ENGINE REFERENCE · STUDIO METAPHOR

A Mastering Studio for Data

Northern Star Systems runs your catalog through a studio-grade signal chain: load the raw take, scan it on the console, process it through the correction rack, and bounce a procurement-ready master — your corrected CSV / XLSX / branded-PDF export.

5 domain presets · HTTP API · branded PDF artifacts · generated 2026-06-15 10:58 UTC

THE SIGNAL CHAIN

RAW TAKE	CONSOLE	RACK	MASTER
Upload	**Validate**	**Correct**	**Bounce = Export**
CSV / XLSX catalog, as recorded	every field gets a channel strip	deterministic fixes, like plugins	CSV · XLSX · branded PDF

Every run is a session: the same take through the same chain produces the same master, every time — with a Run ID on every artifact for the audit trail.

UNDER THE HOOD — THE LITERAL PIPELINE

HTTP API	VALIDATORS	CORRECTORS	EXPORTERS
FastAPI	**Schema + rule packs**	**Worksheet apply**	**csv · xlsx · pdf**
multipart upload, JSON + file responses	per-domain JSON, regex + required	reviewed fixes merged onto the take	pandas / openpyxl / reportlab

Same four stages, no metaphor: a FastAPI service, per-domain JSON schema + rule packs, a worksheet-based correction pass, and pandas / openpyxl / reportlab exporters. No browser, no GPU, no queue, no external calls.

"Branded PDF" means: your company name, URL, accent colors and optional logo on every artifact's letterhead, footer and accents — configured once in branding.json, not hardcoded. Rendering is pure reportlab; there is no headless Chrome to maintain.

STUDIO METAPHOR — WHAT MAPS TO WHAT

Console **Validation engine** Schema + domain rules, per field	**Channel strips** **Field validators** One strip per column	**Meters** **Severity findings** HIGH red · MEDIUM amber · LOW blue
Effects rack **Correctors** Auto-fix worksheet, then apply	**Master bus** **Health score** 0–100 acceptance grade per run	**Presets** **Domain packs** music · healthcare · comms · accounting · inspection

TRANSPORT CONTROLS

CONTROL	STUDIO ACTION	API	STATUS
Record	Load a take (file upload)	`POST /validation/validate`	ACTIVE
Mix	Build the fix worksheet (CSV)	`POST /validation/worksheet`	ACTIVE
Process	Apply the rack — fixes onto the take	`POST /correction/apply`	ACTIVE
Bounce	Export the master (CSV / XLSX / PDF)	`POST /export/file`	ACTIVE
Print	Validation report (branded PDF)	`POST /validation/report`	ACTIVE
A/B	Original vs corrected, side by side	`original + correction columns`	ACTIVE
Loop	Scheduled re-scan + drift detection	`recurring runs`	PLANNED
Punch-in	Re-validate only the rows you touched	`row-scoped runs`	PLANNED

SEVERITY MODEL — ASSIGNMENT, SCORING, ACCEPTANCE

SEVERITY	ASSIGNED WHEN	SCORE IMPACT	ACCEPTANCE EFFECT
HIGH · P1	Required value missing, or a HIGH-severity format rule fails	–6 points each	Blocks acceptance; grade capped at ACTION REQUIRED
MEDIUM · P2	A MEDIUM-severity format or domain rule fails	–1 point each	Fix within the 7-day cycle
LOW · P3	Low-impact or cosmetic rule fails	–1 point each	Tracked; does not gate

Score: start at 100; subtract 6 per Critical finding and 1 per Medium or Low finding; floor at 0. Formula: Score = max(0, 100 – (6 × critical) – (1 × medium) – (1 × low)).
Worked example: 1 Critical + 2 Medium → 100 – 6 – 1 – 1 = **92**, but the open Critical caps the grade at ACTION REQUIRED. Fix the Critical and re-run: 100 – 0 – 1 – 1 = **98** → **HEALTHY**.
Severities come from the preset schema per field — never heuristics — so the same data always grades the same. Cross-reference: the inspection preset grades an unknown condition value (e.g. "utmarkt") HIGH and a malformed date MEDIUM, every run, deterministically.

EXECUTION MODEL

Ordering	**Deterministic per run**
State	**Stateless HTTP · per-run artifacts**
Concurrency	**Parallel runs, isolated Run IDs**
Scaling	**Share-nothing: add processes**

MENTAL MODEL

You get	**A mastering studio for data**
Scale	**Any CSV / XLSX catalog**
Extension	**New domain presets**
Output	**Procurement-ready masters**

PERFORMANCE ENVELOPE & DEPLOYMENT FOOTPRINT (MEASURED)

• Validate a 1,000-row catalog: **~60 ms**. A 100-row take: **~7 ms**.
• Bounce a 100-row branded PDF master: **<100 ms**; CSV / XLSX faster still.
• Production service footprint: **~70 MB RSS** (measured live), cold start under 2 s, one CPU core per request.
• Scaling path: **single worker** → **N uvicorn workers** → **N hosts**, partitioned by Run ID. Runs share nothing, so there is no state to migrate and no coordination layer to add.
• Dependencies: 8 pinned, pure-Python packages (FastAPI, uvicorn, pandas, openpyxl, reportlab, pydantic, numpy, python-multipart). Deploys as one systemd unit behind nginx; containerizes trivially.

EXTENDING THE ENGINE — NEW DOMAIN PRESETS

1. Drop a schema. One JSON file: required columns, regex patterns, severities, messages (schemas/<domain>_schema.json).
2. Add a rule pack (optional). Domain-specific checks beyond format — same JSON convention (rules/<domain>_rules.json).
3. Done. The preset appears across the API, dashboard and every report — zero engine-code changes. The pytest suite covers the validate → correct → export round trip.
Rule-set isolation: presets are self-contained JSON. Adding or editing one cannot change the behavior of any other domain — integration risk stays local.

PRESET VERSIONING & RULESET GOVERNANCE

Versioning. Every schema and rule pack carries a **version** field (year.month.rev, e.g. 2026.06.1) and lives in git. Additive changes bump the revision; breaking changes bump year.month. Every report stamps the preset + schema + rules versions it ran with, so historical artifacts stay reproducible against their exact ruleset.
Adding a rule. A rule change is a pull request: the JSON edit + a seeded demo take that exercises it + the pytest round trip (validate → correct → export). Nothing ships untested.
Deploying. Rules deploy as files with a service restart; rollback is a git revert. No migrations, no downtime windows.
Deprecating. Retired presets stay in git history; artifacts produced under them remain verifiable forever.

DETERMINISTIC REPLAY & ERROR MODEL

Replay. Same take + same preset version = identical findings, worksheet and master. Schemas and rule packs are versioned JSON in git; every artifact carries its Run ID, so any past result can be reproduced and audited.
Post-remediation. Fix the flagged rows and re-run the same file: the score recalculates automatically and cleared findings drop from the report. No manual clearing, no state to reset.
Malformed upload → HTTP 400 with the parse error — never a crash.
Missing required columns → reported as HIGH findings in the run, not an exception.
Unknown domain → falls back to the base schema and still validates structure.
Correction conflicts → HTTP 400 with the reason; the original take is never modified in place.
Telemetry. Every request, latency and error is tracked and visible live on the dashboard.

ARCHITECTURE OVERVIEW (LITERAL)

COMPONENT	WHERE	RESPONSIBILITY
API layer	`api/ — FastAPI`	4 endpoints, multipart in, JSON / file out; telemetry middleware on every request
Services	`services/`	validate · generate_worksheet · apply_corrections · export — thin orchestration over core
Validators	`core/validators/`	schema-driven required + format checks; domain validators add vertical logic (e.g. royalty gaps)
Correctors	`core/correctors/`	merge reviewed worksheet corrections onto the take; original never modified in place
Exporters	`core/exporters/`	csv / xlsx / branded PDF on a shared report kit (pure reportlab)
Presets	`schemas/ + rules/`	versioned JSON, one pair per domain; loaded per request, isolated per domain
Runtime	`systemd + nginx`	one uvicorn unit behind TLS (Let's Encrypt); console behind basic auth

SECURITY & COMPLIANCE POSTURE

• **No external calls** during validation, correction or export — the chain runs entirely in-process.
• **No customer data retained.** Uploads are processed in memory per request; artifacts are returned to the caller, not stored.
• **Telemetry, precisely:** request counts, latency and error rate per endpoint — no row-level data, no PII, held in memory only until process restart. Disable entirely with TELEMETRY_ENABLED=false.
• **Data residency by construction.** The engine runs entirely on your infrastructure; nothing leaves the host. GDPR-friendly: no PII storage means no retention schedule to manage.
• **Transport & access.** TLS via nginx + Let's Encrypt; the console and API sit behind HTTP basic auth; the marketing surface is the only public path.
• **Auditability.** Run ID + preset/schema/rules versions on every artifact — any result can be traced to the exact code and rules that produced it.
• **Supply chain.** 8 pinned, widely-audited pure-Python dependencies; no browser, no GPU, no queue, no database.

WHY THIS IS HARD TO BUILD

• **The artifact chain is the product.** Validation is easy; a deterministic, procurement-grade evidence chain (report → worksheet → corrected master, all version-stamped and replayable) is what compliance teams actually accept — and what takes the engineering discipline.
• **The rule packs are distilled domain knowledge.** Five presets encode the working vocabulary of five live vertical products: ISRC/ISWC/IPI and §507(b) statute logic (music), CARC/RARC/NPI/CPT (healthcare claims), CIP channel findings (comms), SIE/BAS double-entry conventions (Swedish accounting), besiktningsprotokoll condition scales (inspection). That's operator knowledge, not just code.
• **Print-grade branded PDFs without a browser.** The report kit renders score rings, gauges and finding cards in pure reportlab — no headless Chrome fleet to babysit, which is what makes per-request rendering on a 70 MB service possible.
• **Determinism is a constraint, not a feature flag.** No model calls, no clocks in the logic, no hidden state — kept honest across every preset by the test suite.

INTEGRATION PLAN (30–60 DAYS)

Days 1–7 — Deploy & smoke. Stand up the unit (VPS or container), run the bundled demo kit: all five presets through the full chain in one command, artifacts compared against known-good output.
Days 8–21 — Map your data. Point your catalog exports at an existing preset, or author your own schema + rule pack (JSON only). Pilot on real files; tune severities with your compliance owner.
Days 22–45 — Wire the pipeline. Integrate the four endpoints into your intake flow; stand up the operator worksheet loop for corrections that need human review.
Days 46–60 — Gate & monitor. Make the score an acceptance gate in your procurement/release workflow; schedule re-scans; watch the dashboard telemetry.
Typical effort: one engineer, part-time. There is no data migration — the engine is stateless by design.

WHY TEAMS BUY IT

- **Deterministic.** Same take in, same master out — no model drift, no surprises in re-runs.
- **Evidence on every run.** Validation report, fix worksheet, corrected master and audit trail — each carrying the Run ID.
- **Preset-extensible.** A new vertical is a new schema + rule pack, not a rewrite — five presets shipped, more on request.
- **Light footprint, real scale.** Pure Python, stateless — one VPS today, horizontal scale by adding processes, because runs share nothing.