eip` protocol driver, the same driver path used for the v20.12 controller above; nothing in Ring's code is version-pinned to v20.12 specifically. No other firmware revision has been connected to a live Ring install.
Other Rockwell controller families reachable over `ab-eip` (e.g. Micro800, PLC-5, SLC-500 via a gateway)	**Untested.** `libplctag` supports protocol variants for these families, but Ring's configuration (`PlcSettings.Protocol`, hard-set to `"ab-eip"` — `docs/reference/plc/APPSETTINGS_REFERENCE.md`) and its tag/UDT assumptions were built against a ControlLogix-family memory model. Connecting Ring to a different family would need its own verification pass; treat it as a new integration, not a supported option.
Non-Rockwell controllers	**Not supported.** Ring has no driver path to anything outside the Allen-Bradley EtherNet/IP world `libplctag` covers.

Controller-side programming tool

The reference site's controller-side project is a **v20-era Studio 5000** project. Restoring or modifying the controller-side starch-batch logic (a separate, scoped controls-engineer task from anything Ring does) needs a Studio 5000 version that can open that project — practically, the v20-24 range; a newer major version may prompt an upgrade that is not reversible. (`docs/Ring_RS3000_Sales_Positioning.md`; `docs/reference/plc/PLC_BENCH_FINDINGS.md` records the bench work that was completed *without* Studio 5000, and the short list — mainly `MainTask` un-inhibit — that still needs it.) This is a controls-engineering detail, not a Ring requirement: Ring itself never opens or needs Studio 5000 at runtime.

The comms library

Ring's controller `.csproj` pins two NuGet packages (`Ring/packages.config`):

Package	Version
`libplctag`	1.5.2
`libplctag.NativeImport`	1.0.41

`libplctag` is the open-source EtherNet/IP client Ring's `Ring/Services/PLC/` layer builds on; Ring does not implement CIP framing itself. Upgrading either package is a real change to the comms stack and belongs in engineering/03 — PLC communications's scope, not a routine bump.

L5K / ACD export vintages the tooling reads

Ring's own tag-audit and formula-math tooling (`scripts/code-tag-audit.md` and related scripts) parses **L5K text exports**, not `.ACD` binaries directly, and only two vintages are in scope at the reference site:

Export	What it's good for
`scripts/RS_3000_2024_APR_22.L5K` (2024, plaintext)	The math authority for `Formula_Calc` and band arithmetic — stale as a picture of what's *running*, but readable.
`scripts/RS_3000_RUNNING_2026-06-09.L5K` (2026, source-protected)	The current running program's tag *names* and structure; 88 of 90 routines are opaque `ENCODED_DATA` and cannot be read as ladder or Structured Text.

Full detail, including why neither export alone is trustworthy on its own for every kind of question, is in reference/01 — PLC and controller facts. No other export vintage or format (a raw `.ACD`, an export from a different controller family) has been fed through this tooling.

The PC Ring runs on

Requirement	Value	Enforced by
Operating system	Windows 10, version 1803 or newer, or Windows 11 (x64)	Ships the .NET Framework 4.7.2 runtime in-box; `docs/production-readiness/12_INSTALL_RUNBOOK.md` names this as the reason the boundary sits at 1803. Not a hard runtime check inside Ring itself — a pre-install and installer-checklist requirement (`scripts/Preflight-Check.ps1`, installer chapter 1 and 2).
.NET runtime	.NET Framework **4.7.2**	`Ring/Ring.csproj` `TargetFrameworkVersion`

Requirement	Value	Enforced by
Minimum screen resolution	**1280×720**	`ReadinessChecks.ClassifyScreenResolution` (`Ring/Services/Readiness/ReadinessChecks.cs`) — a hard PASS/FAIL check surfaced in the first-start wizard's Readiness page and in `Preflight-Check.ps1`. Ring's layout scales down to fit a smaller window but never grows past it.
Free disk space	Warn at **2048 MB**, critical at **512 MB** free (both configurable)	`AppSettings.DiskSpaceWarnMb` / `DiskSpaceCriticalMb` defaults (`Ring/Infrastructure/Configuration/AppSettings.cs`), evaluated by `DatabaseHealthService.EvaluateDiskSpace` and surfaced the same way in both the running app's own disk alarm and the wizard's Readiness check — the two are deliberately kept on identical thresholds so they never disagree.
Database engine	**SQLite, bundled**	`Ring/packages.config` (`System.Data.SQLite`, `System.Data.SQLite.Core`, `Stub.System.Data.SQLite.Core.NetFramework`) — no separate database server or install step; see reference/06 — Data dictionary.
Runtime prerequisite install	Microsoft Visual C++ 2015-2022 Redistributable (x64)	`docs/production-readiness/12_INSTALL_RUNBOOK.md`; needed by the bundled SQLite native components, not by Ring's own code.
CPU / RAM	No measured floor — Ring is a single-process desktop app; any x64 PC that runs Windows 10/11 comfortably is sufficient. 8 GB RAM is a sensible commissioning baseline (guidance, not a measured limit).	Not enforced anywhere in code or the readiness checks; stated here so a new-site estimate has an honest answer instead of an invented number.

This same table, in installer voice with the same figures, opens installer/01 — Before you go — that chapter is the one to hand to whoever is provisioning the PC; this row exists here so a plant engineer or integrator checking fit has it without opening the installer book.

What was not tested

Stated plainly, so it doesn't get papered over in a sales conversation or a new-site estimate:

- **Only one controller family, at one firmware revision, has ever run Ring in production or on a matched bench twin** (CompactLogix 1769-L36ERM, v20.12). Every other combination in this chapter is "expected to work because of how `libplctag` and Ring's tag model are built," not "verified."
- **Windows 10 versions older than 1803, and 32-bit Windows, are unsupported and unverified** — the site's install runbook draws the line at 1803 because that's where the in-box .NET Framework 4.7.2 boundary sits, not because anything older was tried and failed.
- **No `.ACD` binary format has been read by any Ring tooling** — only the two L5K text exports named above, from the one reference controller.
- **Multi-controller or multi-site topologies have not been exercised.** Ring's configuration and every gate in the write-path stack assume one controller per station.

For a new-site fit check

Before quoting or scoping a new install, confirm: the target controller is a ControlLogix-family Rockwell PLC reachable over EtherNet/IP (ideally the same CompactLogix family already proven); the install PC meets the OS/.NET/screen/ disk rows above; and — separately, and only if the controller-side batch logic needs work — whoever holds that scope has a Studio 5000 license that can open a v20-era project. None of this substitutes for the readiness checklist a real install runs through; see installer/01 — Before you go and installer/02 — Installing Ring.

9. Ring Remote View reference

Who this is for. Plant IT, integrators adding a site, a security reviewer checking the package's claims against its code, and a future developer who needs an exact value without re-reading every script.

Not a walkthrough. For the guided install see installer/07 — Ring Remote View (v1, LAN) and installer/07a — off-site (v2, Cloudflare Tunnel). For what an operator sees, see operator/10 — Remote viewing; for where this sits in the plant network, see reference/02 §2.4.

Ring Remote View is a **separate, optional** package — a pinned, hardened MeshCentral server plus a portable Node runtime, installed into `C:\Ring\RemoteView\`. It mirrors the one live Ring HMI session; it never starts a second Ring, never opens a second PLC connection, and no Ring file is touched by anything in it. Every table below comes from the package's own scripts and config template, the two runbooks, and a bench verification pass run **2026-09-04** — where that pass found something a runbook got wrong, or left unproven, this chapter says so.

9.1 Pinned constants

Item	Value
Product display name	`Ring Remote View` (a wrapper name — MeshCentral itself is not rebranded)
MeshCentral version	**1.2.4** (Apache-2.0, `engines.node >= 20`)
Node runtime	**Node 24 LTS ("Krypton"), win-x64, portable** — bundled, never installed machine-wide. Node 20 left maintenance 2026-04-30; the build refuses a line already past end-of-life. Node 24's own EOL: 2028-04-30.
Install root	`C:\Ring\RemoteView\`
HTTPS port (web UI)	**8443**
MPS port	**4433** — Intel AMT presence-server port, bound to `127.0.0.1`, no inbound rule. See §9.2.
HTTP redirect port	disabled (`redirPort: 0` — no plaintext listener exists)
Server service — **display name**	`MeshCentral`
Server service — **real service name**	**`meshcentral.exe`**, a node-windows daemon at `<root>\meshcentral\WinService\daemon\meshcentral.exe`. The SCM service name is the exe name, not "MeshCentral." Bench-confirmed 2026-09-04.
Agent service — display name	`Mesh Agent`
Agent service — binary	`C:\Program Files\Mesh Agent\MeshAgent.exe`

Item	Value
Device group	`Plant HMI`
Device group features	**4** (Record Sessions)
Device group consent	**9** (1 desktop notify + 8 desktop prompt)
Control mask	**12586504**
View-only mask	**12586760**

`test\Verify-RemoteViewStatic.ps1` asserts every value above across every script and the config template — see §9.7.

v2 pinned constants (Cloudflare Tunnel, optional)

Item	Value
cloudflared version (pinned)	**2026.7.3**
cloudflared asset	`cloudflared-windows-amd64.exe`
cloudflared SHA256 (pinned)	`8635DA433B6DF8194746E88ED9D2589566C20E38BFC2A80E431A348B7C765841`
cloudflared licence	Apache-2.0
Windows service	`cloudflared`, Automatic start, `--no-autoupdate` pinned into the ImagePath
Binary location	`C:\Ring\RemoteView\cloudflared\cloudflared.exe`
Outbound ports	TCP **443** (required), UDP **7844** (QUIC, optional)
Inbound ports added	**none, on any NIC** — asserted by the installer

---

9.2 Listeners and firewall

What actually listens (bench-observed, 2026-09-04)

Port	Observed
8443	Listening on `0.0.0.0:8443` and `[::]:8443`, TLS 1.3. Plain HTTP gets no response at all.
4433 (MPS)	**No socket at all**, not merely "bound to loopback." MeshCentral 1.2.4's `mpsserver.js` line 42: `if ((args.lanonly != true) && (args.mpsport !== 0))` — in LAN-only mode (always on here) the AMT MPS listener never starts. The runbooks' "bound to `127.0.0.1`" is conservative; the bench truth is "not listening."

Port	Observed
Plant/PLC NIC	Nothing inbound, ever.

Inbound firewall rules

Rule	Created by	Scope	Purpose
`Ring Remote View (HTTPS 8443)`	`New-RemoteViewFirewallRule.ps1` (installer calls it unless `-NoFirewall`)	TCP 8443 inbound, `RemoteAddress = LocalSubnet`, `Profile = Domain,Private` (never `Public`/`Any`), optionally one NIC via `-InterfaceAlias`	The only inbound surface this package adds on purpose
`Mesh Agent WebRTC Traffic`	**The Mesh Agent's own installer** (`meshagent64 -fullinstall`) — not created or documented by this package or either runbook. Bench finding, 2026-09-04.	Allow, UDP, any port, program `MeshAgent.exe`, profiles **Domain+Private+Public**	Unused (`webRTC: false` always), but present on every install with the local agent. Disable with `Disable-NetFirewallRule -DisplayName 'Mesh Agent WebRTC Traffic'`, or accept it explicitly. Removed automatically by `MeshAgent.exe -fulluninstall` — bench-confirmed gone after uninstall.

No rule is ever created for MPS 4433.

> **Dual-homed:** `LocalSubnet` alone is not enough if the plant subnet is also "local" — bind the rule to the office NIC with `-InterfaceAlias`; the installer requires both `-LanCidr` and `-InterfaceAlias` together on a multi-homed box and asserts the rule's interface filter matches. **Profile:** the rule is `Domain,Private` only — if the NIC's `Get-NetConnectionProfile` category is **Public**, the rule does nothing and 8443 is unreachable with no error (the installer warns in its summary). The 2026-09-04 bench run hit exactly this and used `-NoFirewall` instead, since the lab's only NIC was Public.

v2 is outbound-only. `cloudflared` dials out on TCP 443 (and UDP 7844 for QUIC). The v2 installer creates no inbound rule and asserts that none exists referencing `cloudflared`.

9.3 Files and folders on the HMI PC

```
C:\Ring\RemoteView\
    node\                          portable Node 24 LTS "Krypton" (not machine-wide)
    meshcentral\                   MeshCentral 1.2.4 (agents\ pruned to Windows x64)
        meshcentral-data\        JUNCTION -> C:\Ring\RemoteView\meshcentral-data\
        meshcentral-recordings\  JUNCTION -> C:\Ring\RemoteView\meshcentral-recordings\
```

```
meshcentral-data\            real folder: config.json, certificates, database
meshcentral-recordings\      real folder: session .mcrec recordings
manifest.json                the payload SBOM
payload-inventory.json       path + size + SHA256 of every payload file
cloudflared\                 v2 only — see below
```

The junctions are load-bearing — do not delete them. MeshCentral resolves its data folder relative to the package location, and `winservice.js` forwards only `user/port/aliasport/mpsport/mpsaliasport/redirport/exactport/debug` to the Windows service — not `--datapath`. Deletes in this package are junction-aware: a link is unlinked first, so a recursive delete can never follow it into real data.

Item	Detail
Certificates	`meshcentral-data\` holds root / webserver / agentserver / codesign / mps cert+key pairs, generated by MeshCentral itself on first start (no synchronous `--cert` step). `webserver-cert-public.crt`'s subject CN is the rendered certificate name — see §9.5 — not the product name and not the bare NetBIOS name.
Recordings	Capped **90 days / 500 files / 20,480 MB**, scoped to `Plant HMI` only, indexed, protocol `2` (desktop) only. Evidence — fold into the plant's backup/retention policy rather than pruning ad hoc.
Shortcuts	`...\Start Menu\Programs\Ring Remote View.url` and `C:\Users\Public\Desktop\Ring Remote View.url`, both to `https://<hostname>:8443/`.
v2: `cloudflared\`	`cloudflared.exe` (pinned, SHA256-verified) and `install-state.json` (version, SHA256, hostname, patched-key list, edge evidence, a non-reversible token fingerprint — never the token itself).
v2: config backups	`meshcentral-data\config.json.bak-<yyyyMMddHHmmss>`, written before every v2 patch. Never pruned.

Rollback deny-list. `C:\Ring\Icons`, `C:\Ring\Pictures`, `C:\Ring\App` and `C:\Ring` itself can never be deleted by a rollback — only artefacts *this run* created (install root, junctions, `config.json`, both services, the firewall rule, the shortcuts, and any accounts it created) are ever undone.

9.4 Accounts and rights

The rule the design depends on: a MeshCentral **site administrator bypasses every device-group rights mask entirely.** So every operator is created with `--rights none` — zero *server* rights — and access is granted only on the `Plant HMI` device group, per account, by numeric mask. The admin account stays with plant management, never handed to an operator.

Account type	Server rights	Device-group mask	Can drive?	Terminal/files/registry/software
Site administrator	Full (`siteadmin = 4294967295`, i.e. `0xFFFFFFFF`)	n/a — bypasses masks	yes	yes — exactly why this never goes to an operator
Operator, **control**	`none`	**12586504**	yes	denied
Operator, **view-only** (default)	`none`	**12586760**	no — input discarded	denied

```
   8          REMOTECONTROL
 512          NOTERMINAL
1024          NOFILES
2048          NOAMT
4194304       NOREGISTRY
8388608       NOSOFTWARE
--------
12586504      control mask
    +256      REMOTEVIEWONLY
--------
12586760      view-only mask
```

Bit **values** are the `MESHRIGHT_*` constants in `meshuser.js` @ 1.2.4, lines 40–64. MeshCtrl **flag names** map to the same bits in `meshctrl.js` @ 1.2.4, `AddUserToDeviceGroup`, lines 1648–1665: control = `--remotecontrol --noterminal --nofiles --noamt --noregistry --nosoftware`; view-only = the same **plus** `--desktopviewonly` (the flag for the 256 bit — `--viewonly` is **not** a real flag).

Erratum, corrected on purpose. An earlier brief quoted `12587016` / `12587272` — the same flags, mis-added by exactly 512. This package ships the correct masks above, and the static gate fails the build if either superseded literal reappears anywhere.

What MeshCtrl actually shows (`ListUsersOfDeviceGroup`, bench-observed): `ringadmin` → `FullAdministrator`; `jbaker` (control) → `RemoteControl, NoTerminal, NoFiles, NoAMT`; `shiftlead` (view-only) → `RemoteControl, RemoteViewOnly, NoTerminal, NoFiles, NoAMT`. **Note the gap:** MeshCtrl's text view never names the `NoRegistry`/`NoSoftware` bits — only the numeric mask read-back (12586504/12586760) confirms them.

Enforced server-side, not just a hidden tab. The 2026-09-04 pen-test pass (§9.7) confirmed the relay itself refuses to route a restricted account to the agent for terminal, file and command requests.

2FA is domain-wide, not per-account. `force2factor: true` applies to every account — MeshCentral 1.2.4 has no per-account 2FA setting. It is enforced **post-login**: a successful password login is signalled to force 2FA *enrolment* afterward, not refused outright (`webserver.js`, feature flag `0x00040000`). `skip2factor` is deliberately **absent**.

9.5 Configuration keys as rendered by the installer

Rendered by `Install-RemoteView.ps1` from `config-template.json` into `meshcentral-data\config.json`. `<TOKEN>` = a placeholder.

`settings.*`

Key	Rendered value	Why
`cert`	`<HOSTNAME>` — **lowercase, WITH A DOT**: the FQDN if DNS gives one, else `<hostname>.local`	MeshCentral 1.2.4 only offers 2FA enrolment in the UI when `certificates.CommonName` contains a dot (`webserver.js`, feature flag `0x00001000`). A bare NetBIOS name never has one, so nobody could ever enrol while `force2factor` blocked every feature until they did — a dead lock. Fixed on branch `docs/remote-access-guide`; bench-rendered as `laptop-t8d4hur8.local`.
`npmpath`	absolute path to the bundled `node\node_modules\npm\bin\npm-cli.js`	Belt-and-suspenders: if MeshCentral ever falls to its startup module-install path it must use the bundled npm, never a missing machine-wide one.
`LANonly`	`true`	No tunnel, no internet exposure by default. Also makes the served UI build WebSocket URLs from `window.location.hostname`, not the cert CN — load-bearing **for** v2, not just a v1 default.
`port` / `redirPort`	`<HTTPS_PORT>` (8443) / `0`	The web UI port; no plaintext listener exists at all.
`mpsPort` / `mpsPortBind`	`4433` / `"127.0.0.1"`	See §9.1/§9.2 — never reachable off-box; in LAN-only mode never even opens a socket.
`exactPorts`	`true`	Refuses to silently walk to another port on a collision — loud failure, not a surprise listener. Also required by v2: cloudflared's origin hard-codes `127.0.0.1:8443`.
`userAllowedIP` / `agentAllowedIP`	`<LAN_CIDR>,127.0.0.1`	The source-IP gate — set again under `domains[""]` below; both are independent and must move together.
`selfUpdate` / `noAgentUpdate`	`false` / `1`	A validated HMI PC does not silently take new server or agent code.
`allowLoginToken`	`false`	No login-token bypass of password/2FA.

Key	Rendered value	Why
`allowFraming`	`false`	The UI is never framed by another page.
`webRTC`	`false`	This is *why* the "Mesh Agent WebRTC Traffic" rule (§9.2) is unused even though it exists.
`plugins.enabled`	`false`	No plugin system.
`maxInvalidLogin` / `maxInvalid2fa`	`{time:10, count:5, coolofftime:10}` each	Brute-force throttle on password and 2FA attempts.

`domains[""].*`

Key	Rendered value	Why
`newAccounts`	`false`	Self-registration off. Bench-confirmed: `/createaccount`, `/register`, `/signup` all HTTP 404, no form.
`newAccountsRights`	`notools, nonewgroups, nonewdevices, locksettings`	Dormant while `newAccounts` is false — documents intent if ever flipped.
`guestDeviceSharing`, `agentSelfGuestSharing`, `allowSavingDeviceCredentials`, `ipkvm`, `mstsc`, `novnc`, `ssh`, `myServer`, `agentInviteCodes`, `localSessionRecording`	all `false`	Every unused feature is off, not merely unused.
`userAllowedIP` / `agentAllowedIP`	`<LAN_CIDR>,127.0.0.1`	The **second, independent** copy — `webserver.js` evaluates the global and per-domain lists separately.
`allowedOrigin`	`<ALLOWED_ORIGINS>` — lowercase array: bare hostname, certificate name, `localhost`, `127.0.0.1`, primary LAN IPv4	MeshCentral closes `/control.ashx` with cause `invalidorigin` unless the browser's Origin hostname is listed here — or, with no list, unless it equals the certificate CommonName **case-sensitively**, which a lowercase browser Origin never matches on an uppercase Windows machine name. This was a real, bench-observed blocking defect before the fix — see §9.8.
`userSessionIdleTimeout` / `logoutOnIdleSessionTimeout`	`30` / `true`	Idle browser sessions log out.
`clipboardGet` / `clipboardSet`	`false` / `false`	No clipboard bridging between browser and HMI desktop.
`passwordRequirements.min/upper/lower/numeric/nonalpha`	`12/1/1/1/1`	Password complexity floor.

Key	Rendered value	Why
`passwordRequirements.banCommonPasswords` / `oldPasswordBan`	`true` / `5`	Common-password ban (triggers the bundled `wildleek` module); no reusing the last five.
`passwordRequirements.force2factor`	`true`	Domain-wide, post-login — see §9.4.
`passwordRequirements.otp2factor` / `backupcode2factor`	`true` / `true`	The only two 2FA methods enabled — a plant LAN has no SMTP/SMS/Duo reachability, so any other method would offer enrolment paths that can never complete (all left `false`).
`passwordRequirements.skip2factor`	**deliberately absent**	Its presence would let an IP range bypass 2FA entirely.
`desktop.viewonly` / `disableconnectall`	`false` / `true`	Control is granted **per account** via the rights mask, never domain-wide; no bulk "connect to all" action.
`terminal.sshConnect` / `files.sftpConnect`	`false` / `false`	No SSH/SFTP bridging on top of the already-denied tabs.
`userConsentFlags.desktopnotify` / `desktopprompt`	`true` / `true`	Matches the device-group numeric consent value `9` (1 + 8).
`consentMessages.Desktop`	`"{0} ({1}) is asking to view or take control of this Ring HMI. Allow?"`	The exact text shown at the panel.
`consentMessages.Terminal` / `Files`	`"...This should never appear - deny it and report it."`	If either ever appears, the rights model has failed — an incident, not a click-through.
`consentMessages.consentTimeout`	`60` (seconds)	How long the panel has to answer.
`consentMessages.autoAcceptOnTimeout` / `autoAcceptIfNoUser` / `autoAcceptIfLocked`	all `false`	**An unattended panel denies the connection.** Bench-confirmed: after 61 s unanswered, the session returned to Disconnected with no screen ever shown.
`agentConfig`	`["disableUpdate=1", "noUpdateCoreModule=1"]`	The local agent is frozen — no silent code change on a validated HMI PC.
`sessionRecording.*`	scoped `true`, indexed, `protocols:[2]`, `90` days / `500` files / `20480` MB	Desktop only, `Plant HMI` only, capped as shown.
`limits.*`	`MaxDevices 8, MaxUserAccounts 25, MaxUserSessions 16, MaxAgentSessions 8`	Sane ceilings for a single-HMI deployment.

v2 — the five keys it patches, and why

Each is justified line-by-line against the MeshCentral 1.2.4 source in `REMOTE_ACCESS_V2_RUNBOOK.md` §4A; this is the one-line summary.

#	Key	Rendered value	Reason
1	`settings.trustedProxy`	`"127.0.0.1,::1"`	The **only** key that makes MeshCentral trust cloudflared's forwarded `cf-connecting-ip`, so the audit log and login-throttle bucket use the real client IP instead of `127.0.0.1` for every tunnel login.
2	`settings.cookieIpCheck`	`"lax"`	Explicit (matches the 1.2.4 default) so the behaviour is documented, not discovered — accepts a `/24` move once `trustedProxy` is on, forces re-login on a bigger one.
3	`settings.userAllowedIP`	Widened with approved off-site CIDRs, **or** removed via `-NoUserIpAllowList`	A deliberate Section-2D decision — see §9.8. Widened, the LAN CIDR and `127.0.0.1` are preserved, never replaced.
4	`domains[""].userAllowedIP`	Same change, again	Independent of #3 — both must move together or logins die before authentication runs.
5	`domains[""].allowedOrigin`	Widened: public FQDN + this machine's names + loopback	Without the public hostname here, `/control.ashx` — every desktop session depends on it — closes with `invalidorigin`.
—	`_ringRemoteViewV2`	Documentation-only array	Underscore-prefixed top-level keys are ignored by MeshCentral; a review aid only.

v2 — must NOT set (the installer refuses to proceed if any is present)

Key	What it would do
`settings.tlsOffload` (any value)	Breaks v2 outright — MeshCentral serves plain HTTP on 8443, cloudflared's `https://` origin fails; also drops the cookie's `Secure` flag.

Key	What it would do
`settings.trustedProxy = "CloudFlare"`	Wrong for a same-machine tunnel — matches the *TCP peer* against Cloudflare's ranges, but the peer here is `127.0.0.1`, so `cf-connecting-ip` is silently ignored.
`settings.trustedProxy = true`	Trusts forwarded headers from **every** peer — any LAN host could forge `X-Forwarded-For` past `userAllowedIP`. Also out of schema (typed `string`).
`settings.aliasPort` (e.g. `443`)	Poisons LAN agent discovery — the scanner tells the local agent to connect to 443, where nothing listens.
`domains[""].certUrl`	Solves a non-problem; causes outbound cert fetches and agent churn on every Cloudflare edge-cert rotation.
`ignoreAgentHashCheck` (either scope)	Disables the agent's TLS pinning — "for debugging only" per the schema's own text.
`domains[""].dns = <public fqdn>`	On the default domain this takes the **whole server offline** — `dns` is for extra sub-domains, never `""`.
`settings.relayDNS = <public fqdn>`	Replaces the MeshCentral UI with the app web-relay for exactly the hostname you meant to serve.
`settings.LANonly=false`, `settings.WANonly=true`	Breaks the tunnel; `WANonly` also kills local-relay and the LAN discovery scanner.
`settings.strictTransportSecurity`	Not in the 1.2.4 schema at all, and locks LAN users out of the self-signed cert's click-through.
`allowFraming` variants	Unnecessary (Access uses a full-page redirect, not an iframe) and widens the CSP `frame-ancestors` surface.
`domains[""].allowedOrigin = true`	Disables the origin gate entirely instead of enumerating hostnames.
`settings.cookieIpCheck = "none"`	Unbinds cookies from the client IP just as `trustedProxy` makes that IP meaningful.
`agentPort/agentAliasPort/agentAliasDNS`	Opens a second agent-only listener — there is exactly one, local agent, and it already reaches 8443.

9.6 Scripts and parameters

`Build-RemoteViewPackage.ps1` — packager-side payload builder

Parameter	Default	Meaning
`-OutDir`	`<script dir>\dist`	Build output directory.
`-NodeMajor`	`24`	Node LTS major to stage. Fails if past end-of-life.

Parameter	Default	Meaning
`-ProbeOnly`	off	Resolve + HEAD-probe URLs, then stop. Downloads nothing.
`-SkipZip`	off	Stage `dist\payload` + manifest, skip the `.zip`.
`-Force`	off	Rebuild from scratch.

`Install-RemoteView.ps1` — target-side installer

Parameter	Default	Meaning
`-AdminUser`	`ringadmin`	Site administrator account id.
`-AdminPassword`	generated	SecureString; omitted, a 20-char password is shown once.
`-AdminPasswordFile`	—	UTF-8 file, first line = password; deleted after reading.
`-LanCidr`	auto-detected	Allowed LAN range. Required with `-InterfaceAlias` on a multi-homed box.
`-InterfaceAlias`	—	Office NIC the firewall rule binds to; required on a multi-homed box.
`-HttpsPort`	`8443`	Pinned.
`-Operators`	`@()`	`@{User='...'; Pass='...'; ViewOnly=$true/$false}` array.
`-OperatorSpec`	—	Flat form: `"jbaker:control,shiftlead:view"` (defaults view-only).
`-PayloadZip`	`<script dir>\dist\RingRemoteView-payload.zip`	Never expanded with `Expand-Archive` — extracted entry by entry and proven against the archive's own listing and `payload-inventory.json` before anything is copied.
`-PayloadDir`	—	Alternative: an already-expanded directory.
`-InstallRoot`	`C:\Ring\RemoteView`	Pinned.
`-NoFirewall`	off	Skip the inbound rule.
`-NoAgent`	off	Skip the local Mesh Agent.
`-Reinstall`	off	Adopt-and-replace an existing service instead of aborting.
`-TempExpandRoot`	`$env:TEMP`	Payload expansion directory.

Parameter	Default	Meaning
`-VerifyPayloadHashes`	off	Re-hash every extracted file (adds minutes under on-access AV).
`-HealthTimeoutSeconds`	`180`	Wait for `https://127.0.0.1:<port>` to answer.

`Provision-RemoteView.ps1` — device group + operator accounts

Parameter	Default	Meaning
`-InstallRoot` / `-HttpsPort`	`C:\Ring\RemoteView` / `8443`	
`-AdminUser` / `-AdminPassword`	mandatory	Site administrator id / SecureString.
`-Operators`	`@()`	Same shape as the installer's.
`-GroupName`	`Plant HMI`	Pinned.
`-EmitInviteLink`	off	Time-bounded background-agent invite link — off by default (a never-expiring link is a standing credential).
`-InviteLinkHours`	`24`	Must be `> 0`.

`Uninstall-RemoteView.ps1`

Parameter	Default	Meaning
`-InstallRoot` / `-HttpsPort`	`C:\Ring\RemoteView` / `8443`	Port is used to find the firewall rule by name.
`-KeepData`	off	Preserve `meshcentral-data\` and `meshcentral-recordings\`.
`-Force`	off	No confirmation prompt.

`New-RemoteViewFirewallRule.ps1`

Parameter	Default	Meaning
`-Port`	`8443`	Pinned.
`-InterfaceAlias`	—	Restrict to one NIC.
`-Remove`	off	Remove instead of create.

`Install-RemoteViewTunnel.ps1` — v2 companion

Parameter	Default	Meaning
`-TunnelToken`	—	Tunnel token (String/SecureString). Required unless `-TunnelTokenFile` or `-Uninstall`. Stored in the service ImagePath — an accepted residual.
`-TunnelTokenFile`	—	UTF-8 file, first line = token; deleted after reading.
`-Hostname`	—	Public FQDN the tunnel publishes — no scheme/port/path.
`-RemoteAllowedIP`	`@()`	Approved off-site egress CIDRs, **added** to the v1 allow-list. Choose this or `-NoUserIpAllowList` — the installer refuses to guess.
`-NoUserIpAllowList`	off	Removes `userAllowedIP` entirely; Access becomes the whole perimeter, LAN users lose the source-IP gate too.
`-ExtraAllowedOrigin`	`@()`	Extra bare hostnames for `allowedOrigin`.
`-CloudflaredVersion`	`2026.7.3`	Changing it **requires** `-ExpectedSha256`.
`-ExpectedSha256`	pinned hash	No way to skip verification.
`-PayloadPath`	—	Already-downloaded binary (air-gapped) — still hash-verified.
`-InstallRoot` / `-HttpsPort`	`C:\Ring\RemoteView` / `8443`	v1 root and port.
`-EdgeWaitSeconds`	`120`	Bounded wait for evidence of a real edge connection.
`-AllowUnconfirmedEdge`	off	Downgrades "no edge evidence" to a warning — never suppresses an observed failure.
`-HealthTimeoutSeconds`	`180`	Matches v1.
`-Uninstall` / `-Force`	off / off	Remove the service, revert only the keys v2 added / no confirmation prompt.

`Test-RemoteViewTunnelSmoke.ps1` — v2 staged smoke

Parameter	Default	Meaning
`-CloudflaredExe`	staged path, else PATH	Path to `cloudflared.exe`.
`-HttpsPort`	`8443`	Origin is always `https://127.0.0.1:<port>`.

Parameter	Default	Meaning
`-BudgetSeconds`	`240`	Hard ceiling on the exposure window (raised from 120 after a bench-found defect — §9.7).
`-TunnelStartTimeoutSeconds`	`45`	Wait for cloudflared to print its ephemeral hostname.
`-EvidenceDir`	`<script dir>\dist`	Transcript location (git-ignored).
`-NoDelay`	off	Skip the 5 s abort window on the exposure banner.

`Verify-RemoteViewStatic.ps1` — static gate

`param()` — **no parameters.**

```
powershell -NoProfile -ExecutionPolicy Bypass -File .\test\Verify-RemoteViewStatic.ps1
```

Runs entirely offline: installs nothing, starts no service, touches no firewall. See §9.7 for the current pass result.

9.7 Verification evidence

Everything here was observed, in order, on a bench pass **2026-09-04** on the developer laptop (Windows 11 Home, single NIC classified **Public**), source tree `main @ fd4e5173`. Items not exercised are marked **NOT VERIFIED** — that phrasing is deliberate; do not read it as a pass.

Check	Result
Static gate, before the certificate/`allowedOrigin` fix	PASS, 0 failures
Static gate, after the fix (branch `docs/remote-access-guide`)	PASS, 0 failures — see `remote-view/test/Verify-RemoteViewStatic.ps1` output for the current check count
`Build-RemoteViewPackage.ps1 -ProbeOnly`	PASS — MeshCentral pin 1.2.4 confirmed at npm; Node resolved to v24.20.0 (EOL 2028-04-30); nodejs.org URLs HEAD 200
Existing payload `RingRemoteView-payload.zip` (built 2026-07-31)	115,737,143 bytes, SHA-256 `be01f3ca00c84ce18df9e802bda82f03ab7a7a3774104aa4ebcdaa929bf2191f`, matches `manifest.json` (Node v24.18.1, MeshCentral 1.2.4, four extra modules)

Fresh install with the fixed package (authoritative). `Install-RemoteView.ps1` (`-NoFirewall`, with the local Mesh Agent), branch `docs/remote-access-guide`: **exit 0, 175 s** (an earlier run before the cert/origin fix took 203 s under the same conditions — expect longer on an HMI PC with on-access AV scanning the ~394 MB payload). Printed: certificate name `laptop-t8d4hur8.local`; `allowedOrigin` = `laptop-`

`t8d4hur8, laptop-t8d4hur8.local, localhost, 127.0.0.1, 192.168.1.206`; server cert `subject=CN=laptop-t8d4hur8.local`; agent service Running; shortcuts for `https://laptop-t8d4hur8:8443/`.

Browser walkthrough as operator `jbaker` (control mask) — observed and screenshotted

Step	Observed
Login page	Title "Ring Remote View - Login"; footer credits MeshCentral 1.2.4 (Apache-2.0).
First login, one-time password	"Password change requested," two password fields, then the main UI loads.
2FA enrolment	My Account → **"Account security"** → **"Manage authenticator app"**: QR code, secret, 6-digit token field. A valid code: "Authenticator app activation successful."
Next login	A second page asks for the 6-digit token.
Device list	Group **Plant HMI**, device **LAPTOP-T8D4HUR8**, "Agent, Powered." Operator tabs: **General, Desktop, Events only.**
Desktop → Connect	"Connected"; area reads **"Waiting for user to grant access..."** At the panel: **"jbaker (jbaker) is asking to view or take control of this Ring HMI. Allow?"** (Allow / Deny, plus an "auto-accept next 5 minutes" checkbox).
Nobody answers	After **61 s**, status returns to **Disconnected**; no screen content was ever shown. **PASS — fails closed.**

Not exercised: clicking Allow, view-only input-discard, registry/software tabs, off-subnet. The auto-accept checkbox is stock MeshCentral — tell operators to leave it unticked unless the shift agrees.

Uninstall — verified. `Uninstall-RemoteView.ps1 -Force`: exit **0**, **50 s**. Removed the Mesh Agent (`-fulluninstall`) and MeshCentral service, printed "No Ring Remote View firewall rules found" (this run used `-NoFirewall`), "UNINSTALL COMPLETE." Confirmed afterward: no services, install root gone, `C:\Program Files\Mesh Agent` gone, **the "Mesh Agent WebRTC Traffic" rule gone**, no 8443 listener, shortcuts gone.

Penetration-test items — `REMOTE_ACCESS_RUNBOOK.md` §8, observed via MeshCtrl

#	Attack	Account	Observed	Verdict
2	File transfer (Upload/Download)	jbaker (control)	`Connecting...` then **`Unable to route`**; nothing transferred	DENIED
2 (control case)	same	ringadmin (admin)	Connected, download completed	allowed, as expected
3	`RunCommand --run whoami`	jbaker / shiftlead	**`Access denied`** (both)	DENIED
3 (control case)	same	ringadmin	`OK`	allowed, as expected

#	Attack	Account	Observed	Verdict
4	`Shell` (interactive terminal)	jbaker	`Connecting...` then **`Unable to route`**; no shell in 20 s	DENIED
server rights	`ListUsers`	jbaker	**`Access denied`** (`--rights none`)	DENIED
9	Self-registration	unauthenticated	**HTTP 404** each, no form	DENIED
10	MPS exposure	—	no listener on 4433	PASS
7	Consent bypass / unattended panel	—	confirmed in the walkthrough: 61 s timeout, Disconnected, no content shown	**PASS — fails closed**
1	Raw terminal relay (`p=1`)	—	not run raw; test #4 exercises the same relay gate	partially covered
5	Registry / software tabs	—	**NOT VERIFIED** — no headless probe available	NOT VERIFIED
6	View-only sends input	—	**NOT VERIFIED** headlessly	NOT VERIFIED
8	Off-subnet reach	—	**NOT possible** on a single machine	NOT VERIFIED

Rights masks are enforced server-side — the relay itself refuses to route a restricted account, not merely a hidden tab. **The full §8 table on a clean VM with a second machine is still owed for items 1, 5, 6 and 8.**

v2 — what was verified:

Item	Result
`cloudflared-windows-amd64.exe` 2026.7.3	54,213,360 bytes, SHA-256 `8635DA433B6DF8194746E88ED9D2589566C20E38BFC2A80E431A348B7C765841` — **matches the pin**; `--version` confirms the tag.
`Test-RemoteViewTunnelSmoke.ps1` (after the fix below)	**PASS — every gated probe passed, 59 s.** Hostname resolved after 30 s; `GET /` HTTP 200; CSP `connect-src` named the tunnel's own hostname (Host header survives the hop); `/control.ashx`, `/meshrelay.ashx`, `/2fahold.ashx`, `/echo.ashx` upgrades all PASS; the Origin gate closed an unknown Origin with `invalidorigin`; tunnel process confirmed gone at the end.
`Install-RemoteViewTunnel.ps1` itself	**NOT run** — needs a real Zero Trust tunnel token and domain.
Cloudflare Access policy, full off-site smoke, v2 penetration test	**OUTSTANDING.**

Defect found and fixed in the smoke script. As shipped, it probed the new hostname before DNS had propagated and every probe failed with an opaque exception. Fixed on branch `docs/remote-access-guide`: waits up to 90 s for DNS, retries the login-page GET for up to 60 s through transient Cloudflare 530/1033 errors, unwraps the real inner exception, and raised the default `-BudgetSeconds` from 120 to **240**. The PASS above is the re-run.

Other verified facts. The Inno Setup GUI wrapper (`RingRemoteView.iss`) has **never been compiled** — no ISCC installed, no `RingRemoteView-Setup.exe` anywhere in the tree; **the PowerShell installer is the only proven install path.** Multi-homed detection ignores loopback and `169.254.*` APIPA addresses. **IPv6 caveat:** browsing the bare hostname from the laptop itself answered **HTTP 401 before any login page**, because the name resolved to IPv6 addresses first and `userAllowedIP` lists only the IPv4 CIDR and `127.0.0.1` — make sure the HMI name resolves to IPv4 for LAN clients, or browse the IPv4 address directly (`allowedOrigin` permits that). **A PowerShell trap, not MeshCentral's:** `Invoke-WebRequest`/`ClientWebSocket` with a `ServerCertificateValidationCallback` scriptblock fails non-interactively ("There is no Runspace available to run scripts in this thread") — use `curl.exe -k https://127.0.0.1:8443/` or a real browser for a hand check instead.

9.8 Security posture and residual risks

Defence in depth, any one gate stopping an attacker on its own: (1) the network perimeter — LAN-only (v1), or Cloudflare Access authenticating at the edge (v2); (2) MeshCentral's own accounts — login, `force2factor`, the panel consent prompt, session recording; (3) the rights mask, enforced server-side (confirmed in §9.7).

Fixed defects (bench 2026-09-04 — read before trusting an unmodified `main` build):

Defect	Root cause	Fix
Login succeeded, then "Invalid origin in HTTP request, click to reconnect."	`settings.cert` was `$env:COMPUTERNAME` (always uppercase); the Origin check is case-sensitive and browsers always lowercase the Origin.	Cert name rendered lowercase; `allowedOrigin` added (hostname, cert name, localhost, 127.0.0.1, LAN IPv4).
2FA enrolment was **impossible** while `force2factor` made it **mandatory**.	MeshCentral only shows "Account Security" when the cert CommonName contains a dot; a bare NetBIOS name never does.	Cert name rendered **with a dot** (FQDN, or `<hostname>.local`).
A failed install sometimes printed `ROLLBACK INCOMPLETE` for folders that had actually been removed.	The rollback closures for `meshcentral-data`/`meshcentral-recordings` used `.GetNewClosure()` blocks that couldn't see the script's own `Remove-TreeRobust` function.	The function is now captured into the closure.

All three are fixed on branch `docs/remote-access-guide` (the static gate gained one new assertion for the fix). **Verify which branch a given build came from** before assuming 2FA enrolment or a plain-hostname browser session will work.

Residual risks — v1. MeshCtrl takes credentials on its command line during provisioning, briefly visible in the process list (console-only, install-time only); the Inno wrapper passes `-AdminPassword` as an argument — same residual; a site administrator can do everything, and the control is procedural — that credential stays with plant management.

Residual risks — v2 (only where the tunnel is installed). The tunnel token is a standing credential in the service ImagePath (registry) — rotate it in the Zero Trust dashboard if the box is suspect. `trustedProxy: "127.0.0.1,::1"` means any **local** process can forge a source IP — accepted only because `cloudflared` is the sole local client of 8443 on a single-purpose HMI, and `force2factor` still stands behind it. Cloudflare sits in the trust path (edge-decrypt, re-encrypt to origin) — the Tailscale alternative avoids this if policy forbids a third party. Double authentication is intentional: MeshCentral has no key consuming an Access identity header, so Access is a perimeter, not single sign-on. `cookieIpCheck: "lax"` forces re-login on a roaming IP move outside its `/24` — correct behaviour, not a fault to chase. The v2 install restarts MeshCentral, dropping any live LAN session; Ring and the PLC are untouched.

Still outstanding — a verification gap, not a residual risk. Pen-test items 1 (partial), 5, 6, 8 on a clean two-machine VM; the entire v2 off-site chain (Access policy, off-site smoke, v2 penetration test); whether the Inno Setup wrapper compiles at all.

9.9 Patch policy and licences

Component	Pin mechanism	To move it
MeshCentral	Pinned in `Build-RemoteViewPackage.ps1`; `selfUpdate:false`; agent frozen (`agentConfig`, `noAgentUpdate:1`)	Watch advisories. On a security release: bump the pin, rebuild, re-run the static gate, **re-run the penetration test**, redeploy in a window. Never `npm update` in place.
Node runtime	The build refuses a line already past EOL	Bump `-NodeMajor` as the pinned line nears EOL.
`cloudflared` (v2)	Pinned version + SHA256 in the installer; auto-update off in the service ImagePath	`-Uninstall`, reinstall with new `-CloudflaredVersion` **and** `-ExpectedSha256` (refuses a bump without it); re-run the staged and relevant off-site smoke lines.
Access policy (v2)	A live policy, not a pinned artefact	Review allowed emails/groups, session duration and MFA on the same cadence as user off-boarding — a drifted policy is a silent hole.

AV/EDR. MeshCentral is widely flagged as a post-compromise RMM tool — support an allow-list request with `dist\manifest.json` (the SBOM). Concrete example: this laptop's Defender history shows `MeshService.exe` quarantined 2026-08-20 as RMM tooling; the 2026-09-04 install needed a Defender exclusion for the scratch directory and `C:\Program Files\Mesh Agent` (removed after). `cloudflared.exe` is legitimately signed but some EDR flags it by category too — raise both with IT before install day.

Licence file	Contents
`remote-view\NOTICE`	Attribution + Apache-2.0 statement of changes.

Licence file	Contents
`remote-view\THIRD-PARTY-LICENSES.txt`	Apache-2.0 full text, Node.js MIT notice, cloudflared licence (also Apache-2.0).
`dist\manifest.json`	The SBOM — the artefact an AV/EDR allow-list request should cite.

---

90. Error code reference

Ring's error dialogs can carry a short stable code — `RING-DB-001`, for example — in a small selectable field next to the plain-language message. The code is deliberately the one thing about a failure that never changes: the sentence beside it is translated and can be reworded release to release, but the code is not. It is written to the application log ahead of the failure line it belongs to (`[RING-DB-001] [UI] save the formula failed`), and — when Ring recognizes it — the dialog offers an **Open help** button that jumps straight to that code's entry below, the offline copy of this page first and the public website as the fallback.

Only some dialogs carry a code today; a dialog with none is unchanged, one sentence and OK. A code, once minted, is a published deep link (`HelpLinkResolverService.ForErrorCode`, `HelpLinkHub.ErrorCodeUrl`) — see the closing note.

This page is generated by hand from the registry itself (`Ring/Services/Health/RingErrorCodes.cs`), not from a template — every `const` in that file, every title in its `Titles` dictionary, has an entry below. **33 codes** are registered as of this chapter; six of them (`RING-DB-004`, `RING-PLC-002`, `RING-PLC-004`, `RING-IO-003`, `RING-IO-004`, `RING-CFG-002`) are reserved — defined in the registry, well formed, but not yet raised at any call site in this tree. They are listed here anyway because the registry, not "is it wired up today," is what a future call site will pick from; each entry says plainly that it is reserved.

Format

`RING-<AREA>-<NNN>` — an area from a fixed list (`DB`, `PLC`, `IO`, `CFG`, `NET`, `UI`, `EXP`, `RPT`) and a zero-padded three-digit sequence within it. `RingErrorCodes.IsWellFormed` is the exact pattern; `RingErrorCodesTests` pins both the pattern and that every code and title is unique.

Where a code appears

- **In a dialog**, via `Ring.Views.FriendlyError.Show(..., errorCode: RingErrorCodes.SomeCode)` — the code field plus the **Open help** button when a help resolver is registered (`FriendlyError.ShouldOfferHelp`).
- **In the application log**, one line ahead of the failure detail, from the same `FriendlyError.Show` call and from services that log a code without a dialog (the export job, the scheduled export, the station backup path).
- **On the Help → System self-test screen**, next to any row that is not green (`HealthCheckComposer`, one code per failing check, wired to the matching row's **Help** link).
- **In a support bundle's `manifest.txt`**, wherever a collected item itself failed or was skipped.

The Open-help button

`FriendlyError` only offers the button when both a code and a help resolver are present. Clicking it calls `HelpLinkResolverService.ForErrorCode`, which probes for the offline, packaged copy of this exact page (`App\Docs\reference\90-error-codes.html`, staged beside `Ring.exe` by `scripts\New-ReleasePackage.ps1`) before falling back to the public website — the same offline-first rule every help link in Ring follows, because the plant LCP often has no route out. See operator/07 — Troubleshooting, "Error codes" for the operator-facing version of this same story.

Database — `DB`

RING-DB-001 — Database could not be opened

Meaning. Ring could not open `RingwoodDatabase.db`, or the database it opened does not have the tables/columns this version of Ring expects (`HealthCheckComposer.ComposeDbSchema`, `ComposeDegraded`).

What to check first.

1. Note whether the **Help** → **System self-test** "Database structure" or "Degraded-mode latch" row is the one that is red — they raise this code for two different reasons (schema mismatch vs. startup open failure).
2. Confirm nobody moved or renamed `RingwoodDatabase.db` next to `Ring.exe`, and that no other process (an open file explorer, a backup tool, an already-running copy of Ring) has the file locked.
3. Check file permissions on the database folder — the account running Ring needs read/write access, not just the logged-in operator.
4. Restart Ring once. If it comes back in degraded mode again, the database could not be opened at all this time either; stop there and escalate — this code does not carry the "don't keep restarting" overwrite risk that RING-DB-002 does, because the file never opened, so there is no live backup rotation to disturb.

Escalate to support, with a support bundle, before starting another batch — a degraded station is read-only for the session and batch history may not be recorded until this is fixed.

Evidence. `HealthCheckComposer.ComposeDbSchema` / `ComposeDegraded` (`Ring/Services/Health/HealthCheckComposer.cs`); operator-facing detail at operator/07#database and operator/07#degraded-mode.

RING-DB-002 — Database integrity check failed

Meaning. `PRAGMA integrity_check` returned something other than `ok` (`HealthCheckComposer.ComposeDbIntegrity`).

What to check first.

1. Stop using the station for new batches immediately.
2. Do **not** restart Ring repeatedly. The file *did* open — it just failed `PRAGMA integrity_check` — so a startup auto-backup (`BackupOnStartup`) runs against the damaged file on every restart, and each backup rotation can push a last known-good backup out of the retention window. One restart is fine; looping restarts to "try again" is what destroys the good backups.
3. Send the support bundle straight away; it carries the failing `PRAGMA integrity_check` detail.

Escalate to support immediately — this is the one self-test row the book treats as a stop-what-you're-doing signal.

Evidence. `HealthCheckComposer.ComposeDbIntegrity` (`Ring/Services/Health/HealthCheckComposer.cs`); operator/07#database.

RING-DB-003 — Database backup failed

Meaning. Either no backup file has ever appeared in the backup folder, or the newest one is older than the warn/fail thresholds (36 hours amber, 7 days red — `HealthCheckComposer.BackupWarnHours` / `BackupFailHours`).

What to check first.

1. Read the backup folder path and file count off the self-test row or the support bundle's "where the evidence lives" block.
2. Confirm the backup folder's drive has free space (a full drive silently stops backups).
3. Confirm nothing (antivirus, a sync client) is locking or moving files out of that folder.

Escalate to support to check the automatic backup schedule if the folder is empty or the newest file keeps falling behind — there is nothing to restore from until this is fixed.

Evidence. `HealthCheckComposer.ComposeBackup` (`Ring/Services/Health/HealthCheckComposer.cs`); operator/07#backups.

RING-DB-004 — Database restore rejected

Meaning. Reserved in the registry for a database-level restore preflight rejecting a candidate file. **Not wired to a call site in this tree** — the station-backup restore path that exists today raises `RING-CFG-011` / `RING-CFG-012` instead (below). Documented here because it is a real, unique code a future restore path can pick up without minting a new one.

Evidence. `RingErrorCodes.cs`; no call site found by search of `Ring/` for `DatabaseRestoreRejected`.

RING-DB-005 — Could not save to the database

Meaning. A screen's save action wrote to the database and the write failed (raised across the setup/data-entry screens — tank roster, Make-Ready tank names, viscometer settings, shift handoff, shift production log, data-entry setup, receiving verification).

What to check first.

1. Retry once — a momentary lock (a backup or export running at the same moment) is the most common cause and clears on its own.
2. Check free disk space on the database drive.
3. Confirm the values you entered are the right type/range for that field (the dialog's sentence usually says which).

Escalate to IT if retries keep failing (permissions on the database folder), or support if the log line under the code looks unfamiliar.

Evidence. e.g. `Ring/Views/Setup/TankRosterScreen.xaml.cs`, `Ring/Views/Shifts/ShiftHandoffScreen.xaml.cs`, `Ring/Views/MainScreen/ReceivingVerificationScreen.xaml.cs` — all call `FriendlyError.Show("save the changes", ex, errorCode: RingErrorCodes.DatabaseSaveFailed)`.

RING-DB-006 — Could not read from the database

Meaning. A report screen's data load from the database failed (raised across the report windows: batch history, batch usage, formula, inventory, usage, data entry, maintenance audit reports).

What to check first.

1. Narrow the filter — an unusually wide date range against a large table is the common trigger.
2. Retry once.
3. Check the self-test "Database structure" and "Database integrity" rows — a read failure here can be the first visible sign of RING-DB-001/002.

Escalate to support with a bundle if the self-test database rows are also red; otherwise IT if it is a single station repeatedly affected.

Evidence. e.g. `Ring/Views/Reports/BatchHistoryReport.xaml.cs`, `Ring/Views/Reports/UsageReport.xaml.cs` — `FriendlyError.Show("load the report data", ex, errorCode: RingErrorCodes.DatabaseReadFailed)`.

PLC — `PLC`

RING-PLC-001 — PLC connection lost

Meaning. The self-test's PLC-link check is red: `PlcConnectionState` reports `Disconnected` (`HealthCheckComposer.ComposePlcLink`).

What to check first.

1. Check the network cable and the switch light at the controller.

2. Use **Setup** → **Start PLC Monitoring**, or the **Retry** button on the red banner. Ring also retries on its own every 30 seconds.
3. Confirm no other tool (a second Ring instance, RSLinx, a laptop plugged straight into the controller) is holding an EtherNet/IP session — the controller only sustains a handful of concurrent sessions per client IP and takes roughly 30 seconds to release exhausted ones (bench-observed).

Escalate to controls/IT if the cable and switch look fine and retries keep failing — that points at the network path or the controller itself, not Ring.

Evidence. `HealthCheckComposer.ComposePlcLink` (`Ring/Services/Health/HealthCheckComposer.cs`); operator/07#plc-connection.

RING-PLC-002 — PLC probe failed

Meaning. Reserved in the registry for a failed one-shot PLC probe (distinct from a lost steady-state connection, RING-PLC-001). **Not wired to a call site in this tree.**

Evidence. `RingErrorCodes.cs`; no call site found by search of `Ring/` for `PlcProbeFailed`.

RING-PLC-003 — PLC clock differs from this PC

Meaning. The controller's clock and this PC's clock disagree by more than `PlcClockSkewMonitor`'s significance threshold (`HealthCheckComposer.ComposePlcClock`).

What to check first.

1. Read the skew and direction off the self-test row (ahead of / behind, in minutes).
2. Use **Setup** → **Update Processor Time/Date** to re-sync the controller clock.
3. If it drifts again soon after syncing, suspect the controller's battery or its own time source rather than a one-off.

Escalate to controls if syncing does not hold — batch timestamps will not line up with the controller until it does.

Evidence. `HealthCheckComposer.ComposePlcClock` (`Ring/Services/Health/HealthCheckComposer.cs`); operator/07#plc-clock.

RING-PLC-004 — PLC read failed

Meaning. Reserved in the registry for a failed individual PLC tag read. **Not wired to a call site in this tree.**

Evidence. `RingErrorCodes.cs`; no call site found by search of `Ring/` for `PlcReadFailed`.

Local file I/O — `IO`

RING-IO-001 — File could not be written

Meaning. Either the database drive is at or below the critical free-space floor, or the log folder is not writable (`HealthCheckComposer.ComposeDiskSpace`, `ComposeLogWritable`).

What to check first.

1. Read the free-space figure off the self-test row — the database, its backups, and every log share one drive.
2. Ask IT to free space now if it is low; do not wait for it to fill.
3. For a log-folder failure specifically, confirm the Ring service account has write access to the log folder.

Escalate to IT — this is a disk/permissions issue, not something a retry inside Ring fixes.

Evidence. `HealthCheckComposer.ComposeDiskSpace` / `ComposeLogWritable` (`Ring/Services/Health/HealthCheckComposer.cs`); operator/07#disk-space, operator/07#logs.

RING-IO-002 — Support bundle could not be built

Meaning. `SupportBundleService` crashed or failed while assembling the one-click bundle.

What to check first.

1. Check free disk space — the same drive the bundle writes to (`%ProgramData%\Ring\bundles\`).
2. Confirm `%ProgramData%\Ring\bundles\` exists and is writable.
3. Retry once from **Help** → **Get a support bundle**.

Escalate to IT if the folder itself is the problem; otherwise read the application log directly (`[RING-IO-002]`) and relay that line to support by phone — the irony of "the bundle failed" is that you may have to describe the failure without one.

Evidence. `Ring/Services/Health/SupportBundleService.cs:102,111`; `Ring/Views/Help/HealthScreen.xaml.cs:212,218`; `Ring/Views/Setup/DiagnosticConsole.xaml.cs:274`.

RING-IO-003 — File could not be read

Meaning. Reserved in the registry for a generic local file read failure. **Not wired to a call site in this tree** (file-specific reads — station backups, database exports — raise their own more specific codes instead).

Evidence. `RingErrorCodes.cs`; no call site found by search of `Ring/` for `FileReadFailed`.

RING-IO-004 — Folder could not be opened

Meaning. Reserved in the registry for a failed "open this folder in Explorer" action. **Not wired to a call site in this tree.**

Evidence. `RingErrorCodes.cs`; no call site found by search of `Ring/` for `FolderOpenFailed`.

RING-IO-010 — Bundled documentation missing (fell back to the website)

Meaning. A help link (F1, a book-glyph, or an **Open help** button) could not find the offline `App\Docs\` copy beside `Ring.exe` and opened the public website instead.

This code is log-only by design — it never raises an operator dialog. Falling through to the website is a working outcome on a PC with internet, and the log line is there so "help opened nothing" on an offline PC has a matching, explainable cause.

What to check first.

1. If the PC has a working route to the internet, there is nothing to fix — the online copy is a complete answer.
2. If it does not, ask IT whether `App\Docs\` was staged during install — packaging can ship without it under an explicit `-AllowMissingDocs` waiver (`engineering/09-build-deploy-release.md`), which is recorded in the release manifest.

Escalate to IT to re-stage the offline docs folder (or re-run `scripts\New-ReleasePackage.ps1` without the waiver) on a station that has no internet route.

Evidence. `Ring/Services/Documentation/HelpLinkResolverService.cs:38,428-441` (`OfflineDocMissingErrorCode`, `LogOfflineMiss`).

Configuration — `CFG`

RING-CFG-001 — Settings could not be saved

Meaning. A configuration write (Quick Setup, or the underlying `appsettings.json` write path) failed.

What to check first.

1. Confirm no other Ring instance or editor has `appsettings.json` open or locked.
2. Check free disk space in the `Config` folder.
3. Retry the save once.

Escalate to IT if the file is on a network share or under restrictive permissions; support if the log line under the code is unfamiliar.

Evidence. `Ring/Views/Setup/QuickSetupScreen.xaml.cs:570-573`; `Ring/Infrastructure/Configuration/ConfigSnapshotService.cs:263` (a snapshot-history miss is explicitly called out there as advisory, not this code — this code is the write itself failing).

RING-CFG-002 — Configuration is invalid

Meaning. Reserved in the registry for a configuration value failing validation before it is applied. **Not wired to a call site in this tree.**

Evidence. `RingErrorCodes.cs`; no call site found by search of `Ring/` for `ConfigInvalid`.

RING-CFG-010 — Station backup could not be built

Meaning. Setup → **Plant Profile** → **Save station backup** could not gather or write the `*.ringstation.zip` payload.

What to check first.

1. Retry once — check the destination drive has free space first.
2. Confirm the destination folder is writable (a read-only USB stick, a locked network share).
3. Check the application log line under `RING-CFG-010` for the specific section that failed to gather.

Escalate to IT for a destination/permissions problem; support if the gather step itself is failing (the payload could not be assembled before a destination was even chosen).

Evidence. `Ring/Views/Setup/PlantProfileScreen.xaml.cs:298,329,567`; `Ring/Services/Setup/StationBackupService.cs:541`.

RING-CFG-011 — Station backup rejected before anything was applied

Meaning. Setup → **Plant Profile** → **Load station backup** read a file that failed preflight — not a `*.ringstation.zip` at all, no manifest, an unreadable zip, or a missing source file. Nothing was applied.

What to check first.

1. Confirm the file is the `*.ringstation.zip` this station (or a compatible one) produced, not a renamed unrelated zip.
2. Re-download or re-copy the file — a partial transfer is a common cause of "not a readable zip archive."
3. Check the rejection message shown in the dialog; it names the specific preflight check that failed.

Escalate to support if a backup you are confident is genuine is still rejected — that is worth reporting, not retrying blindly.

Evidence. `Ring/Views/Setup/PlantProfileScreen.xaml.cs:355-368`; `Ring/Services/Setup/StationBackupService.cs:285-296,613,652` (`StationBackupFormatException`).

RING-CFG-012 — Station backup applied with failures

Meaning. A station-backup restore passed preflight and started applying, but one or more sections failed partway through. The dialog lists which sections; the rest of the restore completed.

What to check first.

1. Read the list of failed sections in the dialog — apply the ones that succeeded's consequences are already in effect, so a partial restore is not a no-op.
2. Compare the failed sections against what actually matters for this station (e.g. tank roster vs. cosmetic settings) before deciding whether to re-try.
3. Retry the restore once; a section that failed due to a momentary lock can succeed on a second pass.

Escalate to support with the failed-section list and the application log line under the code.

Evidence. `Ring/Views/Setup/PlantProfileScreen.xaml.cs:535-537`.

Network / email — `NET`

RING-NET-001 — Email could not be sent

Meaning. Alarm or batch-completion email delivery failed (`HealthCheckComposer.ComposeEmail`, `EmailHealthLevel.Bad`).

What to check first.

1. Check **Setup** → **Email settings** — the account, server, and port.
2. Ask IT whether the mail server or its credentials changed recently.
3. Confirm the plant network has a route to the SMTP server (a firewall change is a common, invisible cause).

Escalate to IT — this is almost always a mail-server-side or network-side change, not something to retry inside Ring.

Evidence. `HealthCheckComposer.ComposeEmail` (`Ring/Services/Health/HealthCheckComposer.cs`); operator/07#email.

UI — `UI`

RING-UI-001 — Unexpected error

Meaning. The catch-all code: an exception Ring did not have a more specific code for, including a startup failure (`App.xaml.cs`) and a crash/hang report counted by the self-test's "Crash reports since start" row.

What to check first.

1. Note exactly what you were doing when it happened — that context is not always in the log line.
2. Restart Ring.
3. Check **Help** → **System self-test**'s crash-report row; if it is not green, something has already failed more than once this session.

Escalate to support with a bundle — this code by itself does not say enough to diagnose from the code alone; the log entry and the bundle are the actual evidence.

Evidence. `Ring/Views/App.xaml.cs:870,874,885`; `Ring/Views/Help/AboutDialog.xaml.cs:78`; `HealthCheckComposer.ComposeCrashes` (`Ring/Services/Health/HealthCheckComposer.cs`); operator/07#crash-reports.

RING-UI-002 — Screen could not be opened

Meaning. Navigating to a screen from the nav bar, or opening a sub-dialog (filter presets, etc.), failed.

What to check first.

1. Retry — a screen that fails to open once from a transient state (mid navigation, mid PLC reconnect) commonly opens fine on a second try.
2. Try navigating to a different screen and back.
3. Restart Ring if it keeps happening on the same screen.

Escalate to support with a bundle if one specific screen consistently fails to open — that is a real defect, not a fluke.

Evidence. `Ring/Views/UserControls/NavBar.xaml.cs:1214,3138,3162,3252,3276,3300`; e.g. `Ring/Views/Reports/BatchHistoryReport.xaml.cs:265` (filter presets).

RING-UI-003 — Printing failed

Meaning. A report's **Print** action failed (raised across the report windows and the alarm legend screen).

What to check first.

1. Confirm a printer is selected and online (paper, power, the usual).
2. Retry.
3. Use **Save as PDF** instead if printing is unreliable in the moment — every report that can print can also save a PDF.

Escalate to IT for a printer/driver problem.

Evidence. e.g. `Ring/Views/Reports/BatchHistoryReport.xaml.cs:117`, `Ring/Views/Help/AlarmLegendScreen.xaml.cs:93` — `FriendlyError.Show("print the report", ex, errorCode: RingErrorCodes.PrintFailed)`.

Export — `EXP`

RING-EXP-001 — Export failed

Meaning. A single-screen export action (inventory report, trending data, the database export form, the plant-profile export) failed. Distinct from the Export Center's own, more specific `EXP-01x` codes below.

What to check first.

1. Check the destination folder is writable and has free space.
2. Retry with a different destination if the first one is on a removable or network drive.
3. Narrow the data being exported (a filter, a shorter date range) if the export is very large.

Escalate to IT for a destination/permissions problem.

Evidence. e.g. `Ring/Views/Reports/InventoryReport.xaml.cs:325`, `Ring/Views/Reports/TrendingScreen.xaml.cs:1182`, `Ring/Views/Setup/DatabaseExportForm.xaml.cs:314,364`, `Ring/Views/Setup/PlantProfileScreen.xaml.cs:221`.

RING-EXP-002 — Import failed

Meaning. Setup → Plant Profile → Import could not read or apply the chosen file (a distinct, older path from the station-backup restore above, which uses `RING-CFG-011`/`012`).

What to check first.

1. Confirm the file is the export/profile type this screen expects.
2. Re-export from the source station if the file may be stale or partial.
3. Check the application log line for the specific section that failed.

Escalate to support if a file you are confident is correct is still rejected.

Evidence. `Ring/Views/Setup/PlantProfileScreen.xaml.cs:273`.

RING-EXP-010 — Export destination missing or not writable

Meaning. The Export Center's chosen (or remembered) destination folder does not exist and could not be created, or is not writable. This is the one Export Center failure that stops the whole run — every other Export-Center failure below continues with the rest of the bundle.

What to check first.

1. Confirm the remembered folder still exists — a USB drive unplugged or a mapped network drive disconnected is the most common cause.
2. Pick a different destination in the Export Center and retry.
3. Confirm you have write permission to the chosen folder.

Escalate to IT for a permissions or mapped-drive problem.

Evidence. `Ring/Services/Export/ExportJobService.cs:53,131-142`.

RING-EXP-011 — One dataset failed during an export

Meaning. One dataset out of the up-to-46 in an Export Center run failed; the run continued and the other datasets were written normally. Recorded on `manifest.json` with the failing dataset's name and error, not raised as a dialog that stops the run.

What to check first.

1. Open `manifest.json` in the export folder and find the failed dataset entry — it names the table and the error.
2. Re-run the export for just that dataset if you need it.
3. Check the self-test "Database" rows if several datasets fail at once — that points at a database-level problem, not an export-specific one.

Escalate to support with the manifest entry if the failure is unexplained.

Evidence. `Ring/Services/Export/ExportJobService.cs:56` (doc comment at line 40: "the job only reports failure when the destination itself was unusable (RING-EXP-010) or the zip step failed (RING-EXP-014)" — a dataset failure alone does not fail the job).

RING-EXP-012 — Database snapshot failed during an export

Meaning. The "include a full database snapshot" extra in an Export Center run failed to copy the database file.

What to check first.

1. Check free disk space at the export destination.
2. Confirm nothing else (a backup, a large query) is holding an exclusive lock on the database at that moment; retry.
3. The rest of the export bundle is unaffected — check `manifest.json` to confirm what did complete.

Escalate to support if it fails repeatedly with free space and no apparent lock contention.

Evidence. `Ring/Services/Export/ExportJobService.cs:59`.

RING-EXP-013 — Scheduled export skipped (destination unreachable)

Meaning. The nightly scheduled export's configured time-of-day came round, but its destination folder was not reachable, so the run was skipped. `export.schedule.lastRunUtc` is only stamped on success, so a skipped night does not falsely mark itself as done — it retries on the next tick within a 12-hour catch-up window.

What to check first.

1. Confirm the scheduled destination (often a mapped network drive) is actually connected and reachable at the scheduled time — a drive that only reconnects after a user logs in is a common cause on a station that reboots overnight.

2. Check the following morning whether the catch-up run appeared instead (the folder timestamp, or the log).
3. Check **Setup** → **Export Center**'s schedule status for the last attempt.

Escalate to IT if the destination is a network path with intermittent availability — the fix is making the path reliably reachable at boot, not inside Ring.

Evidence. `Ring/Services/Export/ExportScheduleService.cs:16-31` (class doc: "the stamp rule"; `ErrScheduledSkipped`).

RING-EXP-014 — Export folder complete but compression failed

Meaning. The Export Center finished writing every file into the export folder, but the final convenience step — zipping that folder — failed. The export itself is not lost: everything is sitting in the uncompressed folder.

What to check first.

1. Go to the export folder directly — it is complete and usable even without the zip.
2. Check free disk space (a zip needs roughly as much extra space as the folder it is compressing, at least transiently).
3. Zip the folder manually if you need a single file to hand off.

Escalate to IT only if it fails on every run (points at disk space or a locked-down zip/compression capability on that PC).

Evidence. `Ring/Services/Export/ExportJobService.cs:62,225-238` (the code comment: "The FOLDER is complete and usable; only the convenience zip failed. Say so rather than letting the operator think the export was lost.").

Reports — `RPT`

RING-RPT-001 — Report could not be generated

Meaning. A report's preview/generation step failed (raised across the report windows: batch history, batch usage, data entry, formula, usage).

What to check first.

1. Narrow the filter or date range and retry.
2. Check the self-test "Database" rows — a generation failure is often a read failure underneath (RING-DB-006).
3. Retry once; a momentary lock clears on its own.

Escalate to support with a bundle if narrowing the filter does not help.

Evidence. e.g. `Ring/Views/Reports/BatchHistoryReport.xaml.cs:447`, `Ring/Views/Reports/UsageReport.xaml.cs:336`.

RING-RPT-002 — Report could not be saved

Meaning. Saving a report as a PDF or CSV failed (raised across nearly every report window and the maintenance audit report).

What to check first.

1. Confirm the destination folder is writable and the file is not already open in another program.
2. Check free disk space.
3. Retry with a different filename or folder.

Escalate to IT for a destination/permissions problem.

Evidence. e.g. `Ring/Views/Reports/BatchHistoryReport.xaml.cs:148,179`, `Ring/Views/Reports/MaintenanceAuditReportWindow.xaml.cs:95`.

Codes are stable API

Once a code is published — shown in a shipped release, or linked to from this page's anchor — **it is never renumbered or repointed at a different meaning.** `HelpLinkResolverService.ForErrorCode` and `HelpLinkHub.ErrorCodeUrl` both resolve on the bare code, and the anchor on each heading above is that code, lowercased, verbatim. A future addition to `RingErrorCodes.cs` gets the next free number in its area; an area that is fully retired is left with gaps, not renumbered.

The online fallback and this page agree by construction: `HelpLinkHub.ErrorCodeUrl` builds `…/docs/reference/90-error-codes.html#<code>` — exactly where this page is published — and the site additionally serves a `/docs/reference/errors/` redirect to the same page as a safety net for older links. (An earlier build of the constant pointed at the redirect path only; that was corrected during integration, so both the offline copy Ring ships and the online fallback now land here.)