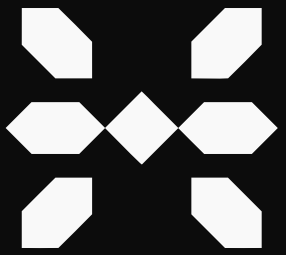
HOOX · AXIS · DOCUMENTATION



SYSTEMS ENGINEERS · CONTRIBUTORS

ARCHITECTURE MANUAL

ADRs, topologies, state namespaces, AXIS ≠ engine

Open charting PWA — own the axes, swap the engine

2026-08-03 DIN A4 · HTTPS://HOOX.SH/AXIS/DOCS OPEN SOURCE

CONTENTS

01 Overview

02 Overview

03 Adrs

04 Topologies

05 State Namespaces

OVERVIEW

title: "Architecture" description: "AXIS system architecture: AXIS vs engine, plugin registry, topologies, ADRs, and state namespaces."

ARCHITECTURE

AXIS architecture is the discipline of **separable axes**—price history, live time, calculation, and library storage—coordinated by a Solid AXIS UI that never owns a closed interpreter.

Abstract

Layer	Responsibility
AXIS	UI, store, plugin orchestration, chart apply
Plugin contracts	`source | stream | engine | storage` (+ reserved component)
Engines	PYNE evaluation (browser Pyodide or remote `/run`)
Optional edge	Cloudflare Pages + Worker + DO/KV/D1/R2
Optional desk API	Flask Pro API

Invariant: AXIS ≠ engine. Evaluation always crosses an engine plugin boundary.

Track map

Page	Contents
Overview	End-to-end data/control flow
ADRs	ADR-001 … ADR-013
Topologies	Dev, static, edge deploy shapes
State namespaces	Keys, migration, hash, IDB

Conceptual model

```
flowchart TB
  subgraph AxisUI["Browser AXIS"]
    UI[Solid UI]
    ST[store pynescript.axis.v1]
    REG[PluginRegistry]
    UI --> ST
    UI --> REG
  end
  REG --> SRC[Sources]
  REG --> STR[Streams]
  REG --> ENG[Engines]
  REG --> STO[Storages]
  ENG -->|server| API[Flask or Worker proxy]
  ENG -->|pyodide| PY[In-browser PYNE]
  STO -->|cloud| API
  STO -->|git| FORGE[GitHub/GitLab]
  STO -->|local| IDB[(IndexedDB)]
```

Formal namespaces

```
Contract namespace:  pynescript.axis.plugins.v1
App state key:       pynescript.axis.v1
App state prefix:    pynescript.axis.*
CF project id:       pynescript-axis   (frozen)
Health service id:   pynescript-axis-worker
```

Related tracks

- End User — operator workflows
- UI — UI subsystem design
- Plugins — contracts & catalogs
- Worker — edge data plane
- DevOps — build & CORS
- PYNE runtime — language evaluation

See also

- `frontend/README.md` — package-level map
- `frontend/LEGACY.md` — pre-Solid shell boundary

OVERVIEW

title: "Architecture overview" description: "End-to-end AXIS architecture: Solid AXIS UI, unified registry, dual engines, chart apply pipeline, and optional edge plane."

ARCHITECTURE OVERVIEW

Abstract

AXIS is a **composition host**. The product thesis: own the axes (history, live, calc, library), swap the implementations. Shipping UI is Solid + Vite + lightweight-charts + CodeMirror 6; calculation is never inlined into UI components.

Conceptual model

```
flowchart TB
  subgraph Boot
    I[index.tsx] --> A[app.tsx]
    A --> B[registerBuiltins]
    A --> R[restoreInstalledPlugins]
    A --> L[loadSymbolData]
    A --> P[prefetch/preload Pyodide]
  end
  subgraph Runtime
    TOP[Topbar] -->|Load| SRC[SourcePlugin.fetchHistorical]
    SRC --> BARS[store.bars]
    BARS --> CH[PaneManager / ChartHost]
    TOP -->|Run| RUN[runAndApply]
    RUN --> ENG[EnginePlugin.run]
    ENG --> LR[lastRun]
    LR --> CH
    LR --> RES[ResultsPanel]
    TOP -->|Live| MUX[streams/multiplex]
    MUX --> BARS
    LIB[ScriptLibrary] --> STO[StoragePlugin]
  end
```

Interface surface (module map)

Concern	Primary paths
Entry	`frontend/src/index.tsx`, `app.tsx`
State	`frontend/src/store/`
Plugins	`frontend/src/plugins/{registry,types,loader,bootstrap,active}.ts`
Catalogs	`sources/catalog.ts`, `streams/*`, `engines/catalog.ts`, `storage/*`
Chart	`chart/ChartHost.tsx`, `pane-manager.ts`, `drawing-layer.ts`
Editor	`editor/*`
Run apply	`indicators/runner.ts`
Results	`results/*`, `ui/ResultsPanel.tsx`
Worker	`frontend/worker/`

Legacy parallel: `state.js`, `registry.js`, `main.js` — not product path. See repo `frontend/LEGACY.md`.

Control plane vs data plane

Plane	What moves	Who owns it
Control	activePlugins, endpoint, theme, layout	Solid store + localStorage
Historical data	Bar arrays	Source plugins → `store.bars`
Live data	Bar updates	Stream plugins → multiplex → bars/chart
Calculation	script + bars → RunResult	Engine plugins
Library	ScriptDocument	Storage plugins
Telemetry	plane connectivity / latency / ticks	Ephemeral `store.telemetry` + Connection HUD

Transport preference (ADR-014)

Path	Preferred transport	Notes
Load (history)	REST / local	Venue kline HTTP, CSV, mock walk
Live	**WebSocket**	Venue WSS with reconnect; mock-poll local; DO = brokered
Engine server	HTTP	`POST /run` batch; latency in HUD
Engine pyodide	Local	Offline-capable after asset cache

Live pairs with history via `defaultStreamForSource`. Optional `live.preferAfterLoad` auto-starts WS after Load (default **off**).

Dual engines

Engine	Transport	Fidelity host
`server`	HTTP JSON `{ script, data }` → `/run`	Flask or Worker→Flask
`pyodide`	In-worker-thread-ish browser Python	Self-hosted wheels + `pynescript_runtime.py`

Timeout policy in runner scales with bar length (clamped). Live re-runs use shorter silent timeouts.

Chart apply pipeline

`runAndApply`:

1. `getActiveEngine().run(...)`
2. `setLastRun`
3. Optionally open results
4. **Sync** overlays in place (`syncOverlayLines`) — no blank destroy frame
5. Markers + strategy report side effects
6. Atomic Pine script drawings replace
7. Equity pane when appropriate (skip hide thrash on silent live re-runs)

History vs live data path: `loadBars` bumps `chartDataGen` → ChartHost full `setDataToChart` + fit. Live ticks only `manager.appendBar` (never full setData/fitContent).

PaneManager is imperative DOM under a Solid-owned outer shell—**Solid must not reconcile the pane root** (ChartHost invariant).

Plugin registry

`PluginRegistry` holds ordered maps per kind; events `registered` / `unregistered`. Built-ins register once (`ensureBuiltins` / catalog `ensure*Registered`). Dynamic plugins via `loadPluginFromUrl`. Active selection is **not** inside the registry—it is store state.

Contract namespace conceptually: `pynescript.axis.plugins.v1` (fields: `id`, `name`, `kind`, `configSchema`, `capabilities`, kind-specific methods).

Edge plane (optional)

Cloudflare project **`pynescript-axis`**:

- Pages: static `dist/`
- Worker: `/api/run` proxy, keys/usage KV, scripts D1, stream DO fan-out
- AXIS still speaks engine/storage plugins—not raw CF APIs in UI components

Invariants

1. AXIS ≠ engine
2. Bars/logs/lastRun not fully persisted
3. Storage and engine endpoints may coincide (Worker) but remain separate plugin roles
4. Older keys migrate forward, never the reverse on write

Failure modes (architectural)

Mode	Manifestation	Mitigation
God-object UI	Logic in components	Keep runner/registry pure modules
Engine leakage	Fetch `/run` from random widgets	Only engine plugins perform calc I/O
Registry/store split brain	Flat `engine` ≠ `activePlugins.engine`	`setActivePlugin` keeps alignment
SPA asset fallback	Pyodide loads HTML	Assert ZIP magic on assets

See also

- ADRs
- Topologies
- State namespaces
- AXIS store

ADRS

---

title: "Architecture Decision Records" description: "ADR-001 through ADR-014 for AXIS: registry, AXIS≠engine, Solid+Vite, dual engines, configSchema, URL load, storage, CF project id, proxies, DO fan-out, migration, CORS, results export, transport preference + telemetry."

ARCHITECTURE DECISION RECORDS

Formal ADRs for AXIS (product + `frontend/`). Status values: **Accepted** unless noted. Dates are conceptual product epochs, not git archaeology.

Index

ID	Title
ADR-001	Pluggable unified registry
ADR-002	AXIS ≠ engine
ADR-003	Solid + Vite product UI
ADR-004	Dual engines (server + Pyodide)
ADR-005	Declarative configSchema
ADR-006	URL-loadable plugins
ADR-007	Storage as a plugin
ADR-008	Frozen CF project id `pynescript-axis`
ADR-009	Worker proxies first
ADR-010	Durable Object stream fan-out
ADR-011	State namespace migration
ADR-012	CORS as deploy concern
ADR-013	Results export in AXIS
ADR-014	Transport preference + connection telemetry

ADR-001: Pluggable unified registry

Context

Early AXIS registered sources/streams/engines in separate ad hoc maps. Operators and authors needed one mental model and one install path.

Decision

Adopt a **unified** **`PluginRegistry`** (`frontend/src/plugins/registry.ts`) with kinds:

`source | stream | engine | storage | component(reserved)`

Every plugin shares `PluginBase` (`id`, `name`, `kind`, `description`, `version`, `builtIn`, `configSchema`, `capabilities`, lifecycle hooks). Registration is ordered; listeners observe register/unregister.

Consequences

- Active selection lives in the Solid store, not the registry.
- Built-ins protected from casual unregister.
- Catalogs (`sources/catalog.ts`, …) become registration facades.
- Contract docs under Plugins.

ADR-002: AXIS ≠ engine

Context

Closed chart hosts couple rendering and language runtime. That blocks offline mode, alternate evaluators, and headless reuse of PYNE.

Decision

AXIS never embeds a closed interpreter. All evaluation goes through `EnginePlugin.run`. UI modules call `getActiveEngine()` / `runAndApply`, not Python bindings directly (except inside the pyodide engine module).

Consequences

- Language fidelity owned by PYNE.
- AXIS docs do not duplicate grammar theory.
- New engines (tiny demo, remote Worker, future WASM) are additive plugins.

ADR-003: Solid + Vite product UI

Context

Legacy vanilla JS shell (`main.js`, `style.css`) mixed DOM wiring with business logic and aged TV-blue tokens.

Decision

Product path is **SolidJS + Vite + Tailwind 4**, entry `src/index.tsx` / `app.tsx`. Imperative chart library (lightweight-charts) is isolated in `ChartHost` / `PaneManager` so Solid reconciliation does not thrash canvas DOM.

Consequences

- Legacy files retained for smoke/static tests only (`LEGACY.md`).
- Icons via `lucide-solid`.
- Deploy artifact is Vite `dist/`, not repo-root static tree.

ADR-004: Dual engines (server + Pyodide)

Context

Researchers need server fidelity and offline demos. A single transport cannot satisfy both without compromise.

Decision

Ship two built-in engines:

Id	Role
`server`	`POST {endpoint}/run?mode=` with `{ script, data: bars }`
`pyodide`	Browser Python + vendored pynescript / antlr wheels

Both implement the same `EnginePlugin` contract and return `RunResult`.

Consequences

- Endpoint field only for server-like engines.
- Pyodide requires correct static asset deployment (ZIP integrity checks).
- Timeouts unified at runner layer.

ADR-005: Declarative configSchema

Context

Per-plugin settings UIs do not scale; custom React/Solid forms per plugin forks the manager.

Decision

Plugins declare optional **`configSchema`**: map of field name → `{ type, default, label, options, min, max, ... }`. Settings and library panels resolve defaults + `pluginsConfig` overrides via shared resolve helpers.

Consequences

- New plugins get settings UI without shell changes (within field types).
- Types: `string | number | boolean | select` initially.
- Invalid user values remain a plugin responsibility at runtime.

ADR-006: URL-loadable plugins

Context

Third-party sources/engines should not require rebuild of the PWA for experiments.

Decision

Support **`loadPluginFromUrl`**: fetch ES module, validate shape, register, persist install metadata for restore on boot. Example plugins published under `public/plugins/`.

Consequences

- Browser origin trust model: URL plugins are code execution.
- Security tests constrain schemes and storage-via-URL footguns.
- Offline restore needs prior successful fetch/cache.

ADR-007: Storage as a plugin

Context

Script library began as localStorage-only; cloud and git were bolted differently.

Decision

Storage is a first-class plugin kind with `list/read/write/remove` (+ optional draft/sync/status). Built-ins: `local` (IDB), `cloud` (Worker `/api/scripts`), `git` (GitHub/GitLab Contents API). Active storage in `activePlugins.storage`.

Consequences

- Manager Script Library is backend-agnostic.
- Git commits only on explicit save; drafts local.
- Cloud uses Bearer Pro keys and optional optimistic concurrency.

ADR-008: Frozen CF project id `pynescript-axis`

Context

Renaming Cloudflare Pages/Worker projects severs KV/D1/R2 bindings, custom domains, and CI secrets. Product brand evolved to AXIS.

Decision

Keep infrastructure name `pynescript-axis`. Brand in UI/manifest/docs is AXIS. Health JSON may identify `pynescript-axis-worker`. Optional DNS aliases (`axis.*`) point at the same project without rename.

Consequences

- `wrangler.toml` `name` and Pages `--project-name` stay frozen.
- Docs call out the freeze explicitly (this ADR).
- npm package name may lag brand.

ADR-009: Worker proxies first

Context

In-Worker Pyodide is attractive but heavy; production needed a path immediately.

Decision

Worker **`/api/run` proxies to an external Python backend** first (`EXTERNAL_BACKEND`). In-Worker runtime remains scaffolded/gated. Keys, usage metering, scripts API, and stream DO can ship independently of full edge evaluation.

Consequences

- AXIS `server` engine talks to Worker URL transparently.
- Operational dependency on Flask (or equivalent) until edge runtime matures.
- Clear layering: Worker is data plane + auth, not necessarily the interpreter.

ADR-010: Durable Object stream fan-out

Context

N browser clients each opening venue WebSockets waste connections and risk rate limits.

Decision

`SessionDO` **Durable Object**: one upstream WS per session key; fan-out to N client sockets via `/api/stream`. Hibernation-friendly design on CF.

Consequences

- Stream plugins may target DO URLs (example plugin provided).
- Session/symbol/interval query params are part of the contract.
- Not a historical source—live only.

ADR-011: State namespace migration

Context

Hard-renaming storage namespaces would brick user layouts and libraries if keys changed hard.

Decision

Canonical key `pynescript.axis.v1`. On read, fall back to `pynescript.axis.v2` (and parallel library/draft keys), then **write forward**. Do not keep dual-write forever.

Consequences

- Seamless upgrade for existing browsers.
- Persistence strips bars/lastRun/logs.
- Tests cover migration (`tests/state.test.ts` patterns).

ADR-012: CORS as deploy concern

Context

Browser AXIS on origin A calling Pro API on origin B fails without CORS. Hardcoding origins in AXIS is wrong.

Decision

CORS is a backend/deploy configuration (`ALLOWED_ORIGINS` on Flask; Worker headers as deployed). AXIS documents required origins; ops sets explicit lists in production (avoid perpetual `*` outside demos).

Consequences

- Localhost regex for dev.
- Troubleshooting centers on preflight, not engine code.
- Static file server does not replace API CORS.

ADR-013: Results export in AXIS

Context

Researchers need artifacts (trades CSV, run JSON) without a separate analytics service.

Decision

Results panel owns export: download JSON of `lastRun`, CSV of closed trades, clipboard helpers. Strategy pairing and stats run client-side from engine events (`results/strategy.ts`, `results/events.ts`).

Consequences

- Export fidelity bounded by engine event quality.
- No server-side report store required for MVP.
- Viewer metrics ≠ broker statements (documented for operators).

ADR-014: Transport preference + connection telemetry

Context

Operators need to see **how** data and calculation move (WS vs REST vs local vs brokered), whether the live socket is actually open, and engine run latency. Naively "prefer WebSocket everywhere" is wrong: venues expose **history over REST** and **live klines over WSS**. Server engines batch-run over HTTP; Pyodide is local WASM.

Decision

1. **Transport policy**
   - History → REST (or local mock/CSV).
   - Live → **WebSocket first**, paired per source via `defaultStreamForSource` (`binance-rest` → `binance-ws`, …).
   - Engine → **prefer** `WS /ws/run` on the Pro API when available (`flask-sock`); fall back to HTTP `POST /run`. Pyodide remains local.
   - Brokered → CF Durable Object / `needsProxy` streams surface as transport class `broker`.
2. **Honest stream state** — `connecting` until `onStatus({ state: 'open' })`; reconnect uses `reconnecting` / degraded telemetry, not false green.
3. **Venue streams use reconnect with exponential backoff** (`streams/reconnect-ws.ts`).
4. **Connection HUD** — StatusBar second row (`ConnectionHud`) reads ephemeral `store.telemetry` (SRC/STR/ENG/STO chips, tick pulse, engine latency). Persist only `telemetry.hud` prefs + `live.preferAfterLoad` / `live.rerunOn`.
5. **Live re-run modes** — `every-tick` (default) or `bar-close` (venue `Bar.closed` or bar time advance).
6. **Chart stability** — full `setDataToChart` only on history load (`chartDataGen`); live uses `appendBar`; overlay/script drawings update in place to avoid hide/show flash.

Consequences

- UI never claims WS for historical load.
- Auto-live after Load is **opt-in** (`preferAfterLoad`, default off).

- Plugin capabilities may declare `transport`; otherwise HUD classifies from id/capabilities.
- See Streams, UI shell, Overview.

ADR lifecycle

New ADRs should:

1. State context, decision, consequences.
2. Cross-link code paths.
3. Avoid re-litigating ADR-002 without supersession note.

See also

- Overview
- Topologies
- Plugins contracts

TOPOLOGIES

title: "Topologies" description: "AXIS deploy and dev topologies: Vite+Flask, static PWA, Cloudflare Pages+Worker, and offline Pyodide lab."

TOPOLOGIES

Concrete process and network shapes for AXIS. Choose a topology; then compose plugins inside it (recipes).

Abstract

Topology	AXIS	Calc	Live	Library
Dev desk	Vite `:3000`	Flask `:5002`	Binance WS direct	local
Static demo	`axis_pwa_server` `:8081`	Flask or remote	venue / mock	local
Edge	CF Pages	Worker → Flask	DO fan-out optional	cloud
Offline lab	any AXIS	Pyodide	mock-poll	local

Topology A — Dev desk

```
flowchart LR
  V[Vite AXIS app :3000] -->|REST klines| BN[Binance]
  V -->|WS klines| BN
  V -->|POST /run| F[Flask :5002]
  F --> PYNE[pynescript runtime]
```

Bring-up

```
make run
cd frontend && bun install && bun run dev
```

Notes: Vite may proxy `/run`; Settings endpoint can still point absolute at `:5002`. CORS must allow Vite origin.

Topology B — Static PWA + Pro API

```
flowchart LR
  S[axis_pwa_server :8081] --> DIST[dist/]
  DIST -->|browser| API[Flask :5002]
  DIST --> BN[Venue APIs]
```

```
cd frontend && bun run build
python3 axis_pwa_server.py
make run   # separate process
```

VPS demo pattern: systemd units for `axis-pwa` and `pynescript-api`; `ALLOWED_ORIGINS` includes AXIS origin.

Topology C — Cloudflare AXIS at the edge

```
flowchart TB
  U[Browser] --> P[CF Pages pynescript-axis]
  U --> W[Worker same project]
  W -->|proxy /api/run| F[EXTERNAL_BACKEND Flask]
  W --> KV[KV keys/usage]
  W --> D1[D1 scripts]
  W --> DO[SessionDO stream]
  DO --> UP[Upstream venue WS]
  U -->|optional direct| BN[Venue REST/WS]
```

Deploy sketch

```
cd frontend/worker && wrangler deploy
cd frontend && bun run build
wrangler pages deploy dist --project-name=pynescript-axis
```

Frozen id: never rename `pynescript-axis` lightly (ADR-008).

AXIS config:

- Engine `server`, endpoint = Worker origin
- Storage `cloud`, same origin + API key
- Stream: direct venue or DO-backed plugin

Topology D — Offline lab

```
flowchart LR
  G[UI] --> MW[mock-walk]
  G --> MP[mock-poll]
  G --> PY[pyodide engine]
  G --> IDB[(local storage plugin)]
```

No Flask, no venue. Requires vendored pyodide + wheels on origin once.

Topology E — Git-centric research

Any of A–C for calc/data; storage axis = `git`. Forges are orthogonal network peers:

```
AXIS --storage:git--> api.github.com | gitlab
AXIS --engine:server--> Flask/Worker
AXIS --source--> venue or CSV
```

Port map (defaults)

Service	Port
Vite dev	3000
Flask Pro API	5002
Static PWA server	8081
Wrangler Worker	8787

Comparison

Concern	A Dev	B Static	C Edge	D Offline
HMR	yes	no	no	n/a
PWA SW realism	partial	full	full	full
Multi-tenant keys	no	no	yes	no
Ops complexity	low	medium	high	low
Fidelity	high (Flask)	high	high via proxy	engine-dependent

Failure modes by topology

Topology	Typical break
A	Flask not running; CORS localhost vs 127.0.0.1 mismatch
B	Stale SW; missing `dist` pyodide assets
C	Unbound KV/D1; wrong project name; EXTERNAL_BACKEND down
D	First visit offline; SPA fallback for wheels

Internals

Path	Role
`frontend/vite.config.ts`	Dev/build
`frontend/axis_pwa_server.py`	Static host
`frontend/worker/wrangler.toml`	CF name + bindings
`frontend/worker/README.md`	Edge ops

See also

- ADR-008, 009, 010, 012
- Installation
- Worker
- DevOps

STATE NAMESPACES

title: "State namespaces" description: "AXIS persistence keys, key migration, pluginsConfig, editor docs, library IDB, and URL hash state."

STATE NAMESPACES

AXIS persistence is multi-homed: **layout/config** in localStorage, **script library** in IndexedDB (or remotes), **ephemeral run data** in memory, optional **URL hash** for shareable slices.

Abstract

Namespace	Medium	Purpose
`pynescript.axis.v1`	localStorage	App shell state
`pynescript.axis.editor.doc`	localStorage	Editor document backup
`pynescript.axis.storage`	IndexedDB	Local library
`pynescript.axis.library.v1`	localStorage fallback	Library if no IDB
`pynescript.axis.library.draft`	localStorage	Draft buffer
Plugin install list	localStorage	URL plugins restore
`store.bars` / `lastRun` / `logs`	memory	Session only

Contract / product identifiers (not storage keys):

```
pynescript.axis.plugins.v1   # conceptual plugin contract id
pynescript-axis        # CF project id (infra)
```

Conceptual model

```
flowchart TB
  subgraph Persist
    LS[localStorage axis.v1]
    ED[editor.doc]
    IDB[(IDB scripts)]
  end
  subgraph Memory
    BARS[bars]
    LR[lastRun]
    LOGS[logs]
  end
  UI[Solid store] --> LS
  UI --> BARS
  UI --> LR
  LIB[storage local] --> IDB
  LEG[older keys] -->|migrate on read| LS
```

App state key `pynescript.axis.v1`

Solid store (`frontend/src/store/index.ts`):

Persisted (representative):

- Market: `symbol` , `interval` , `exchange` , `source` , `engine` , `endpoint`
- `activePlugins` `{ source, stream, engine, storage }`
- `pluginsConfig`
- Layout: `theme` , `editor` , `watchlist` , panel open/width/height
- `panes` , `scripts` (indicator list metadata)
- `live.streamId` (and related live flags carefully)
- `drawings` (user annotations)

- Flat mirrors: `source` / `engine` aligned via `setActivePlugin`

Explicitly omitted on persist:

- `bars`
- `lastRun`
- `logs`

Debounced write (~200ms).

Legacy migration

Read order:

1. `pynescript.axis.v1`
2. else `pynescript.axis.v2`

On older-key hit: copy forward into `pynescript.axis.v1`.

Legacy vanilla `state.js` uses the same STORAGE_KEY constant for the pre-Solid shell.

pluginsConfig

Map of configuration objects. Preferred keys:

```
`${kind}:${id}`   e.g. storage:cloud, storage:git, engine:server
```

Bare ids may still be read for compatibility (`cloud`, `git`). Values feed `configSchema` resolution inside plugins.

Sensitive fields: `apiKey`, git `token` — browser-local; treat profile as secret store.

Editor document

Key	Role
`pynescript.axis.editor.doc`	Draft / shared doc
Bridge messages	Popout synchronization

Popout mode uses `editor-bridge` + shared storage so detached windows share source without re-fetching library.

Local library IDB

Constant	Value
DB name	`pynescript.axis.storage`
Version	1
Stores	`scripts`, `kv`

Migration flags (`pynescript.axis.library.migrated`) prevent re-import loops from older library keys:

- `pynescript.axis.library.v1`
- `pynescript.axis.library.legacy`

Cloud / git (remote namespaces)

Not browser keys—server-side partitions:

Backend	Partitioning
cloud	Hash of API key on Worker; D1 table or memory
git	`owner/repo@branch` + `basePath`

Revisions: cloud may expose `revision` / If-Match; git uses blob SHAs via forge APIs.

URL hash state (legacy helper)

`frontend/src/state-hash.js` (legacy state path):

Feature	Detail
Keys	symbol, interval, engine, source, stream, timeRange
Script	base64 in hash if short
Cap	~2000 chars
API	`applyHashState`, `pushHashState`, `watchHashState`

Solid-first product may not wire this by default; treat as optional share mechanism. Prefer library export + recipe docs for durable shares.

Invariants

1. Migration is **read-repair** to AXIS keys.
2. Never persist full OHLCV in `axis.v1`.
3. `setActivePlugin` keeps flat fields coherent.
4. Logs panel open state forced closed on hydrate (noise control).
5. Drawing tool resets to `cursor` on hydrate; geometries restore.

Failure modes

Issue	Effect	Mitigation
QuotaExceeded	persist no-op	Export library; shrink drawings
Poisoned JSON	load defaults	Security tests; clear key
Split profiles	“missing” library	Same browser profile
Stale activePlugins id	missing plugin	Fall back / reselect built-in

Internals

Path	Role
`frontend/src/store/index.ts`	Solid persistence
`frontend/src/store/types.ts`	`AppState`
`frontend/src/state.js`	Legacy state
`frontend/src/state-hash.js`	Hash sync
`frontend/src/storage/local.ts`	IDB library
`frontend/src/plugins/loader.ts`	Installed plugin list

See also

- ADR-011
- AXIS store
- Script library