OSF — Methodological Framework (B)

transobserver — structured metrics collector (engines + schema + tools)

0) Identity / scope

- **Recommended OSF title**: *transobserver — metrics collection/structuring (engines + schema)*
- **Source repository (URL)**: `github.com/dalozedidier-dot/transobserver`
- **Resource type**: *Software*
- **Observed status**: public repository, 27 commits, detected languages (Python / HTML / Shell / Jupyter).
- **Version to freeze (Registration snapshot)**:
  - Branch: `main`
  - Tag: […]
  - Commit hash: […]
  - Freeze date: […]

1) Definition (Description / Abstract)

transobserver is a software layer for **collecting, structuring, and preparing** metric data. Its architecture is organized around dedicated directories for configuration, **engines** (collectors), **schema** (structural contract), tools, examples, fixtures, and test data, with Python packaging via `pyproject.toml`. The expected product is a set of **structured, versionable metrics** intended for auditability and downstream use (with **no interpretation embedded** in the collector).

2) Materials / components to archive (Materials)

Include in OSF (copy/archive):

- **Directories**:
  - `config/`, `engines/`, `schema/`, `tools/`, `examples/`, `fixtures/prepared_bands/`, `test_data/`, `transobserver/`
- **Files**:
  - `pyproject.toml`, `QUICKSTART.txt`, `README.fr.md`, `TEST_DATA_LINKS.md`
- **Frozen archive**: ZIP of the repository at the recorded commit/tag (OSF “Files”).

3) Data (Data / Inputs)

- **Sources** (must be declared; not inferable from UI listing alone):
  - source types (logs, exports, APIs, local files, etc.)
  - access mode (pull/push), prerequisites (tokens not stored), granularity
- **Contract**:
  - `schema/` is the central artifact: it defines the expected structures.
- **External references**:
  - if `TEST_DATA_LINKS.md` points to resources outside the repository, OSF should include either a snapshot or a versioned reference (URL + date + hash if available).

4) Operational procedure (Methods)

Minimum procedure to describe (operational, without implicit assumptions):

1. Select a configuration under `config/`.
2. Run an engine under `engines/` against a declared source.
3. Produce structured metric artifacts compliant with `schema/`.
4. (If implemented) validate conformance and generate fixtures in `fixtures/` / `test_data/`.
5. Archive outputs and execution logs.

Exact command(s): to be filled in OSF from `QUICKSTART.txt` / scripts (not reliably extractable here without executable-level inspection).

5) Outputs / products (Outputs)

Freeze in OSF (at least one complete example):

- 1 “structured raw metrics” output compliant with the schema (format to specify: CSV/JSON/NDJSON…)
- 1 `fixtures/prepared_bands` output (if these “bands” are an internal operating standard)
- 1 minimal `test_data/` set for non-regression
- Versioned schema (`schema/`) included.

6) Validation / verifiability criteria (QA / Validation)

Minimum validation:

- outputs conform to `schema/`
- reproducibility on `test_data/` (stable diffs / snapshots)

If CI exists under `.github/workflows/`: include a description of the checks and success criteria.

7) Packaging / environment (Reproducibility)

- `pyproject.toml` present (packaging).
- Freeze in OSF:
  - target Python version
  - effective dependencies (lock/requirements or export)
  - OS/runner if determinant

8) License / reuse (License)

- **OSF License field**: must be explicitly filled in (even if the repository has no LICENSE file).

9) Limitations (Limitations)

- Without hash/tag + archive, the software artifact is not fixed.
- Without a conforming output example + included schema, the tool is not auditable by third parties.
- transobserver does not “conclude”: it structures/collects; it does not “decide”.

OSF note (OSF project structure + registration)

An OSF **Registration** is an immutable snapshot; it is the appropriate mechanism to freeze a software state plus artifacts. For a “software” registration, the non-compensable minimum is: **frozen archive + hash + license + procedure + input/output examples**.