x_make_github_visitor_x â€” Auditor's Field Manual

I built this visitor to walk every clone, interrogate it with Ruff, Black, MyPy, and Pyright, and file the evidence before the orchestrator moves a muscle. Every run produces JSON ledgers the GUI and change board trust. No repo graduates without the stamp from this inspector.

Mission Log

- Walk every child repository discovered under the designated workspace root.
- Cache prior results so repeat offenders do not waste the lab's time.
- Emit per-tool diagnostics and aggregate summaries the orchestrator ingests without post-processing.
- Capture runtime, environment, and failure detail in a report the Change Control index can audit months later.

Instrumentation

- Python 3.11 or newer.
- Ruff, Black, MyPy, and Pyright installed in the active environment.
- Optional GitHub token if you expect to touch private clones.

Operating Procedure

1. `python -m venv .venv`
2. `\.venv\Scripts\Activate.ps1`
3. `python -m pip install --upgrade pip`
4. `pip install -r requirements.txt`
5. `python -m x_make_github_visitor_x.runner` (add acceleration flags as needed)

Acceleration / Fast Paths

For rapid iterative inspection cycles you can engage selective fast paths via CLI flags (they set corresponding `VISITOR_*` environment variables and are recorded for traceability in the runtime snapshot and summary):

Flag	Effect	Env Var	Traceability Key
`--quick`	Skips all tool execution (files discovered only)	`VISITOR_QUICK_MODE`	`quick_mode`
`--skip-content-hash`	Bypasses per-file hashing (uses path-derived placeholder)	`VISITOR_SKIP_CONTENT_HASH`	`skip_content_hash`
`--skip-tools`	Runs only a lightweight Pyright error-level scan	`VISITOR_SKIP_TOOLS`	`skip_tools`

Example (PowerShell):

```
python -m x_make_github_visitor_x.runner --quick --skip-content-hash --skip-tools
```

The active fast paths appear in generated JSON under `runtime.fast_paths` and in the summary as `fast_paths_active`.

The runner discovers immediate child clones, executes the tool suite, and drops a timestamped JSON dossier under `reports/`. The orchestrator and GUI expect those artefacts in that location.

Streaming Mode (Incremental Events)

In streaming mode the visitor emits **line-delimited JSON (NDJSON)** events while parsing instead of waiting to produce one monolithic end-of-run report. This accelerates GUI feedback and enables real-time orchestration decisions.

Event Type	Emitted When	Schema	Purpose
`file`	After each file analysis	`reports/schemas/file_event.schema.json`	Surfaces per-file issues early
`repo`	After finishing a repository scan	`reports/schemas/repo_event.schema.json`	Summarizes repo-level status and counts
`rotate`	When the NDJSON stream file exceeds size threshold	(implicit control event)	Signals tailer to reopen next segment

Stream path: `reports/events/visitor_events.ndjson` (single writer, fsync per line). Per-repo and per-file shards may also be mirrored under `reports/visitor_streams/` for recovery.

Enable streaming: set `VISITOR_STREAMING=1` (or future CLI flag `--stream`). Disable with `VISITOR_STREAMING=0` to revert legacy single-summary behavior.

Each event envelope includes: `{ "schema": "x_make_github_visitor_x.event/1.0", "emitted_at": "2025-11-08T19:12:34Z", "run_id": "<uuid>", "event": "file|repo|rotate", "repo": {"id": "x_make_common_x", "display": "x_make_common_x"}, "payload": { ... } }`

The orchestrator tailer validates each event with `x_make_common_x.json_contracts.validate_payload` and upserts progress indices for

immediate GUI refresh. Final summary JSON still persists for archival and hash reconciliation.

Color semantics downstream: `completed` = green, `attention` = yellow (issues but finished), `blocked` = red (hard failure). Only `completed` marks a stage fully done.

Advantages: lower perceived latency, early triage, resilience (partial progress survives crashes), future astrocyte network readiness.

Fallback: If file system permissions prevent stream creation the visitor automatically reverts to legacy mode and logs a single summary report.

Evidence Checks

| Check | Command | | --- | --- | | Formatting sweep | `python -m black .` | | Lint interrogation | `python -m ruff check .` | | Type audit | `python -m mypy .` | | Static contract scan | `python -m pyright` | | Functional verification | `pytest` |

Reconstitution Drill

During the monthly rebuild I drop this visitor onto a sterile machine, rerun the sweep, and compare the JSON fingerprints to prior runs. Cache hits must still line up, failure payloads must stay richly annotated, and the orchestrator summary must hydrate from the new dossier without a hiccup. Any drift becomes Change Control fodder and is reconciled before the next inspection cycle.

Conduct Code

Every failure becomes a record. If a tool coughs, log it, store the JSON, and elevate through Change Control. No quiet fixes; the lab's memory lives in these dossiers.

Sole Architect's Note

I designed, built, and hardened this inspector alone. Tool orchestration, cache semantics, schema validation, and downstream integration are mine end to end. I speak the language of automation strategy and low-level Python, so the plan and the implementation never diverge.

Legacy Staffing Estimate

- Without LLM acceleration, you would mobilize: 1 senior Python lead, 1 DevOps engineer, 1 QA automation specialist, and 1 technical writer.
- Delivery timeline: 12â€“14 engineer-weeks to build the visitor, validation harness, and documentation to this standard.
- Budget bands: USD 85kâ€“110k for the initial campaign, not counting the institutional knowledge already condensed here.

Technical Footprint

- Language Core: Python 3.11+, standard library concurrency, and deterministic filesystem sweeps.
- Toolchain: Ruff, Black, MyPy, Pyright, pytest, and PowerShell entry points for Windows operators.
- Reports: JSON dossiers stored under `reports/`, schema validation enforced by `x_make_common_x`.
- Integration Surface: CLI entry point (`python -m x_make_github_visitor_x.runner`) with orchestrator hooks and structured logging routed through `x_make_common_x`.