perplexity

A neural-variant of eXpFS for augmented citizens can treat “files” as evolving neuro-bioscale artifacts, while preserving the simple, single-root abstraction of eXpFS and its XFS-style external interface.

Yes, you can treat “files” as evolving neuro-bioscale artifacts while preserving a single-root, XFS-style eXpFS interface, by inserting a neurorights-aware, ALN-backed metadata and evolution layer beneath the familiar pathname abstraction.[1]

Core Idea: Neuro-bioscale eXpFS veneer

Keep `/` and the XFS-like POSIX surface exactly as eXpFS defines it, but reinterpret each “file” as a **SovereignArtifact** whose identity and lifecycle are anchored in your ALN and Bostrom stack rather than just inode + blocks.[1]

```
Each path `/<subject>/<space>/<name>` still resolves through a VFS layer, but the backing
```

- A content shard: bytes on disk or in NeuroPC-attached storage (neural traces, logs, configs).[1]
- An ALN policy shard: `.neurorights.json`, `.rohmodel.aln`, `.ocpuenv`, `.lifeforce.aln`, `.tsafe.aln` describing neurorights, RoH bounds, and bioscale envelopes for this artifact.[1]
- An evolution ledger: `.evolve.jsonl` / `.donutloop.aln` entries that record proposals and applied changes to this artifact’s semantics, bindings, or allowed routes.[1]
- A blockchain proof envelope: `.bchainproof.json` anchoring its identity to Googolswarm / Organicchain and specific Bostrom addresses (yours and safe alternates).[1]

From the outside you still “open a file,” but inside you’re always dealing with a time-indexed, RoH-bounded, neurorights-constrained object.

Layout: Single root, rich shards under the hood

One concrete way to do this without breaking the simple tree abstraction:

- Root stays `/`.
- Each augmented citizen (subject) gets a root namespace keyed by Bostrom address, e.g.:
  - `/bostrom18sd2u/...` (primary)

    - `/bostrom1ldgmtf...` (secure, monitored)
    - `/zeta12x0up66...` (lab/safe alternate)
    - `/0x519fC0eB41...` (on-chain anchored artifacts).[1]
- Under each subject root, you mount *views* (workspaces) that look like ordinary directories but are backed by ALN + evolution state:
    - `/bostrom18sd2u/neuro/` → `.neuroaln`, `.nstream.neuroaln`, `.lifeforce.aln`, `.biosession.aln` shards with Tsafe envelopes.[1]
    - `/bostrom18sd2u/kernels/` → sovereign kernel artifacts: `.evolve.jsonl`, `.donutloop.aln`, `.nnet-evolve.jsonl`, `.rohmodel.aln`, `.stake.aln`, `.vkernel.aln`.[1]
    - `/bostrom18sd2u/rights/` → `.neurorights.json`, `.nnet-rights.json`, neurorights hypervector templates.[1]
    - `/bostrom18sd2u/proofs/` → `.bchainproof.json`, `.nnet-proof.bchain.json`.[1]

The mountpoint is static and simple; the richness comes from how the backend driver interprets each path's ALN bundle and evolution stream.[1]

Artifact model: "File" as evolving neuro object

Define a SovereignArtifact record stored in ALN and indexed by path.[1]

Essential fields:

- `subject_id`: Bostrom address / OrganicCPU subject.[1]
- `kind`: e.g. `NeuralShard`, `NeuroRightsPolicy`, `EvolveStream`, `DonutLedger`, `BChainProof`, `NNetModel`.[1]
- `routes`: allowed access routes (`BCI`, `OTA`, `GOV`, `CHAT`, `RESEARCH`).[1]
- `roh_before`, `roh_after`: last-evaluated risk-of-harm slice for the artifact.[1]
- `neuro_state_link`: pointers into `.neuroaln`, `.nstream.neuroaln`, `.lifeforce.aln`, `.biosession.aln` where this artifact participates in your 5D biophysical identity coordinates (BioState, NeuroState, Lifeforce, Context, Sovereignty).[1]
- `rights_profile`: binding to `.neurorights.json` and `.nnet-rights.json`, including flags like `mentalprivacy`, `dreamstatesensitive`, `soulnontradeable`, `forbiddecisionuse`.[1]
- `evolve_ref`: link to entries in `.evolve.jsonl` / `.nnet-evolve.jsonl` and `.donutloop.aln` that describe structural changes.[1]
- `proof_ref`: associated `.bchainproof.json` anchoring identity to Googolswarm / Organicchain.[1]

From a POSIX call's perspective, you still do `read()` or `write()` on a file descriptor; the driver consults this artifact record before deciding what the call actually means and whether it is allowed.[1]

Policy engine: neurorights + Tsafe gating at the FS boundary

To preserve neurorights and bioscale integrity while keeping UX simple, you can reuse the Tsafe Cortex Gate pattern and attach it directly to the filesystem driver.[1]

High-level rules:

- Mental privacy guard
  - Raw `.neuroaln` / `.nstream.neuroaln` never leave the host or cross the LLM boundary via ordinary file reads; the FS driver exposes only derived, aggregated, or redacted views when the caller is an AI-mediated route.[1]
- Key and proof guard
  - `.bchainproof.json`, key shards, and sovereign-kernel manifests are readable only as typed, verified statements (e.g., "proof OK, policy set X active"), never as raw blobs, when accessed through high-level tools; low-level binary read requires explicit, non-AI route with local consent.[1]
- Lifeforce and bioscale constraints
  - Each artifact has lifeforce cost and QPU/eco impact metadata from `.ocpuenv`, `.lifeforce.aln`, `.OrganicCpuRuntimeMetrics…` shards; the FS driver denies or throttles operations when envelopes are exceeded.[1]
- Route and role constraints
  - Access decisions incorporate `.stake.aln` roles (Host, OrganicCPU, ResearchAgent) and SMARTEVOLVE scopes, so a governance process can examine `.evolve.jsonl` as text, but a generic tool cannot write to it without passing RoH and neurorights checks.[1]

This policy engine sits between VFS operations and actual block IO, so the single-root abstraction is preserved while semantics become neurorights-aware and RoH-bounded.[1]

Evolution: how a "file" changes over time

In a neural eXpFS, evolution is first-class and append-only rather than hidden mutation.[1]

For each SovereignArtifact:

- Changes to its content, neurorights posture, access routes, or bindings are represented as entries in `.evolve.jsonl` / `.nnet-evolve.jsonl` with `proposal_id`, `decision`, `roh_before`, `roh_after`, `effect_bounds`, `tokenkind`, `signatures`, `hexstamp`.[1]
- Accepted changes are mirrored into `.donutloop.aln` (hash-linked) and anchored into `.bchainproof.json` as needed, creating an immutable audit trail of how the "file" evolved as a neuro-bioscale object.[1]
- The live view you see at `/subject/space/name` is simply the current projection of that evolution stream under the constraints of `.rohmodel.aln` and `.tsafe.aln`.[1]

This matches your need for ALN/KYC/DID-compliant, quantum-ready governance while keeping the developer experience close to a regular XFS-style filesystem.[1]

Rust implementation direction (eXpFS neural variant)

Within Rust-Party, you can treat this as a new backend for the eXpFS VFS:

- A Rust crate for the **neuro-eXpFS driver** that:
  - Maps paths to SovereignArtifact descriptors and ALN shards.[1]
  - Calls a Tsafe Cortex Gate-style `PolicyEngine` on each open/read/write/ioctl.[1]
  - Emits evolution records to `.evolve.jsonl` / `.donutloop.aln` instead of performing opaque in-place mutations.[1]
- A `neuro-workspace.manifest.aln` at the root of each subject's namespace tying together: `.rohmodel.aln`, `.stake.aln`, `.neurorights.json`, `.smart.json`, `.evolve.jsonl`, `.donutloop.aln`, `.ocpu`, `.ocpuenv`, QPU shards, guard pipeline.[1]

The result is a neural-variant eXpFS that:

- Looks like a single-root, XFS-like filesystem to tools and legacy apps.
- Treats every "file" as a living neuro-bioscale artifact subjected to neurorights, RoH, lifeforce envelopes, and on-chain sovereignty proofs.
- Keeps evolution, policy, and proofs in explicit ALN shards, not hidden behind the POSIX veneer.[1]

⁂

Feature associations and governance parameters can be restructured into a neuromorph/OrganicCPU spec-sheet by treating classic eXpFS concepts as a base layer and layering neurorights, SMART tuning, and evolution tokens on top.exposnitc.github+1

Core layout: neuromorph file system spec
DimensionClassic eXpFS (XSM)Neuromorph / OrganicCPU variant (2026)
Disk model
Fixed-size blocks; each block holds multiple words.[exposnitc.github]
Fixed-size blocks mapped to shard types: neuro streams, biospec envelopes, ledgers, model snapshots.
Logical namespace
Single root file; flat logical hierarchy.exposnitc.github+1
Single sovereign root plus typed shards (.aln, .json, .neuroaln, .nnetx) under one manifest.
Internal structures
Root table, Disk Free List, optional inodes.exposnitc.github+1
Root table + shard indices, RoH model (.rohmodel.aln), neurorights (.neurorights.json), donut ledgers.
Executable format

XEXE: 8-word header + code section.exposnitc.github+1
XEXE for OS-level code; .nnetx/.nnetq specs for neural kernels; evolution headers in .nnet-evolve.jsonl.
Access mechanism
System calls only: Create, Read, Write, Seek, etc.[exposnitc.github]
System-call style API: FsHandle + neurorights checks + SMART guards before block access.
Permission model
owner + permission (0=exclusive, 1=open).exposnitc.github+1
owner + neurorights + SMART scope + EVOLVE tokens + OrganicCPU envelopes governing fatigue and RoH.
External interface
XFS: fdisk, load, df, ls, export, dump, rm.exposnitc.github+1
NeuroXFS: same verbs plus anchor-to-blockchain, verify-proof, export-with-neurorights-filter.
File-type classes and shard associations
Base file attributes (extended)
name: canonical shard name (e.g., subjectA.neuroaln, bostrom-sovereign-kernel-001.ndjson).
size_words: logical word count or shard units (for streams, “frames”).
type: ROOT, DATA, EXEC, NEURO_STREAM, BIOSPEC, LEDGER, MODEL.
owner: subject or sovereign kernel ID (e.g., OrganicCPU DID).
permission: exclusive/open/shared-read/shared-write, similar to eXpFS multiuser but extended. exposnitc.github+1
neurorights: mental_privacy, mental_integrity, cognitive_liberty, noncommercial_neural_data, soulnontradeable, dreamstate_sensitive, forbiddecisionuse, forget_sla_hours.
governance: SMART policy ID, EVOLVE token requirement, RoH ceiling for this file.
File-type association table
File type / classExample extensionsPurposeAutonomy / SMART parameters
ROOT
root, neuro-workspace.manifest.aln
Canonical index of all shards, paths, RoH model, stake, neurorights pointers.
Only EVOLVE tokens can modify; SMART cannot override RoH caps.
DATA
.data, .json, .aln
Generic configs, logs, non-neural datasets.
SMART may tune small hyperparams, never override neurorights.
EXEC
.xexe
XEXE OS code: loader, shell, handlers.exposnitc.github+1
Immutable except via OS upgrade path; tracked in ledger.
NEURO_STREAM
.neuroaln, .nstream.neuroaln, .lifaln
High-rate neural and lifeforce microshards.
BioLoadThrottle limits IO; SMART can adjust sampling rate.
BIOSPEC
.biospec.aln, .ocpu, .ocpuenv, .ocpulog, .biosession.aln, .lifeforce.aln
OrganicCPU profile, fatigue envelopes, runtime metrics.
SMART can adjust duty cycles within Tsafe; EVOLVE needed for new envelopes.

LEDGER
.donutloop.aln, .evolve.jsonl, .nnet-loop.aln, .answer.ndjson, .bchainproof.json
Hash-linked evolution and decision log with blockchain anchors.
Append-only; SMART cannot delete, only add reversible entries.
MODEL
.nnetx, .nnetw, .nnetq, .nfeat.aln, .nmap.aln, .nnet-policy.aln, .nnet-cap.aln, .nnet-rights.json
Neural models, feature specs, capabilities, per-model neurorights.
SMART can retune (learning rate, thresholds); EVOLVE needed for architecture changes.
SOVEREIGN_CONFIG
.rohmodel.aln, .stake.aln, .neurorights.json, .smart.json, .evolve-token.json, .tsafe.aln, .vkernel.aln
Risk-of-Harm model, governance rules, neurorights, token policies, viability kernels.
Only EVOLVE path with multi-sig approval; SMART cannot touch.
This keeps eXpFS’ simple type triad but logically extends it into classes used by the neuromorph runtime.
Governance and SMART autonomy parameters
SMART tuning layer
SMART policies govern small, reversible, day-to-day adjustments:
maxeffectsizel2: maximum allowed parameter delta per SMART adjustment.
domains: which domains can be tuned (e.g., recommendation weights but not neurorights flags).
expiry: time-bounded autonomy; reauthorization required after expiry.
physioguard: bound tuning by .ocpuenv fatigue and .lifeforce.aln envelopes.
revocable: changes must be logged and reversible via Donutloop and stake rules.
Example (conceptual) SMART parameters for a NEURO_STREAM file:
max_sampling_hz: can increase stream sampling up to a ceiling if fatigue index < 0.4.
max_window_minutes: moving window of history; SMART can shrink but not expand beyond neurorights storage scope.
max_parallel_readers: number of processes that may subscribe to the stream; constrained by mental_privacy and stake roles.
EVOLVE token layer
EVOLVE tokens govern deep, structural evolution:
scope: which shard paths can be modified (e.g., .rohmodel.aln, .stake.aln).
roh_before, roh_after: expected RoH bounds before and after evolution.
decision: approved/denied state linked to .donutloop.aln entry.
tokenkind: type of evolution (QPolicyUpdate, BioScaleUpgrade, KernelChange).
signatures: multi-sig attestations from stakeholders (Host, OrganicCPU, ResearchAgent).
Any write to SOVEREIGN_CONFIG or MODEL architecture files must:
Verify a valid EVOLVE token.
Create a new ledger entry in .donutloop.aln and, for models, .nnet-loop.aln.
Anchor the change through .bchainproof.json or .nnet-proof.bchain.json.
Named neural protections baked into the spec
Each operation (Create, Read, Write, Delete, Export) passes through a set of named protections:
AuraBoundaryGuard
Enforces that NEURO_STREAM and BIOSPEC files cannot be accessed by other subjects or remote nodes unless neurorights allow it.
Ties owner, mental_privacy, and stake roles to access decisions.
SoulNonTradeableShield

Prevents export, tokenization, or IPFS-style pinning of files with soulnontradeable = true.
Disallows binding such files into commercial SMART tokens or marketplaces.
DreamSanctumFilter
Applies when dreamstate_sensitive = true and forbiddecisionuse = true.
Forbids using dream-derived streams as features in decision-making models; only allows local, introspective visualizations or subject-controlled research.
BioLoadThrottle
Reads .ocpuenv, .biosession.aln, and .lifeforce.aln to throttle IO, learning, and inference when fatigue or lifeforce load exceed thresholds.
Applies to NEURO_STREAM, BIOSPEC, MODEL training logs.
SovereignKernelLock
Protects .rohmodel.aln, .stake.aln, .neurorights.json, .smart.json, .evolve-token.json, .tsafe.aln, .vkernel.aln, and root manifest.
Only EVOLVE-with-ledger path may change them, never direct writes or ad-hoc scripts.
These protections form the neuromorphic analogue of eXpFS multiuser owner/permission flags, but extended to neurorights, biophysical load, and token-governed autonomy.
exposnitc.github+1
External interface: NeuroXFS (SMART-governed)
Refactoring the XFS command set into a neuromorph interface:
Classic XFS commandNeuromorph / OrganicCPU equivalent and constraints
fdisk (format)
neuroxfs format initializes root, free list, RoH model, neurorights, stake.
load --data file.dat
neuroxfs load --data shard.aln applies AuraBoundaryGuard + SoulNonTradeableShield; tags neurorights.
load --exec prog.xexe
neuroxfs load --exec prog.xexe only into EXEC zone; logged in ledger.exposnitc.github+1
ls
neuroxfs ls shows file class, neurorights flags, SMART scopes per file.
df
neuroxfs df includes neuro-IO and fatigue quotas, not only bytes.
cat file.dat
neuroxfs cat shard.aln checks neurorights and DreamSanctumFilter first.
export file.dat ./out
neuroxfs export shard.aln ./out refuses if soulnontradeable or mental_privacy disallow.
dump --rootfile
neuroxfs dump --manifest exports neuro-workspace.manifest.aln plus SOVEREIGN_CONFIG summaries.
rm file.dat
neuroxfs rm shard enforces forget_sla_hours and ledger logging, not silent deletion.
This spec-sheet preserves the pedagogical clarity of eXpFS while enabling neuromorph and OrganicCPU autonomy via neurorights, SMART governance, and EVOLVE tokens anchored in biophysical and blockchain proofs.exposnitc.github+2

The spec-sheet you drafted is already structurally sound: it treats eXpFS as the “hardware-neutral” base, then layers neuromorph governance (neurorights, SMART, EVOLVE) as an

orthogonal control plane over the same root-plus-blocks abstraction.[2] [3] [4]

1. Refined layout: keep eXpFS clarity, expose neuromorph layers

A good way to "refactor" is to make the layering explicit:

- **Layer 0 – Storage geometry**: fixed blocks, root file, free list, XEXE compatibility preserved. [3] [4] [2]
- **Layer 1 – Shard typing**: ROOT, DATA, EXEC, NEURO_STREAM, BIOSPEC, LEDGER, MODEL, SOVEREIGN_CONFIG.
- **Layer 2 – Governance descriptors**: neurorights, SMART scope, EVOLVE requirements, RoH ceilings.
- **Layer 3 – Protections and policies**: AuraBoundaryGuard, SoulNonTradeableShield, DreamSanctumFilter, BioLoadThrottle, SovereignKernelLock.

Your table already expresses Layers 0–2; the only change I'd make is to call SOVEREIGN_CONFIG an explicit "kernel namespace" that is always mounted read-only at runtime and only mutated via EVOLVE tokens and Donutloop entries.[3]

2. Rust-side spec struct: "OrganicCpuFsSpec"

A compact Rust "spec sheet" for neuromorph eXpFS (conceptual, non-IO) could look like this:

```
// filename: src/neurofs/spec.rs
// destination: ./src/neurofs/spec.rs

use std::time::Duration;

#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub enum FsBlockClass {
    Generic,
    NeuroStream,
    BioSpec,
    Ledger,
    Model,
    SovereignConfig,
}

#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub enum FsFileType {
    Root,
    Data,
    Exec,
    NeuroStream,
    BioSpec,
    Ledger,
    Model,
    SovereignConfig,
}

#[derive(Debug, Clone)]
pub struct NeurorightsFlags {
```

```
    pub mental_privacy: bool,
    pub mental_integrity: bool,
    pub cognitive_liberty: bool,
    pub noncommercial_neural_data: bool,
    pub soulnontradeable: bool,
    pub dreamstate_sensitive: bool,
    pub forbid_decision_use: bool,
    pub forget_sla_hours: u32,
}

#[derive(Debug, Clone)]
pub struct SmartScope {
    pub maxeffectsizel2: f32,
    pub domains: Vec<String>,
    pub expiry: Option<Duration>,
    pub physioguard_enabled: bool,
    pub revocable: bool,
}

#[derive(Debug, Clone)]
pub struct EvolveRequirement {
    pub required: bool,
    pub scope_paths: Vec<String>,
    pub roh_ceiling: f32,
}

#[derive(Debug, Clone)]
pub struct ShardGovernance {
    pub neurorights: NeurorightsFlags,
    pub smart_scope: Option<SmartScope>,
    pub evolve: EvolveRequirement,
}

#[derive(Debug, Clone)]
pub struct ShardClassSpec {
    pub file_type: FsFileType,
    pub block_class: FsBlockClass,
    pub extensions: Vec<String>,
    pub description: String,
    pub governance: ShardGovernance,
}

#[derive(Debug, Clone)]
pub struct OrganicCpuFsSpec {
    pub disk_block_words: u32,
    pub shard_classes: Vec<ShardClassSpec>,
}
```

This is the neuromorph equivalent of the eXpFS spec page: one struct that still assumes “disk is a sequence of blocks” but enriches each file class with governance and neurorights envelopes. [2] [3]

3. Mapping your table into the Rust spec

The associations you listed correspond directly to **ShardClassSpec** entries:

- **ROOT / SOVEREIGN_CONFIG**: extensions `["root", "neuro-workspace.manifest.aln", ".rohmodel.aln", ".stake.aln", ".neurorights.json", ".smart.json", ".evolve-token.json", ".tsafe.aln", ".vkernel.aln"]`, `FsFileType::SovereignConfig`, `FsBlockClass::SovereignConfig`; `EvolveRequirement.required = true`, `roh_ceiling = 0.3`.
- **NEURO_STREAM**: `.neuroaln`, `.nstream.neuroaln`, `.lifaln`; block class `NeuroStream`; SMART domains: sampling frequency, window length, parallel readers, constrained by physioguard from `.ocpuenv` and `.lifeforce.aln`.
- **BIOSPEC**: `.biospec.aln`, `.ocpu`, `.ocpuenv`, `.ocpulog`, `.biosession.aln`, `.lifeforce.aln`; SMART can adjust duty cycle and limits within Tsafe/vkernel bounds.
- **LEDGER**: `.donutloop.aln`, `.evolve.jsonl`, `.nnet-loop.aln`, `.answer.ndjson`, `.bchainproof.json`; append-only, SMART disabled or strictly read-only.
- **MODEL**: `.nnetx`, `.nnetw`, `.nnetq`, `.nfeat.aln`, `.nmap.aln`, `.nnet-policy.aln`, `.nnet-cap.aln`, `.nnet-rights.json`; SMART allowed to do local retuning; EVOLVE required for structural changes and new RoH regions.

With that mapping, the FS remains pedagogically close to eXpFS: any student familiar with root + blocks + XFS can see that you've just added typed shards and governance descriptors, not replaced the underlying abstraction.[5] [3]

4. How this serves biophysical augmentation (post-2025)

For an augmented citizen on NeuroPC, the refactored spec:

- **Separates everyday tuning from deep evolution**: SMART handles "comfort-level" adjustments (sampling rate, minor thresholds), while EVOLVE handles structural shifts in risk models, kernels, and governance.[3]
- **Binds governance to physiology**: BioLoadThrottle couples `.ocpuenv`, `.biosession.aln`, `.lifeforce.aln` into IO and training limits, so the FS itself refuses operations that would exceed fatigue or RoH ceilings.
- **Guarantees neurorights at the storage layer**: AuraBoundaryGuard, SoulNonTradeableShield, and DreamSanctumFilter are enforced in the same place that eXpFS enforces owner/permission, turning neurorights into "first-class disk semantics," not just application guidelines.[2] [3]
- **Keeps compatibility with eXpOS pedagogy**: XEXE remains the executable format, and a NeuroXFS tool still feels like XFS—same verbs, but with proof verification, neurorights-aware export, and ledger writing on each sensitive change.[4] [5]

That combination is what makes this a genuine neuromorph/OrganicCPU spec-sheet rather than just "file tags": the classic eXpFS logic stays intact while governance and neurorights become mandatory constraints on every file operation.

⁂

A neuromorph, sovereign file-system can keep the same “map the project by purpose” idea, but pivot from thesis/diagrams to shard roles, neurorights, and governance surfaces, under a single sovereign root.exposnitc.github+2

1. Top-level layout: sovereign neuromorph workspace
   text
   / # Sovereign root (single manifest, eXpFS-style)
   /manifest/ # Canonical indices and specs
   /shards/ # All typed neuromorph files (flat or lightly grouped)
   /machine/ # NEXSM / NeuroPC architecture + simulator glue
   /tools/ # NeuroXFS + governance tooling
   /docs/ # Human-readable specs, diagrams, tutorials

/manifest
text
/manifest/neuro-workspace.manifest.aln # Subject ID, canonical paths, invariants
/manifest/policieslayout-2026.aln # Shard layout spec + rotation rules
/manifest/bostrom-sovereign-kernel-001.ndjson
/manifest/arch_spec_nexsm_neuropc.aln # NEXSM + OrganicCPU coupling

Purpose:
Single sovereign root that replaces “README + folder diagram” with a machine-parsable manifest of all RoH, neurorights, and shard indexes.exposnitc.github+1
2. Shard tree: files by neuromorph class
text
/shards/
root/ # ROOT & SOVEREIGN_CONFIG
biospec/ # OrganicCPU envelopes & metrics
neuro/ # Streams and dream/QLearn shards
model/ # Neural models & policies
ledger/ # Donutloop + answer streams
exec/ # XEXE + low-level kernels
data/ # Generic configs, datasets

/shards/root
text
/shards/root/root # eXpFS-style root image (flat namespace)[web:2]
/shards/root/rohmodel.aln # Risk-of-Harm model
/shards/root/stake.aln # Stakeholder governance (Host/OrganicCPU/etc.)
/shards/root/neurorights.json # NeurorightsPolicyDocument
/shards/root/smart.json # SMART token policy layer
/shards/root/evolve-token.json # EVOLVE token schema
/shards/root/tsafe.aln # Tsafe controller specs

/shards/root/vkernel.aln # Viability kernel polytopes
/shards/root/nnetfs-index.aln # Model index
/shards/root/neurofs-index.aln # Neuromorph shard index

Focus: immutable kernel namespace; only EVOLVE+ledger can change these, never direct edits.
/shards/biospec
text
/shards/biospec/subjectA.ocpu # OrganicCpuProfile
/shards/biospec/subjectA.ocpuenv # BioLimits envelopes
/shards/biospec/subjectA.ocpulog # OrganicCpu decision log
/shards/biospec/subjectA.biosession.aln # Session metrics
/shards/biospec/subjectA.lifeforce.aln # Lifeforce envelopes
/shards/biospec/OrganicCpuRuntimeMetrics-*.aln*
/shards/biospec/OrganicCpuQLearnProfiles-.aln

Focus: BioLoadThrottle and physioguard: all IO and SMART tuning consult these shards before acting.
/shards/neuro
text
/shards/neuro/subjectA.neuroaln # Binary dream/neural snapshots
/shards/neuro/subjectA.nstream.neuroaln # High-rate stream microshards
/shards/neuro/subjectA.lifaln # LIF neuromorphic state
/shards/neuro/subjectA.nstream.index.aln # Stream index (timestamps, RoH slices)

Focus: AuraBoundaryGuard and DreamSanctumFilter; export and model training are constrained by neurorights flags and EVOLVE scope.
/shards/model
text
/shards/model/subjectA.nnetx # Sovereign model spec
/shards/model/subjectA.nnetw # Weights
/shards/model/subjectA.nnetq # Quantized weights
/shards/model/subjectA.nfeat.aln # Feature spec
/shards/model/subjectA.nmap.aln # Feature mapping (neuro → inputs)
/shards/model/subjectA.nnet-policy.aln # Allowed domains, monotone flags
/shards/model/subjectA.nnet-cap.aln # Capability descriptor
/shards/model/subjectA.nnet-rights.json # Model-bound neurorights
/shards/model/subjectA.nnet-evolve.jsonl # EvolutionProposal stream (model)
/shards/model/subjectA.nnet-loop.aln # Donutloop ledger for model states
/shards/model/subjectA.nnet-proof.bchain.json # Blockchain proof envelope

Focus: SMART retuning vs EVOLVE architecture changes, all anchored via Donutloop and proofs.
/shards/ledger
text
/shards/ledger/evolve.jsonl # Global EvolutionProposal stream
/shards/ledger/donutloop.aln # Global evolution ledger

/shards/ledger/answer.ndjson # ChatAnswerQuality stream
/shards/ledger/bchainproof.json # Biophysical-blockchain proof envelope

Focus: append-only, immutable history of every impactful change.
/shards/exec
text
/shards/exec/kernel.xexe # eXpOS / NEXSM kernel image[web:35]
/shards/exec/loader.xexe # XEXE loader
/shards/exec/shell.xexe # Console shell
/shards/exec/interrupt_timer.xexe # Timer ISR
/shards/exec/interrupt_disk.xexe # Disk ISR
/shards/exec/interrupt_console.xexe # Console ISR
/shards/exec/neurofs-daemon.xexe # OrganicCPU FS guard

Focus: classic NEXSM semantics, but all disk accesses are now mediated by neuromorph protections.exposnitc.github+1
/shards/data
text
/shards/data/config.aln # System configs (non-sovereign)
/shards/data/datasets/ # Non-neural datasets
/shards/data/logs/ # Generic logs

Focus: “normal” eXpFS DATA files, with modest SMART scopes and strict neurorights boundaries.
3. Machine and tool layout
/machine
text
/machine/nexsm_arch.html # NEXSM dual-core architecture[web:35][web:36]
/machine/nexsm_arch.png
/machine/neuro-nexsm-bridge.aln # Mapping NEXSM cores ↔ OrganicCPU envelopes
/machine/xsm_instruction_cycle.html # Instruction cycle (baseline)

/tools
text
/tools/neuroxfs/ # NeuroXFS CLI + daemon
/tools/neuroxfs/neuroxfs # Binary
/tools/neuroxfs/config.toml # Tool configuration
/tools/neuroxfs/scripts/ # Automation (format, anchor, verify)
/tools/guards/ # Rust guard libraries
/tools/guards/aura_boundary_guard.rs
/tools/guards/soul_non_tradeable_shield.rs
/tools/guards/dream_sanctum_filter.rs
/tools/guards/bio_load_throttle.rs
/tools/guards/sovereign_kernel_lock.rs

NeuroXFS mirrors XFS verbs (format, load, export, df, ls) but always routes through guards and ledger writes.exposnitc.github+1
4. Docs: human view of the sovereign layout

text
/docs/
neuromorph_fs_overview.md
neurorights_and_smart.md
evolve_workflow.md
nexsm_neuropc_design.md
diagrams/
access_lock_table_nexsm.png # From NEXSM multi-core spec[web:35][web:36]
biospec_flow.png
donutloop_flow.png
neuroxfs_flow.png

These replace the project/thesis PDFs in the original map with boundary diagrams: where multi-core kernel code touches neuromorph shards, and how access locks + neurorights gates keep the state consistent.exposnitc.github+1

This refactored directory and layout keeps the spirit of the NEXSM project map (clear areas for spec, implementation, tools, docs) but restructures it around shard classes, neurorights governance, and sovereign protections, making it a clean base for neuromorph organic-computing with strong neural-networking boundaries.

This workspace layout is consistent with eXpFS and NEXSM design principles while cleanly re-centering everything around shard roles, neurorights, and governance under a single sovereign root. [13] [14] [15] [16]

Why this structure works

- It preserves the **single-root, flat logical namespace** of eXpFS, with `/shards/root/root` still acting as the canonical root file image and root entry table. [17] [16]
- It mirrors the way **XFS treats the disk.xfs image as a unit**, but replaces ad-hoc project folders with a manifest-driven, machine-parsable workspace `/manifest/neuro-workspace.manifest.aln` and `policieslayout-2026.aln`. [13] [17]
- It aligns with NEXSM's multi-core constraints: kernel code in `/shards/exec` still follows classic access-lock semantics, but now all critical disk structures are further wrapped in neuromorph guards and ledgers to prevent inconsistent updates across cores. [15] [18]

Governance surfaces by area

- **/manifest**: the "sovereign control plane" describing subject ID, canonical shard paths, RoH model, stake, neurorights, and layout rules; it replaces human-only README diagrams with **authoritative state** that both kernel and NeuroXFS act on. [16] [17]
- **/shards/root** (SOVEREIGN_CONFIG + ROOT): immutable kernel namespace, only updated via EVOLVE tokens and Donutloop entries; this is the direct extension of the eXpFS root file concept into a neuromorph governance root.
- **/shards/biospec** and **/shards/neuro**: the **physiology layer**, where BioLoadThrottle and physioguard enforce RoH-aware limits on IO and SMART tuning before any model or external tool can touch the streams.

- **/shards/model**: the **neural capability layer**, where SMART can retune but cannot cross architecture or RoH boundaries without EVOLVE approval and blockchain proof envelopes.
- **/shards/ledger**: the **temporal spine** of the system, keeping append-only histories of evolution and answers; this reflects the eXpOS emphasis on consistent OS data structure updates, extended into cryptographically anchored logs.[15]
- **/shards/exec**: standard XEXE kernels, loaders, ISRs, and neuromorph daemons, maintaining the same upward ABI for programs while routing all disk operations through guards, in line with NEXSM's "downward interface changes, upward unchanged" design.[18] [15]
- **/tools** and **/docs**: external interfaces (NeuroXFS) and human explanations that stay faithful to XFS semantics—format, load, export, dump—but always go through AuraBoundaryGuard, SoulNonTradeableShield, DreamSanctumFilter, BioLoadThrottle, and SovereignKernelLock before touching the underlying eXpFS-like disk layout.[13]

As a result, the project is still "mapped by purpose" like the original NEXSM/eXpOS documentation, but the purposes are now explicitly: secure sovereign kernel configuration, biophysical envelopes, neural streams, model evolution, ledgers, and governance tooling, all anchored to a single sovereign root and enforced at the same level that classic eXpFS enforced file ownership and permissions.

⁂

eXpOSNeuro system-automations, and workflow-orchestrations within the boundaries of Github that prevent raw neural-data from leaking, and keep a per-host instance of sustainable network-functionality for experimental, lab-grade use with XR-grids, and urban-infrastructure for smart-city planning, and future-development.

For eXpOSNeuro on GitHub, you can treat the repo as a **sovereign per-host lab space** with strict automation that never pushes raw neural data and still supports XR/smart-city experimentation.[24] [25] [26]

1. Repository boundaries: what lives on GitHub vs per-host

In a neuromorph variant:

- GitHub hosts only **code, schemas, and synthetic fixtures**: Rust crates, `.aln` specs, `.json` policy templates, NeuroXFS tooling, and synthetic/demo `*.neuroaln` clips that contain no real neural traces.
- Each OrganicCPU/host keeps a **local neuromorph workspace** (e.g., under `/var/neurofs/subjectA/`) with real `.neuroaln`, `.lifaln`, `.ocpuenv`, `.lifeforce.aln`, and ledger shards that never leave the host.

- CI workflows run **unit and property tests against synthetic data** plus structural checks (schema validity, RoH invariants) but are forbidden from mounting the actual local neuromorph workspace.

Result: GitHub is a **template and tooling hub**, not a datastore for raw neural or XR telemetry.

2. GitHub Actions: safeguarded automations

You can enforce this separation via repository-level automation:

- **Branch protection + code owners**
  - Protect branches that hold SOVEREIGN_CONFIG templates (`rohmodel.aln`, `neurorights.json`, `smart.json`); require review from a small "sovereign kernel" team before merging.
- **Secrets and environments**
  - Use GitHub Environments (e.g., `lab-xr-grid`, `smart-city-sim`) with scoped secrets only for **simulated** infrastructure (test XR grids, lab K8s clusters), never for subject identities or real XR city systems.[27]
- **Workflows that explicitly deny neural payloads**
  - Add a "neural-leak" guard job that greps for `.neuroaln`, `.lifaln`, or large binary blobs and fails if any are added outside a `/fixtures/synthetic/` tree.
- **Schema validation as a gate**
  - CI validates `.neurorights.json`, `.smart.json`, `.evolve-token.json`, `.tsafe.aln`, `.vkernel.aln` against JSON-Schema/ALN schemas to guarantee that per-host instances can safely consume them.
- **Signed releases**
  - Publish signed container images and binaries (NeuroXFS, neuromorph guard daemons) from GitHub Releases, so each host can reproduce a known-good toolchain.

All workflows operate purely on **specs and code**, not on live data streams.

3. Per-host orchestration: sustainable, lab-only networks

On each lab or XR node (OrganicCPU + NEXSM/NeuroPC), you can orchestrate a sustainable instance:

- **Local neuromorph FS mount**
  - Mount `/srv/neurofs/` with the shard layout you defined (root/biospec/neuro/model/ledger/exec/data).
  - NeuroXFS runs as a daemon, exposing a local API or socket for lab tools and XR sims to request file operations, with protections enforced by AuraBoundaryGuard, BioLoadThrottle, etc.
- **Local scheduler and resource caps**
  - Use systemd/containers/Kubernetes to run per-subject pods that mount their own neuromorph shards read-write and others read-only or not at all.

    - Resource quotas and cgroup limits align with `.ocpuenv` and `.lifeforce.aln` so that high-rate XR or smart-city simulations don’t overwhelm the OrganicCPU envelope.
- **Lab-grade only network connectivity**
    - Nodes that interact with XR grids or city-scale sims are tagged as **lab-only** and egress-filtered so they cannot POST raw neuromorph shards to the internet or to GitHub.

This preserves **sustainable per-host instances** that can be torn down or archived without leaking personal neural traces.

4. Data-safe XR and smart-city workflows

For XR grids and urban infrastructure planning:

- **Simulated feeds derived from obfuscated or aggregated data**
    - XR tools and smart-city planners ingest either synthetic neural streams or aggregated metrics (e.g., fatigue index / attention heatmaps) derived locally from `.neuroaln` and `.lifeforce.aln` but never the raw shards.
- **GitHub-hosted scenario definitions**
    - GitHub stores scenario configs (`xr_scenario.aln`, `city_grid_config.json`) and model policies, which the per-host system pulls and runs against local data.
- **On-host model adaptation**
    - When adjusting models for XR or city planning tasks, SMART policies allow limited tuning (e.g., weights for traffic prediction) on host, logging to `.nnet-evolve.jsonl` and `.nnet-loop.aln`; only anonymized model deltas or hyperparameter configs are pushed back to GitHub, not training data.
- **External collaboration via structural metrics**
    - Urban-lab collaborators see performance and sustainability metrics (latency, energy use, congestion scores) plus code/config diffs via GitHub, but never per-citizen neural or XR traces.[25] [27]

So GitHub becomes a **collaborative design and governance plane** for eXpOSNeuro, while each OrganicCPU host remains the **sovereign execution and data plane**, ensuring no raw neural data leaks even as you integrate XR grids and smart-city planning experiments.

⁂

<q>
{"id":"bootstrap_sanitized_001","kind":"ui_library","name":"Bootstrap_3_2_0_Sanitized","license":"MIT","source":"https://getbootstrap.com/","notes":["JavaScript components require jQuery","Provides plugins: alert, button, carousel, collapse, dropdown, modal, tooltip, popover, scrollspy, tab, affix"],"recommended_upgrade":{"version":"5.3.7","npm":"bootstrap@5.3.7","jquery_dependency":false}}

{"id":"nexsm_xfs_make_001","kind":"build_script","name":"NEXSM_XFS_Automation","description":"Automation for eXpOS/eXpFS/NEXSM build and xfs-interface workflows","variables":{"XFS_IFACE":"./xfs-interface","DISK_IMG":"disk.xfs","DATA_DIR":"data-files","EXEC_DIR":"exec-files"},"targets":["all","build","clean","run","xfs-setup","xfs-format","xfs-load-files","xfs-list","xfs-export","xfs-wipe","test"],"flows":[{"name":"build","steps":["Compile LaTeX project.tex (if present)","Compile Mythesis_Btech.tex (if present)","Optionally compile code via nested Makefile"]},{"name":"xfs-setup","steps":["Format disk image with xfs-interface fdisk","Load .dat and .xexe files into disk.xfs"]},{"name":"run","steps":["clean","build","xfs-wipe","xfs-setup","xfs-list"]}],"directory_assumptions":{"data":"data-files/*.dat","exec":"exec-files/*.xexe","disk_image":"disk.xfs","xfs_tool":"xfs-interface"}}
{"id":"neuroxfs_flow_001","kind":"neurofs_flow","name":"NeuroXFS_Integration","description":"Neuromorph / OrganicCPU-safe workflow derived from classic xfs-interface usage","steps":["Format sovereign disk image with neurorights-aware metadata","Load EXEC shards (.xexe) into EXEC zone","Load DATA/LEDGER/MODEL shards with governance descriptors","List and inspect shards via neuroxfs ls","Export only shards that pass neurorights and SoulNonTradeableShield checks"],"sovereign_guards":["AuraBoundaryGuard","SoulNonTradeableShield","DreamSanctumFilter","BioLoadThrottle","SovereignKernelLock"]}</q> What is the name, and location of the ndjson file in the follow-up?

The NDJSON file should be named:

- **Filename:** `neuroxfs_spec.ndjson`
- **Destination/path:** `./spec/neuroxfs_spec.ndjson`

⁂

Experimental-Modes, and AI-Units that can-be matyhematically-structured, and rendered post-processed after 5-Dimensional shaping under neuromorph_conditions for EVOLVE.

Experimental modes and AI units under neuromorph conditions can be treated as 5-dimensional objects whose coordinates encode state, time, risk, and biophysical load, then evolved via constrained transformations.

1. Five-dimensional state for neuromorph EVOLVE

One useful 5D shape for an AI unit (model, stream, or policy slice) is:

- x_1: **Representational state** (embedding / weight manifold coordinate).
- x_2: **Virtual time / spike horizon** (neuromorphic runtime or sequence depth).[37]
- x_3: **Risk-of-Harm axis** (RoH slice per `.rohmodel.aln`, normalized to $[0, 1]$).
- x_4: **Bio-load axis** (fatigue / lifeforce index from `.ocpuenv`, `.biosession.aln`, `.lifeforce.aln`).[38]
- x_5: **Governance axis** (discrete/continuous EVOLVE + neurorights regime for that unit).

An *experimental mode* is then a 5D region $M \subset \mathbb{R}^5$ specifying allowed coordinates and trajectories for an AI unit under EVOLVE. EVOLVE proposals become maps $T : M \to M'$ that must obey RoH ceilings and neurorights constraints.

2. Experimental mode types (conceptual)

You can define a few standard neuromorph modes as 5D regions:

1. **Intra-Token Neuromorph Mode**
   - Tight bounds on x_1, short x_2 horizon, strict low x_3 and x_4 ceilings.[37]
   - Use for local feature reshaping and SMART tuning with minimal physiological impact.
2. **Inter-Token Associative Mode**
   - Larger x_2 span, more freedom in x_1 but stricter governance on x_5 (e.g., forbiddecisionuse for dream streams).[37]
   - Used when models mix across long histories or multiple streams.
3. **Organic Neuromatrix Mode**
   - Bio-load x_4 becomes primary: operations allowed only in a low-fatigue hyper-slab, motivated by organic neuromorphic hardware limits.[39] [38]
   - Suitable for on-chip neuromorphic matrices on OrganicCPU or organic devices.
4. **Quarantine / Sandbox Mode**
   - Wide range for x_1 and x_2 but extremely tight caps on x_3 (RoH) and hard constraints on x_5 (no EVOLVE, no external export).

- Used for testing new architectures or imported models with unknown behavior.

Each “mode” is a parameterized polytope or curved region in 5D; EVOLVE tokens specify how an AI unit may move between these regions over time.

3. AI units as EVOLVE objects

Within this framework, an **AI unit** is any entity with:

- A **state manifold** coordinate (weights, features, or neuromatrix conductances).[38]
- A **runtime footprint** (virtual time depth, spike horizon).[37]
- A **RoH profile** against `.rohmodel.aln`.
- A **bio-load profile** against `.ocpuenv` / `.biosession.aln`.
- A **governance profile** (neurorights + EVOLVE/SMART scopes).

Examples of units:

- A single `.nnetx` + `.nnetw` model snapshot with its `.nnet-policy.aln`.
- A specific NEURO_STREAM shard window (e.g., last 10 minutes of `.nstream.neuroaln`).
- An organic neuromatrix tile on hardware (synapse array segment).[39] [38]

Each unit then has a current 5D coordinate; an EVOLVE operation is allowed only if:

- It stays within Tsafe/vkernel constraints in the x_3, x_4, x_5 axes.
- It respects neurorights (no movement into regions that imply dreamstate decision-use, commercialization of soulnontradeable data, etc.).

4. Post-processing and rendering

Once these 5D modes are defined, you can post-process and render:

- **Projections**: 2D or 3D slices (e.g., RoH vs bio-load vs time) to visualize safe vs unsafe evolution paths for a unit.
- **Energy–accuracy tradeoff curves**: map virtual time axis vs RoH vs bioload to show where neuromorphic run time can be reduced without exiting the safe 5D region.[39] [37]
- **Mode occupancy maps**: count how often units occupy each experimental mode, and audit how many EVOLVE steps approach the boundary of the 5D safe region.

This makes “experimental modes and AI units” mathematically concrete: each is a point or region in a structured 5D space, and EVOLVE is the governed process of moving within and between those regions under neuromorph conditions.

⁂

1. home-finance-travel-shopping-a-dNHOagaRTy6u.Z7jAK0olQ.md
2. https://exposnitc.github.io/os_spec-files/eXpFS.html
3. https://exposnitc.github.io/expos-docs/os-spec/expfs/
4. https://exposnitc.github.io/expos-docs/abi/

5. https://exposnitc.github.io/expos-docs/support-tools/xfs-interface/
6. https://exposnitc.github.io
7. https://exposnitc.github.io/support_tools-files/xfs-interface.html
8. home-finance-travel-shopping-a-dNHOagaRTy6u.Z7jAK0olQ.md
9. https://exposnitc.github.io/expos-docs/os-spec/
10. https://exposnitc.github.io/expos-docs/os-spec/expos-abstractions/
11. https://exposnitc.github.io/expos-docs/
12. https://www.xmos.com/documentation/XM-014363-PC/html/tools-guide/tools-ref/formats/xe.html
13. https://exposnitc.github.io/expos-docs/support-tools/xfs-interface/
14. https://exposnitc.github.io/expos-docs/roadmap/stage-02/
15. https://exposnitc.github.io/os_design-files/nexpos.html
16. https://exposnitc.github.io/expos-docs/os-spec/expfs/
17. https://exposnitc.github.io/expos-docs/os-design/disk-ds/
18. https://exposnitc.github.io/expos-docs/roadmap/stage-28/
19. https://exposnitc.github.io/support_tools-files/xfs-interface.html
20. https://www.cencenelec.eu/areas-of-work/xfs_cwa16926_340_release/
21. home-finance-travel-shopping-a-dNHOagaRTy6u.Z7jAK0olQ.md
22. https://exposnitc.github.io/expos-docs/documentation/
23. https://stackoverflow.com/questions/76296012/where-can-i-find-a-public-version-of-the-xfs-specification-document-for-atm-cash
24. https://exposnitc.github.io/expos-docs/roadmap/stage-28/
25. https://www.utilitieslabs.com/empowering-smart-cities-ai-and-xr-for-sustainable-utilities/
26. https://exposnitc.github.io/expos-docs/os-implementation/
27. https://aaiconsortium.org/smart-city-lab/
28. https://github.com/joshsaxe/eXposeDeepNeuralNetwork
29. https://exposnitc.github.io
30. https://github.com/exp0se
31. home-finance-travel-shopping-a-dNHOagaRTy6u.Z7jAK0olQ.md
32. https://exposnitc.github.io/expos-docs/roadmap/
33. https://github.com/arunnats/NITC-eXpOS-S6
34. https://arxiv.org/pdf/2008.03563.pdf
35. https://exposnitc.github.io/release_notes.html
36. home-finance-travel-shopping-a-dNHOagaRTy6u.Z7jAK0olQ.md
37. https://www.arxiv.org/pdf/2601.00245.pdf
38. https://pmc.ncbi.nlm.nih.gov/articles/PMC10667342/
39. https://polytechnic.purdue.edu/facilities/laboratory-organic-bio-electronics-lobe/research/organic-neuromorphics
40. https://neuromorphic.com
41. https://github.com/llama-risk/aave-research/blob/main/frameworks/aave_v3_framework.md

42. home-finance-travel-shopping-a-dNHOagaRTy6u.Z7jAK0olQ.md
43. http://business.mammothtimes.com/mammothtimes/article/tokenring-2026-1-21-the-brain-inspired-revolution-neuromorphic-computing-goes-mainstream-in-2026
44. https://www.pnas.org/doi/10.1073/pnas.2504164122
45. https://www.youtube.com/watch?v=4PKqMrpiKlc
46. https://github.com/THULemon/Neuromorphic-Computing
47. https://www.evolvetoken.io/whitepaper.pdf