SystemD — v0.3.1

OSF Home Document (Project Overview)

Document version: OSF_HOME_v0.3.1.0
Date: 24 January 2026 (Europe/Brussels)

---

1) OSF Metadata

OSF Title (root project)
SystemD — v0.3.1 — Core + Multisector Tests + Sectors

OSF Description (short field)
Root OSF project for SystemD v0.3.1: a descriptive DD-R core runner (optional Equilibrium E) applied to an observation matrix. Includes multisector tests (“profiles-as-contract”) with SHA256 non-regression. Sector specialization is organized as components `SXX_<sector>`.

---

2) Purpose

This OSF project centralizes the **v0.3.1** baseline of SystemD and its structured split into sectors.
The canonical entry point is v0.3.1: sectors are **structural specializations**, and the shared core is not modified outside explicitly traced procedures (see traceability).

This repository is designed to:

- maintain a **single entry point** (v0.3.1);
- enforce a strict separation between **core** / **cross-sector tests** / **sector-specific content**;
- enable **freezing** (“Registration”) of a complete state when required.

---

3) Canonical OSF Structure

Recommended OSF organization: **one root project** plus **components**.

Level-1 components:

- **00_core** — shared baseline (specs, conventions, templates, runner, reference docs)
- **01_tests_multisector** — cross-sector test suite (profiles, fixtures, non-regression, integrity index)
- **SXX_<sector>** — one component per sector (data, local params, adapters, sector outputs)
- **99_releases** — frozen exports (zips, checksums, changelog, snapshots)

Non-compensable split rules:

- Anything **shared** must live in **00_core** (otherwise duplication → divergence).
- Anything **cross-sector** must live in **01_tests_multisector**.
- Anything **sector-specific** must live in **SXX_<sector>**.

---

4) Sector Component Layout

Each sector follows the same minimal layout (same structural slots).

Minimum recommended layout for `SXX_<sector>`:

- `SXX_<sector>/inputs/` — sector sources (files, matrices, references)
- `SXX_<sector>/params/` — local parameters (without editing the core)
- `SXX_<sector>/adapters/` — compatibility bridges (formats → core conventions)
- `SXX_<sector>/outputs/` — sectorized outputs (JSON reports, indexes, logs)
- `SXX_<sector>/notes/` — descriptive notes: explicit assumptions, limits, structuring decisions

Naming convention: `S01`, `S02`, … (creation order). The `<sector>` label is **stable** (no opportunistic renaming).

---

5) Conventions (Determinism and Computation Contracts)

The conventions below are **contracts** (descriptive, not interpretive).

- **Determinism:** same inputs → same outputs (format, ordering, indexing).
- **Proxy series:** sequential IDs 1..n when observations are not timestamped.
- **Robust statistics:** median and MAD when relevant; tail structure via p90/p99 quantiles.
- **Relative divergence:** `div_rel = |post - pre| / |pre|`, base = pre (if base ≠ 0; otherwise explicit rule applies).
- **Computability guardrails:** moment-based metrics (variance/std/entropy) are neutralized when n is insufficient; neutralizations are logged.
- **Outputs:** structured JSON reports + integrity indexes (hashes) for releases.

---

6) Traceability, Freeze, and Publication (OSF)

Two states coexist:

- **Editable project** (work state)
- **Registration** (immutable snapshot)

Freezing is performed at the v0.3.1 **root level** to capture the full assembly (core + tests + sectors).

Freeze rules:

- One registration = one complete state: **root + components**.
- Registration naming: **v0.3.1-r1**, then **v0.3.1-r2** if re-frozen, etc.
- Release bundles (zips) are stored in **99_releases**; registrations also freeze those artifacts.

---

7) Tools (OSF “Tools” / Internal Listing)

These entries can be pasted into OSF tool descriptions.

DD-R Core Runner (core)
Executes the descriptive pipeline: reads an observation matrix, performs structured extraction, computes invariants, measures relative divergence, and generates deterministic JSON reports.

Executable Specification
Versioned configuration defining thresholds, computability rules, and conventions (including neutralization rules and output structure).

Observation Matrix Template
Input structure template to guarantee stable extraction and cross-sector comparability.

Multisector Test Harness
“Profiles-as-contract” execution: runs profile YAMLs, aggregates results, and enforces non-regression via hashes.

Integrity Index
File inventory with cryptographic hashes (SHA256) to attest non-alteration of releases/snapshots.

---

8) Minimal Checklists

Checklist — creating a new sector:

- Create component **SXX_<sector>** (stable name).
- Create the minimal structure: `inputs/ params/ adapters/ outputs/ notes/`.
- In `notes/`, document: sources, explicit assumptions, limits, local conventions.
- Add at least one sector test profile (or link an existing multisector profile).

Checklist — before a release (99_releases):

- Generate an export bundle (core + tests + relevant sectors).
- Generate a SHA256 integrity index and store it with the bundle.
- Update `CHANGELOG.md` (if present) with **structural changes only**.

- Optionally create an OSF registration (root snapshot).

---

9) Status / Explicit Limits

- This document describes structure and contracts; it does not interpret results.
- License is not specified (to be defined based on the intended openness: external publication vs internal use).
- Sector components may require access controls; permissions can be managed per component.

---

10) Contact / Maintenance

- **Maintainer:** D.D Conscience (@DDGraphisme)
- **Channel:** to be specified (email / repo / internal channel)
- **Support convention:** issues logged in `notes/` (sector) or `docs/` (core), with version ID and minimal reproducibility data.