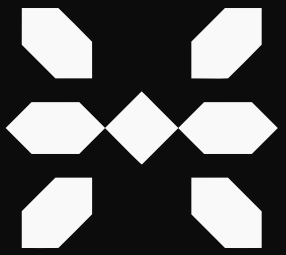
HOOX · AXIS · DOCUMENTATION



FRONTEND ENGINEERS · UX CONTRIBUTORS

UI MANUAL

Chart, editor, UI shell, indicators, results, store

Open charting PWA — own the axes, swap the engine

CONTENTS

OVERVIEW

title: "UI" description: "AXIS UI subsystem: chart host, editor, UI shell, indicators, results, and Solid store—presentation without a closed engine."

UI

UI is the AXIS browser presentation layer: everything the operator sees and clicks that is *not* language evaluation. Engines are plugins; AXIS applies their outputs to panes, markers, and drawers.

Abstract

Subsystem	Responsibility
Charting	Panes, multi-chart, replay, compare, drawings, badges
Editor	CM6, diagnostics, ruler, git bar, stats, dock/popout
Debugging	Scriptlogs, Profiler, Debug chips, Pins, Problems
UI shell	Topbar groups, docks side-by-side, palette, alerts
Indicators	Run pipeline, overlay/sub-pane routing
Results & strategy	Event normalize, trades, equity, export
Store	Solid store, persistence, active plugins

Conceptual model

```
flowchart TB
  SHELL[UI shell] --> STORE[Solid store]
  ED[Editor] --> STORE
  ED -->|Run| IND[indicators/runner]
  IND --> ENG[Engine plugin]
  ENG --> IND
  IND --> CH[Chart host]
  IND --> RES[Results]
  STORE --> CH
  STORE --> SHELL
```

Invariant (AXIS ≠ engine): no Solid component imports PYNE directly; only engine modules talk to Python/HTTP runtimes.

Interface surface (visual map)

```
┌─ Topbar (grouped: market · data · compute · layout · panels · system) ─┐
├─ left docks ─┬─ ChartHost (1–4 slots) + drawings ─┬─ right docks ──────┤
│  watchlist   │  multi-pane + badges + replay       │  indicators|editor │
│  layers …    │                                     │  (side-by-side)    │
├─ Results / scriptlogs (bottom) ──────────────────────────────────────────┤
├─ System logs ─────────────────────────────────────────────────────────────┤
└─ StatusBar (connection HUD) ──────────────────────────────────────────────┘
   + Settings · Plugins · Command palette (⌘K)
```

Implemented in `src/app.tsx` (repo root product UI).

Stack

Concern	Choice
Framework	SolidJS 1.9
Bundler	Vite 6 + `vite-plugin-solid`
Charts	lightweight-charts 5
Editor	CodeMirror 6
Icons	lucide-solid
Style	Tailwind 4 + void tokens (`data-theme`)

Relation to other tracks

- Operators: End User
- Structure: Architecture
- Contracts: Plugins
- Language: PYNE runtime

See also

- `frontend/LEGACY.md` — what is *not* the AXIS product path
- `frontend/TESTING.md` — unit/e2e around UI modules

CHARTING

title: "Charting" description: "AXIS chart: ChartHost, PaneManager, series factory, drawings, crosshair sync, and trade markers."

CHARTING

Charting is the spatial surface for OHLCV, overlays, and annotations. Implementation is **imperative** under a Solid shell so lightweight-charts owns its DOM subtree.

Abstract

Module	Role
`ChartHost.tsx` / `ChartWorkspace.tsx`	Solid mount; multi-slot grid; empty-state
`pane-manager.ts`	Multi-pane LWC charts, time sync, overlay apply
`pane-badge.ts`	Corner name + script action icons
`series-factory.ts`	Candle/volume/line series helpers + palette
`chart-type.ts`	Price style transforms (HA, hollow, ...)
`bar-replay.ts` + `BarReplayControls`	History scrub / play
`layout.ts` / `layout-recipes.ts` / `chart-registry.ts`	Multi-chart layouts + presets
`compare-overlay.ts`	Second-symbol compare series
`volume-profile.ts`	OHLCV volume-at-price overlay
`drawing-layer.ts` + `drawings/*`	User + script drawings, templates, sync helpers
`crosshair-sync.ts`	Cross-pane crosshair
`pine-drawings.ts`	Engine drawing objects → layer
`manager-access.ts`	Process-wide manager/layer getters

Conceptual model

```
flowchart TB
  HOST[ChartHost Solid] --> EL[panesEl div]
  EL --> PM[PaneManager]
  PM --> PRICE[price pane]
  PM --> VOL[volume pane]
  PM --> IND[indicator pane?]
  PM --> EQ[equity pane?]
  PM --> DL[DrawingLayer]
  RUN[runAndApply] --> PM
  RUN --> DL
  BARS[store.bars] --> PM
```

Interface surface

Panes

`store.panes`: `{ id, type, height, order, visible, label }`.

type	Typical series
`price`	Price series (style below) + overlay lines + markers + drawings
`volume`	Histogram
`indicator`	Non-overlay plots
`equity`	Strategy equity curve

Defaults: price + volume. Indicator/equity created on demand.

Price chart styles (`store.chartType`)

Topbar **Type** select (persisted). Implementation: `chart/chart-type.ts` + `createPriceSeries` .

chartType	Series	Notes
`candles`	Candlestick	Japanese candles (default)
`hollow`	Candlestick	Hollow body on up bars
`bars`	Bar	Classic OHLC bars
`heikinashi`	Candlestick	Heikin-Ashi transform of OHLCV
`line`	Line	Close only
`area`	Area	Close with fill
`baseline`	Baseline	Close vs first-bar base

Series still live under pane key `candle` so markers, hlines, and the drawing layer keep a stable host. Changing type swaps the LWC series and rebinds data without a full history reload.

Empty / status overlay

When `bars.length === 0` , ChartHost shows contextual title/sub from `store.status` (loading, error, running, or “Load data to begin”).

Multi-chart layouts

- Modes: **1 / 2H / 2V / 4** (`store.chartLayout`)
- Per-slot symbol/interval via registry; active slot drives topbar + Run
- **Named layouts** save/load (optional chrome snapshot)
- **Recipes** in Layouts menu: Scalp 1m+5m, Swing HTF, Quad, BTC majors (`layout-recipes.ts`)

Bar Replay

Topbar **Replay** → session over loaded OHLCV (`bar-replay.ts`):

- Enters with **full history visible** (cursor on last bar)
- Scrub / step / play; **Play at end restarts from bar 0**
- Mutually exclusive with Live

Compare & volume profile

- **Compare** control: second symbol as % or absolute overlay (`compare-overlay.ts`)
- **Volume profile** toggle in Layers → OHLCV approximation histogram (`volume-profile.ts`)

Pane badges

Each pane’s top-left chip (`pane-badge.ts`): label + script actions when scripts are applied:

Icon	Action
Settings	Script inputs modal
Eye	Show / hide series
Refresh	Re-run script
Trash	Remove from chart

Volume/equity without scripts: name + hide.

Drawings toolbar

See Drawings. Tools include cursor, hline, vline, trend, ray, extend, rect, ellipse, arrow, fib, measure, text. Templates and duplicate helpers live under Layers / drawings modules.

Crosshair & time

`syncTimeScales` / `syncCrosshair` keep multi-pane navigation coherent. `scrollToTime` / `jumpToDebugPin` used from Results and debug pins.

Internals

Solid vs imperative boundary

```
// ChartHost — Solid owns outer shell
// PaneManager only mutates panesEl — never put Solid children inside panesEl
```

On cleanup: destroy drawing layer, dispose manager, clear accessors.

Data path

`setDataToChart(bars)` applies OHLCV; `createEffect` on `store.bars` re-applies if bars change outside load helper.

Run apply (chart side)

From `runAndApply` (indicators):

- `syncOverlayLines` — update existing overlay series in place (avoids destroy→blank→add flash on live re-runs)
- Markers from normalized events (set in place)
- Script drawings atomic replace via DrawingLayer (`setScriptDrawings` without empty frame)
- Equity series when strategy report warrants (silent live runs skip hide thrash)

History vs live bars

- Full loads: `loadBars` → `chartDataGen++` → `setDataToChart` + fitContent
- Live: `multiplex` → `appendBar` + `manager.appendBar` only (no fitContent, no full setData)

series-factory

Centralizes series construction options and color palette so Results plot summary and chart stay chromatically related.

Invariants

1. Solid reconciliation must not rewrite pane roots.
2. Script drawings ephemeral per run; user drawings store-backed.
3. Overlay routing: engine `meta.overlay` + script type; **oscillator-scale** series forced to sub-pane when they would vanish on price (see Indicators).
4. Shared non-overlay sub-pane id is stable: **`indicator`**.
5. Manager may be null before mount—runner no-ops chart apply safely.

Failure modes

Symptom	Cause
Blank panes after hot reload	Manager disposed; full remount
Overlays stack forever	Old path skipped removeOverlays
Fib/tools ignore clicks	No bars / time scale; wrong tool
Markers at t=0	Event times not bar-aligned

Worked example (dev)

```
getManager()?.scrollToTime(barTime);
```

See also

- Indicators
- Results and strategy
- End-user drawings

EDITOR

title: "Editor" description: "AXIS Pine editor: CodeMirror 6, diagnostics, 80-col ruler, git sync, stats, docked vs popout, editor bridge."

EDITOR

The editor is the **source axis of intent**: Pine text the operator authorizes the engine to run. Completion and hover can use local builtins metadata and optional remote LSP; **evaluation** is always an engine plugin.

Abstract

Module	Role
`EditorPane.tsx`	Docked/standalone chrome, header tools, detach
`tabbed-editor.tsx`	Multi-tab docs, stats strip, Problems, git bar
`PineEditor.tsx`	CM6 mount (extensions + view APIs)
`pine-language.ts`	StreamLanguage / highlighting
`pine-lsp.ts` + `data/pine-builtins.json`	Offline completion/hover corpus
`column-ruler.ts`	80-character guide line
`diagnostics.ts`	Underlines, gutter, jump-to from last run
`inline-debug.ts`	EOL chips, pin gutter, line flash
`git-sync.ts` + `ui/EditorGitBar.tsx`	Pull / push / storage status
`ui/EditorProblems.tsx`	Collapsible problems list
`editor-bridge.ts`	Cross-window doc + run messages
`cm-void.ts`	Void-theme CM styling

Conceptual model

```
sequenceDiagram
  participant Main as Main AXIS
  participant Bridge as editor-bridge
  participant Pop as Popout window
  Main->>Bridge: writeSharedDoc
  Main->>Pop: openEditorWindow
  Pop->>Bridge: hello role=editor
  Main->>Bridge: doc push
  Pop->>Bridge: run
  Bridge->>Main: runAndApply on main
  Main->>Bridge: run-status
```

Why run on main? Chart, bars, and pane manager live in the main window. Popout is a pure editing surface.

Interface surface

Docked mode

- Right dock by default (`panelChrome.editor`); may sit **side-by-side** with other right panels (e.g. Indicators left of Editor)
- FloatableShell: dock left/right/bottom, float, new window
- Detach → popup or tab (`openEditorWindow`)

Popout mode

- Main shows “Editor detached” chip + **Reattach**

- Shared doc key + bridge messages: `hello`, `doc`, `run`, `run-status`, `reattach`, `popout-opened`, `popout-closed`

Run entry points

- Editor / topbar **Run** → `runAndApply(getDoc())`
- Popout Run → bridge → main executes
- Command palette: save library, git push/pull, jump to line, toggles

Header tools (`axis-editor-tools`)

Control	Effect
Scriptlogs	Open last-run `log.*` panel
Profiler	`profilerEnabled` + optional line % gutter; re-run when enabling
Debug	Inline end-of-line log/error chips (`inlineDebugEnabled`)
Pins	Chart markers + 📍 gutter (`debugPinsEnabled`); **Alt+P**
Ruler	80-column guide (`editorRulerEnabled`)

Details: Debugging.

Stats strip

Bottom of tabbed editor (`data-testid="axis-editor-stats"`):

Field	Meaning
Pos L:C	Cursor line:column (1-based)
Ln	Total lines in document
Words / Chars	Document stats
Diagnostics badge	Error/warning count from last run — click jumps to first
Problems	Expand/collapse problems list
wrap	Soft wrap is on

Diagnostics & Problems

After a run, `diagnosticsFromLastRun(lastRun, doc)` builds ranges from engine `error`, `meta.errors` / `diagnostics`, and error logs with line refs (`line 12:`, `L12:`, …).

- CodeMirror: wavy underlines, severity gutter, hover tooltips
- **Problems** panel: clickable rows → `scrollToLine` / `jumpToDiagnostic`
- Unit tests: `tests/editor-diagnostics.test.ts`, `tests/editor-problems.test.ts`

80-column ruler

`columnRulerExtension` paints a dashed guide at character column **80** (measured via CM coords). Toggle **Ruler** or command palette “Toggle Editor Ruler”.

Git sync bar

Tab toolbar **Pull / Push / status** (`EditorGitBar`):

- Uses active **storage** plugin (`local` | `cloud` | `git`)
- Push → `writeScript` (git commits when storage is git)
- Pull → optional plugin `sync('pull')` + list/reload bound library id
- Config stays in Settings / Script Library (no second credentials UI)

Completion corpus

`src/editor/data/pine-builtins.json` is synced from PYNE (`scripts/sync-pine-builtins.sh`). Offline completion/hover fall back to this catalog; optional remote LSP when the Pro API is available.

Library load

`loadLibraryDoc(doc, name)` on `editorRef` injects storage reads into the active tab.

Internals

editorRef pattern

App holds `{ getDoc, setDoc?, loadLibraryDoc?, scrollToLine?, jumpToDiagnostic?, getCursor? }` populated by CM mount—avoids full-doc store writes every keystroke (doc also mirrored to `EDITOR_DOC_KEY`).

Persistence

Key / field	Role
`store.editor` + `panelChrome.editor`	Dock geometry, open, mode
`pynescript.axis.editor.doc`	Draft text
`profilerEnabled` / `inlineDebugEnabled` / `debugPinsEnabled` / `editorRulerEnabled`	Editor tools

Invariants

1. Empty doc Run is a no-op at topbar/editor handlers.
2. Popout does not own PaneManager.
3. Reattach restores shared doc into docked CM.
4. AXIS ≠ engine: editor never evaluates Pine itself.
5. Diagnostics/Debug/Pins read **last run** only — re-run after edits.

Failure modes

Symptom	Mitigation
Two editors diverge	Prefer bridge doc events; reattach
Detached but no window	Popup blocked; allow popups; reattach
Lost text on crash	editor.doc localStorage; library Save / git push
Blank CM (no lines)	Dock column height chain; editor must fill strip
No chips/pins	Enable Debug/Pins; logs need line and/or bar_index

See also

- Debugging — Debug vs Pins, Scriptlogs, Profiler
- UI shell — Topbar, docks, command palette
- Indicators
- Script library

DEBUGGING

title: "Debugging" description: "Scriptlogs, Profiler, Debug chips, chart Pins, diagnostics, and System Logs."

DEBUGGING

AXIS separates **script-authored diagnostics** (Pine `log.*`, engine errors, line timing) from **shell telemetry** (load, stream, engine connection).

Abstract

Surface	Source of truth	Open / toggle
Scriptlogs	`store.lastRun` → `normalizePineLogs`	Editor header **Scriptlogs**
Profiler	`profilerEnabled` + `profile.lines`	Editor header **Profiler**
Debug	`inlineDebugEnabled` + last-run logs with **line**	Editor header **Debug**
Pins	`debugPinsEnabled` + logs with **bar_index / time**	Editor header **Pins** · **Alt+P**
Diagnostics / Problems	Engine errors + line-mapped logs	Underlines + Problems list (always from last run)
System Logs	`store.logs` ring buffer	Status bar / logs strip

Module	Role
`ui/ScriptLogsPanel.tsx`	Floatable last-run `log.*` list
`results/pine-logs.ts`	Normalize / filter / TSV
`results/profiler.ts`	Line % profile
`results/inline-debug.ts`	Annotations + pinable helpers
`results/debug-pins.ts`	Bar pins, markers, `countDebugPins`
`editor/inline-debug.ts`	CM chips, pin gutter, line flash
`editor/diagnostics.ts`	Error underlines / gutter / jump
`ui/EditorProblems.tsx`	Problems list UI
`chart/manager-access.ts`	`applyDebugPinsToChart`, `jumpToDebugPin`

Conceptual model

```
flowchart LR
  PIN[Pine log.* / errors] --> ENG[Engine run]
  ENG --> RR[lastRun]
  RR --> SL[Scriptlogs]
  RR --> DBG[Debug chips]
  RR --> PINZ[Pins + markers]
  RR --> DIAG[Diagnostics / Problems]
  RR --> PR[Profiler gutters]
  APP[Load / stream / boot] --> SYS[System Logs]
```

Invariant: Scriptlogs / Debug / Pins never write into `store.logs`.

Debug vs Pins

	Debug	Pins
Question	What happened on this **source line**?	Which **bar** was that on the chart?
Requires	Line ref (`line 12:` , structured `line` , ...)	`bar_index` and/or `time` / `bar_time`
UI	End-of-line chips + optional level gutter	Chart circle markers + 📍 editor gutter
Click	Pin-able chips jump if bar info present	Chip / 📍 / Scriptlogs → chart jump + line flash
Store	`inlineDebugEnabled`	`debugPinsEnabled`

Workflow

1. Add `log.info` / `log.warning` / `log.error` (or rely on engine errors) with **line** and ideally **bar_index**.
2. **Run**.
3. Toggle **Debug** to see chips; **Pins** to mark bars (count shows on the Pins button).
4. Click a pin-able chip or 📍 → `jumpToDebugPin` (crosshair + scroll); source line flashes (`cm-debug-pin-flash`).

Both can be on at once. Pins without Debug still show 📍 gutter and chart markers.

Scriptlogs

Panel for **script** logging from the last run.

Pine call	Panel level
`log.info`	info
`log.warning`	warning
`log.error`	error

Open: Editor → **Scriptlogs** (`axis-btn-scriptlogs`). Pin-able rows jump to the chart when bar time/index is known.

Editor Profiler

Toggle **Profiler** (`axis-btn-profiler`):

- Persists `profilerEnabled`
- May send `profiler: true` to the engine
- Header shows last-run ms
- Gutter shows line `%` when `profile.lines` is present

Diagnostics & Problems

Independent of Debug chips: structural **error reporting** from the run payload.

- Parses `error` , `meta.errors` , `diagnostics` , error-level logs
- CodeMirror underlines + severity gutter

- Stats strip badge (`axis-editor-diag-count`) and **Problems** list (`axis-editor-problems`)
- Click → jump to line/range

Scriptlogs vs System Logs

	Scriptlogs	System Logs
Source	Script `log.*` on last run	AXIS shell events
Store	`lastRun`	`store.logs`
UI	Floatable Scriptlogs	Strip above status bar
Control	Editor header	Status bar

Persistence & test ids

Key	Role
`panelChrome.scriptlogs` / `editor`	Panel chrome
`profilerEnabled`	Profiler
`inlineDebugEnabled`	Debug chips
`debugPinsEnabled`	Chart + pin gutter

testid	Role
`axis-btn-scriptlogs`	Open Scriptlogs
`axis-btn-profiler`	Profiler
`axis-btn-inline-debug`	Debug
`axis-btn-debug-pins`	Pins
`axis-debug-pin-count`	Pin count
`axis-editor-diag-count`	Diagnostics badge
`axis-editor-problems`	Problems panel

Unit coverage: `tests/pine-logs.test.ts` , `tests/profiler.test.ts` , `tests/inline-debug.test.ts` , `tests/debug-pins.test.ts` , `tests/editor-diagnostics.test.ts` .

See also

- Editor
- Charting — markers and jump
- UI shell

UI SHELL

title: "UI shell" description: "AXIS chrome: topbar, watchlist, status bar, settings, plugin manager, logs, and layout chrome around the chart."

UI SHELL

The shell is the non-canvas chrome: navigation of axes, market selection, modals, and status. It binds human gestures to store + plugins without owning calculation.

Abstract

Component	Path	Role
Topbar	`ui/Topbar.tsx`	Axes pickers, Load/Run/Live, theme, modals
Watchlist	`ui/Watchlist.tsx`	Symbols, quotes, click-to-load
StatusBar	`ui/StatusBar.tsx`	Status, engine/storage badges, trade snippet
SettingsDialog	`ui/SettingsDialog.tsx`	Endpoint, engine, storage, intervals
PluginManager	`ui/PluginManager.tsx`	Catalog / install / library
SystemLogs	`ui/SystemLogs.tsx`	Ring buffer of `store.logs`
ResultsPanel	`ui/ResultsPanel.tsx`	Bottom drawer (see results doc)
Icons	`ui/icons.tsx`	Lucide wrappers
ResizeHandle	`ui/ResizeHandle.tsx`	Panel widths/heights

Conceptual model

```
flowchart LR
  TOP[Topbar] --> AP[setActivePlugin]
  TOP --> LOAD[loadSymbolData]
  TOP --> RUN[runAndApply]
  TOP --> LIVE[startLive/stopLive]
  WL[Watchlist] --> LOAD
  SET[Settings] --> AP
  PM[PluginManager] --> REG[registry]
  PM --> LIB[storage service]
  SB[StatusBar] --> ST[store.status]
```

Topbar responsibilities

Grouped left → right (`axis-tb-group` , `data-tb-group`):

Group	Contents
brand	Logo + AXIS
market	Symbol, Interval, Type (`TopbarField`), Compare
data	Source (+ CSV upload), Load, Reload (icon-only)
compute	Engine, Stream, **Run** (primary), Live, Replay
layout	Multi-chart layouts + recipes
panels	List, Editor, Indicators, Layers, Alerts, Data, Inputs, Results
system	Plugins, Settings, Theme (`margin-left: auto`)

Control	Effect
Symbol / interval	Store + Load (Enter / blur reload)
Source / Stream / Engine	`setActivePlugin` + side effects
Chart type	`setChartType` (candles, HA, …)
Compare	Second-symbol overlay
Load / Reload	Historical fetch
Run	Editor doc → runAndApply
Live	multiplex start/stop (stops Replay)
Replay	Bar replay over loaded bars
Panel toggles	`isPanelOpen` / dual-write chrome
Settings / Plugins	Modal open
Detach editor	Bridge + popout

Integrated labels: `ui/TopbarField.tsx` (`.axis-tb-field*`).
`catalogTick` forces memo re-list when plugins install/remove.

Command palette

Ctrl/Cmd+K — `CommandPalette` + `command-registry.ts`: panels, theme, layouts, Run, symbol focus, editor toggles (ruler, debug, pins, profiler), jump to line, git push/pull, save library.

Alerts panel

Dockable **Alerts** (`src/alerts/*` + `AlertsPanel.tsx`): local-first price alerts, webhook optional, list/create/toggle. Continuous server-side arming is not required for session use.

Workspace snapshots

`storage/workspace-snapshot.ts` + optional chrome menus capture layout/plugin prefs for restore (distinct from OHLCV — bars still re-fetch on Load).

Panel docks (side-by-side)

Left/right docks with **2+ open panels** lay out **horizontally** (row), not stacked under each other:

- Example: **Indicators** left of **Editor** on the right strip
- Column width = **sum** of panel widths (capped)
- Each panel keeps its own width (resize handle on chart-facing edge)
- Bottom dock still stacks vertically

See `ui/panels/dock-layout.ts` , `FloatableShell.tsx` .

Watchlist

- Default universe includes majors (`BTCUSDT` , …)
- `refreshSec` clamped 5–120
- Source-aware labels (e.g. CSV mode)

- Click sets symbol and loads through active source

Status bar

Single row:

- **Left — Connection HUD** (`ConnectionHud`, `data-testid="axis-connection-hud"`)
  - LIVE badge (off / connecting / live / err)
  - Tick pulse (fixed-width price + age; ping does not resize layout)
  - Engine chip (id · interpret/compile · latency)
  - Plane chips SRC / STR / ENG / STO with transport badges **WS / REST / LOCAL / BROKER**
  - Source ↔ stream pairing warning when mismatched
- **Right** — `status` / `statusMessage`, strategy snippet, log toggle, bar/ind/pane counts

Telemetry is ephemeral (`store.telemetry`); HUD compact mode and live prefs persist via settings.

Live settings (Settings dialog)

Setting	Default	Effect
Auto-start live after Load	off	`live.preferAfterLoad` → multiplex after successful Load
Indicator re-run	every-tick	or `bar-close` only
Compact connection HUD	off	hide plane chips

Logs

System Logs strip

- `appendLog(level, message, source)`
- Cap `MAX_LOGS` (500)
- Not persisted
- Boot messages, pyodide ready, live errors

Scriptlogs & Profiler

Script `log.*` output and editor line-timing live on a separate path — see Debugging. Controls sit on the **editor header** (not the topbar) and do not write into `store.logs`.

Theming

`store.theme` → `document.documentElement data-theme` (`dark` | `light`). Void dark is default product aesthetic.

Accessibility / test hooks

E2E selectors use `data-testid` on critical controls (`axis-btn-load`, `axis-manager`, …) per `TESTING.md`.

Invariants

1. Shell never calls Python.

2. Plugin lists always come from registry facades (`listSources`, …).
3. Modal settings persist only on Save (not on every keystroke of endpoint field until save).
4. Live off leaves historical bars intact.

Failure modes

Symptom	Shell-side check
Pickers empty	Builtins not registered at boot
Live ignores click	Stream `none` or multiplex error in logs
Settings probe fails	Network / CORS—not topbar bug

See also

- Manager and settings
- Store
- Architecture overview

INDICATORS

---

title: "Indicators" description: "AXIS indicator UI: runAndApply pipeline, active engine binding, overlay routing, and IndicatorPanel."

INDICATORS

“Indicators” in AXIS terms means **scripts applied to the chart**, whether `indicator()` or `strategy()` in Pine. The module boundary is `indicators/runner.ts` plus list UI—not the PYNE evaluator.

Abstract

Piece	Role
`runScript`	Engine invocation + status/timeouts
`runAndApply`	Chart + lastRun + results open
`probeEndpoint`	Settings health check helper
`IndicatorPanel`	List/toggle scripts in store
`IndicatorCard`	Per-script UI chrome

Language semantics: PYNE runtime.

Conceptual model

```
flowchart TB
  DOC[Pine source] --> RS[runScript]
  BARS[store.bars] --> RS
  CFG[getActiveEngineConfig] --> RS
  RS --> ENG[EnginePlugin.run]
  ENG --> RR[RunResult]
  RR --> LR[setLastRun]
  RR --> CH[PaneManager overlays]
  RR --> MK[Trade markers]
  RR --> DR[Script drawings]
  RR --> RES[Results open]
```

Interface surface

runScript options

Option	Default	Meaning
`silent`	false	Quiet status; live re-run logging
timeout	derived	Interactive: scales with bars (60–180s); silent 45s

runAndApply options

Option	Default	Meaning
`openResults`	`!silent`	Open results drawer
`indicatorId`	optional	Update existing store indicator

Overlay routing

```
overlay = resolveOverlayFlag(meta.overlay, script_type)
  // explicit false → sub-pane
  // explicit true  → price
  // missing: indicator → sub-pane; strategy → price

if overlay && seriesWouldHideOnPrice(plots, bars):
  overlay = false   // RSI / scaled oscillators on BTC scale
paneId = overlay ? 'price' : 'indicator'   // stable id `indicator`
```

Non-overlay ensures store + manager pane id **`indicator`** (not a random uid). Orphan empty indicator panes are cleaned up. After apply, sub-panes force autoScale / resize so lines paint on first run.

Prefer `indicator(..., overlay=false)` and raw `plot(rsi)` for oscillators — not `strategy(..., overlay=true)` with `rsi * 0.01` on price.

Series selection

Prefer `result.series` entries whose keys do not start with `_` / `__`. Fallback: single `plots[]` array as one line named from `script_name`.

Colors: `plot_meta[k].color` or `PLOT_PALETTE[i]`.

Indicator panel & pane badges

- Side list of `store.scripts` (`Indicator`: id, name, code, paneId, visible, plots)
- Visibility toggles affect display; re-run is still engine-driven
- Chart pane chips mirror settings / eye / re-run / remove (see Charting)

Active engine binding

```
// plugins/active.ts pattern
getActiveEngine()        // registry lookup by activePlugins.engine
getActiveEngineConfig()  // pluginsConfig + store.endpoint merge
```

Switching engines mid-session does not auto-rerun; operator hits Run.

Live re-run coupling

Streams may set `live.needsRerun`; multiplex/runner path can call `runAndApply` with `silent: true` so the status bar is not spammed. Failures go to logs.

Internals

Path	Role
`frontend/src/indicators/runner.ts`	Pipeline
`frontend/src/indicators/IndicatorPanel.tsx`	List UI
`frontend/src/plugins/active.ts`	Active engine/source helpers
`frontend/src/engines/catalog.ts`	Built-in engines

Invariants

1. Runner is the **only** chart-facing calc orchestrator (avoid duplicate apply logic in components).
2. Errors still `setLastRun` with `status: 'error'` when possible.
3. Clearing overlays happens before re-adding—no unbounded series leak.
4. AXIS does not interpret Pine; it only maps `RunResult`.

Failure modes

Symptom	Layer
Engine exception	Results error + status
Success empty plots	Script/meta—not runner
Manager null	Chart not mounted; run still sets lastRun
AbortError	Timeout—reduce bars or optimize script

See also

- Charting
- Results and strategy
- ADR-002 / ADR-004

RESULTS AND STRATEGY

title: "Results and strategy" description: "UI-side event normalization, trade pairing, metrics, equity curve, and ResultsPanel export pipeline."

RESULTS AND STRATEGY

Results UI turns opaque engine payloads into **operator-consumable tables, metrics, and files**. It is a pure-ish viewer layer over `lastRun` + `bars`.

Abstract

Module	Role
`results/events.ts`	Normalize heterogeneous event shapes; markers; equity series
`results/strategy.ts`	Pair trades; stats; equity SVG helpers; CSV
`ui/ResultsPanel.tsx`	Tabs, export, jump-to-time
`ui/StrategyReport.tsx`	Stats cards, equity curve, trades table, CSV
`ui/StatusBar.tsx`	Compact trade summary

Conceptual model

```
flowchart LR
  LR[lastRun.events] --> N[normalizeStrategyEvents]
  B[store.bars] --> N
  N --> T[buildStrategyReport]
  N --> M[eventsToMarkers]
  N --> E[buildEquityCurve]
  T --> UI[Strategy tab]
  LR --> P[Plots / Metrics / Raw]
  T --> CSV[trades CSV]
  LR --> JSON[run JSON]
```

Event normalization

Engines may emit Pro API parity fields (`kind` , `bar_time` , `direction` , `ohlc`) or legacy UI fields (`type` , `time` , `dir` , `price`). Normalizer:

- Unifies time/price/dir/id
- Optionally fills price from bars when OHLC present/empty
- Can include order-like events for the Events tab

Trade pairing algorithm

Sketch (`buildStrategyReport`):

1. Sort by time.
2. On entry-like kind → open map by `id` .
3. On exit/close-like → match id, else sole open trade.
4. PnL = Δprice × long/short sign.
5. Aggregate win rate, profit factor, max DD on cumulative PnL.

Not modeled: fees, funding, partial fills, portfolio multi-symbol—viewer honesty.

ResultsPanel tabs

Tab	Data source
Events	`normalizedEvents`
Strategy	`report.trades` + `report.stats`
Plots	`series` / `plots` summary
Metrics	meta + context + stats subset
Raw	`JSON.stringify(lastRun)`

Interactions: copy, export, `scrollToTime` on trade rows. Scriptlogs / Debug **Pins** also jump the chart (`jumpToDebugPin`) when bar time/index is present — see Debugging.

Equity

`buildEquityCurve` feeds chart equity pane when runner/strategy path requests it—derived series, not engine-native unless also provided.

Export (ADR-013)

Artifact	Function
`axis-run-*.json`	Full payload
`axis-trades-*.csv`	`tradesToCsv`
Clipboard	CSV when permitted

Invariants

1. Results recompute from `lastRun` reactively; they do not re-query engines.
2. Changing bars without re-run can change normalization fills—re-run after Load.
3. `lastRun` not rehydrated from localStorage (session artifact).
4. Infinite profit factor when no losses but profits exist—display must tolerate non-finite.

Failure modes

Symptom	AXIS explanation
Events > 0, trades = 0	Unpaired or missing prices
CSV disabled	No closed trades
Jump no-ops	Manager unmounted / invalid time

Internals cross-links

Operator narrative: Strategy and results.
Architecture: ADR-013.

See also

- Indicators
- Charting

STORE

title: "Store" description: "AXIS Solid store: AppState shape, activePlugins, persistence rules, logging, and alignment helpers."

STORE

The Solid store is the **control-plane nexus** of AXIS. Plugins read configuration from it; UI binds reactively; persistence snapshots a safe subset to `pynescript.axis.v1`.

Abstract

Export	Role
`store` / `setStore`	`createStore<AppState>`
`persist`	Debounced localStorage write
`setActivePlugin`	Kinded selection + flat field mirrors
`appendLog` / status helpers	Operator-visible telemetry
`STORAGE_KEY`	`pynescript.axis.v1`

Types: `frontend/src/store/types.ts`.

Conceptual model

```
flowchart TB
  UI[Solid components] -->|setStore| S[store]
  S -->|persist subset| LS[(localStorage)]
  LS -->|loadPersisted| S
  S --> ENG[engine config readers]
  S --> SRC[source/stream readers]
  S --> CH[chart effects]
```

AppState map

Field	Persistence	Notes
`bars`	no	Session history
`symbol`, `interval`, `exchange`	yes	Market context
`source`, `engine`	yes	Flat mirrors
`endpoint`	yes	Server engine base URL
`activePlugins`	yes	Canonical four-tuple
`pluginsConfig`	yes	Per-plugin fields
`scripts`, `panes`	yes	Indicator list + layout
`live`	partial	streamId important
`theme`	yes	dark/light
`editor`, `watchlist`, panels	yes	Chrome geometry
`stream.status`	yes/volatile	Connection hint
`status`, `statusMessage`, `lastRunMs`	yes*	UX; ok to restore
`lastRun`	**no**	Forced null on hydrate
`logs`	**no**	Forced empty
`profilerEnabled`	yes	Editor profiler (line % when engine provides profile)
`inlineDebugEnabled`	yes	End-of-line Debug chips from last-run logs
`debugPinsEnabled`	yes	Chart pin markers + editor pin gutter
`editorRulerEnabled`	yes	80-col guide (default **on**)
`panelChrome` / `alertsPanel` / `layerPanel` / `dataViewPanel`	yes	Dock chrome (side-by-side left/right)
`chartLayout` / `savedLayouts`	yes	Multi-chart grid + named layouts
`compare`	prefs yes	Second-symbol overlay (bars ephemeral)
`drawingTool`	reset cursor	On hydrate
`drawings` / `drawingPrefs` / `drawingUi`	yes	User annotations + toolbar prefs
`selectedDrawingId`	**no**	Ephemeral selection

*Status fields may restore but boot also appends fresh logs.

Defaults (product)

Representative defaults from store:

Key	Default
symbol	`BTCUSDT`
interval	`1d`
source / active source	`binance-rest`
stream	`binance-ws`
engine	`server`
storage	`local`
endpoint	demo/API host or localhost depending on build era
editor width	460 docked open
watchlist	majors, 15s refresh

Treat demo endpoints as **overridable**—operators should set local URLs in desk topology.

setActivePlugin

```
setActivePlugin(kind, id):
  activePlugins[kind] = id
  if source → source = id
  if engine → engine = id
  if stream → live.streamId = id
  persist()
```

Prevents registry/store split brain (architecture overview).

Logging

```
appendLog(level, message, source?)
levels: info | ok | warn | error
MAX_LOGS = 500
```

`setStatus` maps app status to log levels for important transitions.

Legacy dual path

Vanilla `state.js` remains for legacy shell with the same storage key family. Product AXIS uses Solid store only—do not dual-write new features to `state.js` unless maintaining legacy.

Invariants

1. Hydrate never restores `lastRun` or `logs`.
2. Persist strips `bars`.
3. Key migration is read-repair (state namespaces).
4. `pluginsConfig` keys prefer `` `${kind}:${id}` ``.
5. Store is not a plugin registry—ids must exist in registry to function at runtime.

Failure modes

Mode	Behavior
JSON parse fail	Defaults
localStorage throws	No-op persist
Unknown plugin id	UI may show id; run/load fails at getActive*
Oversized drawings	Quota risk—export/clear

Testing notes

- Import `tests/setup.ts` for localStorage stubs
- Unit coverage includes store + migration patterns
- Do not require real IDB in pure store tests

See also

- State namespaces
- ADR-011
- UI shell