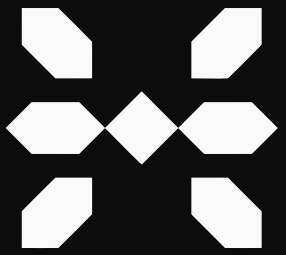
HOOX · PYNE · DOCUMENTATION



AUDITORS · CONTRIBUTORS · PARITY HUNTERS

REFERENCE MANUAL

Compatibility, status, gaps, numerical validation, pine-worker

Open-source Pine Script evaluation framework

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

CONTENTS

COMPATIBILITY

title: "Compatibility guarantee" description: "What PYNE promises for Pine Script™ v5/v6 syntax, semantics, builtins, and real-world script corpora — and what it does not."

COMPATIBILITY GUARANTEE

Abstract

PYNE aims for **high syntactic fidelity** and **strong semantic compatibility** with TradingView® Pine Script™ v5/v6 on the language core: parser, AST round-trip, type system, collections, technical builtins, and strategy primitives. Compatibility is **not** a claim of full platform parity (hosted chart, proprietary market data, editor-only features).

Source distillations: `docs/compatibility_guarantee.md` (v1.1, 2026-07-12) and live test inventory. Metrics move as the suite grows — prefer CI and the implementation status matrix for current checkmarks.

Conceptual model

```
flowchart TB
  subgraph Guaranteed_paths
    P[Parse v5/v6 grammar]
    R[AST round-trip unparse]
    E[Bar-loop evaluate]
    B[Builtin dispatch]
  end
  subgraph Partial_or_external
    D[Real market data]
    G[Pixel-perfect chart chrome]
    TV[TV-only platform APIs]
  end
  SRC[.pine] --> P --> R
  P --> E --> B
  B -.-> D
  B -.-> G
```

Interface surface

Compatibility metrics (summary)

Category	Claim	Status posture
Parser & syntax	Full grammar on real scripts (138+ builtin corpus)	Solid
AST round-trip	Structural match on corpus	Verified
Builtins	Broad coverage (200+ / dispatch hundreds)	Good → excellent by namespace
Technical indicators	Core + advanced; numerical cross-checks	Validated in tests / reports
Strategy	entry/exit/close, events, risk, OCA, commission	Functional / improved 2026-07
Type system	int/float/bool/string/color/series/UDT/collections	Strong
Real-world scripts	High parse success on corpus	Tested

Historical guarantee doc cited **1142 automated tests** at last update — re-run `make test` for live counts.

Language features (core)

Documented at 100% support posture for:

- `var` / `varip`, type annotations, functions/methods
- Control flow, UDTs, operators, string interpolation
- Imports/libraries, v6 `enum`, switch expressions
- Series history, array/matrix/map, tuple unpacking, method chaining

Builtins (guarantee lens)

Family	Guarantee style
`math.*`	IEEE-oriented precision; exact for discrete ops
`ta.*`	Cross-validated vs reference implementations within tight relative error
`str.*` / `array.*`	Behavioral parity on exercised surface
Plotting	Signature + registry effects; rendering is external (AXIS / clients)
`strategy.*`	Execution model in-process; not a licensed broker

Internals

Artifact	Role
`docs/compatibility_guarantee.md`	Long-form guarantee text
`tests/data/builtin_scripts/*.pine`	Parametrized corpus
`tests/test_parse_and_unparse.py`	Round-trip
`tests/test_real_world_compatibility.py`	Real-script posture
`tests/fixtures/parity/`	Cross-impl strategy events
`reports/compatibility_report.json`	Machine-readable reports if generated

Invariants & edge cases

1. **Parse success ≠ evaluate success.** A script may round-trip yet depend on mock `request.*` data.
2. **Plot functions are not TradingView's renderer.** They validate and record; pixels are optional.
3. **Undocumented / beta TV syntax** may fail (guarantee doc notes a v7-beta edge).
4. **Floating-point parity is statistical**, not bit-identical for all recursive smoothers — see Numerical validation.
5. **Library `export` / import** require in-process registration paths when not on TV's CDN.

Worked examples

Round-trip check

```
from pynescript.ast.helper import parse, unparse
src = open("strategy.pine").read()
assert unparse(parse(src))  # structural fidelity; exact whitespace may differ
```

Corpus pytest (narrow)

```
pytest tests/test_parse_and_unparse.py -q --example-scripts-dir=tests/data/builtin_scripts
```

Failure modes

Observation	Interpretation
Parse error on new TV release notes feature	Surface gap — see Missing features
Numerical drift on custom indicator	Compare bar-mode vs list-mode; check `na` propagation
Strategy equity mismatch vs TV	Commission/slippage/fill model differences; seed data
Import resolution fails	Library not registered in runtime

See also

- Implementation status
- Missing features
- Pine v6 surface inventory
- Numerical validation

IMPLEMENTATION STATUS

title: "Implementation status" description: "Checklist-style status of Pine Script™ series, strategy, builtins, and language constructs in PYNE."

IMPLEMENTATION STATUS

Abstract

This annex summarizes the **feature matrix** maintained in `docs/pinescript_implementation_status.md`: a large ✅/❌/🔄 inventory of series variables, strategy fields, and related surfaces. It is the human-readable twin of the full surface inventory (dispatch-oriented, auto-countable).

Status here is **implementation judgment**, not marketing completion percentage.

Conceptual model

```
flowchart LR
  TV[TradingView v6 surface] --> INV[Inventory docs]
  INV --> DISP[Evaluator dispatch map]
  INV --> CTX[Default context series]
  INV --> AST[Grammar / ASDL]
  DISP --> TESTS[pytest]
  CTX --> TESTS
  AST --> TESTS
```

Interface surface

Legend

Mark	Meaning
✅	Implemented and usable
🔄	Partial / stub / mock
❌	Missing
⚙️	By design out of scope
⬜	Editor/platform N/A

Component rollup

Component	Posture (2026-07 consolidation)
Parser (ANTLR4)	Complete for v5/v6 core + recent multiline / export work
Evaluator	Full bar-loop with var/varip, reassignment
Builtins	224+ historical count; **640 callables** on live dispatch inventory (2026-07-25)
TA	150+ `ta.*` keys in inventory
Collections	array / matrix / map complete on exercised surface
Strategy events	`StrategyEvent`, parity corpus, risk enforcement
Drawing / plot	Registry effects; AXIS is external
Linter	Present (`pynescript lint`)
Data providers	Mock, Yahoo, AlphaVantage, CCXT + datafeed wiring
LSP	Diagnostics, completion, hover, format, symbols, defs/refs, …
pine-worker	Colocated TS port (skeleton → growing)
Numba compile path	MVP + object-mode fallback

Series & context (illustrative ✅ families)

Fully listed in the source doc; representative groups:

- **Price:** `open` / `high` / `low` / `close` / `volume` / `hl2` / `ohlc4` /…

- **Time:** `year` … `second`, `time_close`, `timenow`, …
- **Barstate / chart:** `bar_index`, `barstate.*`, `last_bar_*`
- **syminfo.*** ticker, mintick, session, fundamentals fields tracked as ✅
- **session.*** / **dividends.*** / **earnings.*** context fields tracked as ✅
- **strategy.*** position, trades, equity, risk, OCA, commission constants

July 2026 enhancements called out in source

- Full strategy event emission with bar context
- `strategy.long` / `strategy.short` constants
- `var` / `varip` + `:=` reassignment
- Parity fixtures for cross-implementation validation
- Risk helpers and performance series expansion

Internals

Path	Role
`docs/pinescript_implementation_status.md`	Exhaustive checklist (do not paste wholesale here)
`docs/pine_v6_full_surface_inventory.md`	Dispatch-centric inventory
`src/pynescript/ast/evaluator/builtins/`	Per-namespace implementations
`scripts/regenerate_inventory_summary.py`	Inventory regeneration aid

Invariants & edge cases

1. **Checklist lag:** a ✅ may trail a commit by a day; prefer tests when shipping.
2. **Context fields may be defaults** rather than live exchange feeds.
3. **Partial stubs still appear callable** — inventory marks 🔄 when docstrings/heuristics say mock.
4. **Namespace counts ≠ quality.** Many drawing setters are thin mutators; TA is heavier.

Worked examples

Smoke a strategy series field

```
from pynescript.ast.helper import parse
# evaluate via Runtime / library API with OHLCV — see runtime docs
```

Regenerate inventory summary

```
python scripts/regenerate_inventory_summary.py
```

Failure modes

Symptom	Action
Doc says ✅, runtime AttributeError	File bug; fix dispatcher or demote status
Partial mock accepted silently	Assert data_feed wiring in tests
Status doc conflicts inventory	Prefer live dispatch + tests

See also

- Compatibility
- Missing features
- Pine v6 surface
- Roadmap

MISSING FEATURES

title: "Missing features" description: "Known gaps, by-design limitations, and recently closed Pine Script™ v6 items for PYNE."

MISSING FEATURES

Abstract

“Missing” in PYNE is a **moving frontier**. As of **2026-08-01**, core v6 support remains very high (~99%+ for core language), with post-launch TV items closed (multiline strings, library `export const`, footprint mocks, risk enforcement, matrix LA, compile-mode strategy, drawing GC, …) **plus** open-source corpus parse/Runtime hardening and **incremental bar-mode TA** for hot indicators. Remaining gaps cluster into **dual-host host parity**, **by-design platform differences**, **editor-only features**, **fidelity edges**, and a **long-tail Runtime/corpus** bucket.

Authoritative narrative: `docs/missing_features.md` (last updated **2026-08-01**).

Conceptual model

```
flowchart TB
  subgraph Closed_recently
    ML[Multiline strings]
    EX[export const / types / enums]
    FP[request.footprint mocks]
    MAT[matrix LA surface]
    STR[strategy runtime depth]
    CORP[Corpus sanitize + grammar]
    PERF[Incremental bar-mode TA]
  end
  subgraph Remaining
    RD[Real request.* market data]
    ED[Editor-only UX]
    EXO[Exotic TV platform edges]
    TAIL[Corpus TIMEOUT / RUN_FAIL tail]
  end
  TV[TV release notes] --> Closed_recently
  TV --> Remaining
  CORP --> TAIL
```

Interface surface

Recently completed (do not re-open as “missing”)

Summarized — full prose in source doc:

Area	Notes
Multiline `"""` / `'''`	Lexer + LexerBase + unparser
Library `export const` + types/enums	Parse/AST/runtime registry
UDT `sort_field` on array/matrix sort	Implemented
`input.*.active`	Metadata-driven
Dynamic `request.*`	Shared resolvers; mock/feed scaling
Enums runtime + input.enum	Type kind + LSP partial
Strategy open/closed trades, risk, OCA, commission	Golden tests
Drawing `*.all`, plot registry	Effects recorded
Numba compile + object mode	See compiler docs
Official matrix LA & predicates	det/inv/eigen/…
0 missing vs public TV v6 function reference list (434 symbols, 2026-07-25)	
Corpus sanitize + soft keywords / bitwise / typed UDF returns	set01–04 parse ~94.8%
Runtime `_pine_defs_locked` + append-only `current_series`	Host hygiene (backend + pyne-worker)
Incremental `ta.sma/ema/rma/rsi/macd/atr`	Bar mode; `PYNE_TA_INCREMENTAL=0` to disable
Alert engine + L2 webhooks	`alert()` / `alertcondition()` freq rules, dual-host `/run` export, HTTP webhooks

Still incomplete / by design

Item	Status posture
Real (non-mock) `request.*` market data	⚙️ By design for library core; feeds optional
Pixel chart host	External (AXIS / clients)
Some unlimited-history / trim edge cases	High-level support; exotic TV behaviors may differ
Editor word-wrap / UI-only release notes	⬜ N/A
pine-worker full builtin port	In progress (extra tool)
Bit-identical every recursive smoother vs TV	Numerical bounds, not bits
Dual Runtime host copies (**H1**)	⚙️ Host surface + dual-host **alerts** export; package-level Runtime unify still open
Warm-compile product path (**H2**)	⬜ SLOs / prewarm / deploy IR cache defaults
Corpus Runtime tail (**C1**)	⚙️ ~**94.3%** set01–04 OK projected (8-agent pass); PARSE stubs ~118; ~21 residual RUN_FAIL
Series cap (**T1**) / residual TA inc (**T2**)	⬜ Partial — hot path done; BB/nested full paths remain
TV Wilder RMA-ATR (**F1**, if re-baselined)	⚠️ Current oracle uses EMA-of-TR; change is correctness, not silent perf
Alert webhooks (**L2**)	✅ Pro API + pyne-worker (`ALERT_WEBHOOK_URL` / `webhook_url`); see Alerts

Post-v6 launch themes (historical gap list)

v6 launch items (dynamic requests, strict bool, enums, polylines, `log.*`, negative indices, truediv, ...) are largely supported or intentionally stubbed where platform-bound.

2025–2026 monthly TV updates introduced footprint types, `active` inputs, multiline strings, `export const`, UDT sort fields — tracked and largely closed in this repo's July 2026 work.

Corpus + performance snapshot (2026-07-28)

Metric	Value
set01–04 parse	~94.8% OK
set01–04 Runtime (projected after fail re-runs)	**~89.8%** OK (≈2224/2477)
PARSE_FAIL residual	~118 files — almost all truncated/non-Pine
`ta_sma` bar loop (≈3.2k bars)	**~9.5×** vs full recompute
`ta_atr` bar loop	**~10×**
macd+atr+rsi+sma combo	**~8.5×**

Plans: `.opencode/plans/2026-07-28-runtime-performance.md`. Skill: `.grok/skills/pynescript-perf/`. Golden: `tests/test_ta_incremental.py`.

Internals

Path	Role
`docs/missing_features.md`	Living gap analysis
`tests/test_v6_features.py`	v6 feature coverage
`tests/test_pine_surface_gaps.py`	Surface gap probes
`tests/test_ta_incremental.py`	Incremental TA ≡ full recompute
`docs/pine_v6_full_surface_inventory.md`	Full name-level inventory
`resource/*.g4` + builder	Grammar-level feature work
`.../technical_submodules/core.py`	`_sma_inc_update`, `_macd_inc_update`, `_atr_inc_update`, …

Invariants & edge cases

1. **Closing a syntax gap requires grammar + builder + unparser + tests**, not only an evaluator stub.
2. **Mock data can green tests while production feeds differ** — declare data source explicitly.
3. **"0 missing vs reference list" ≠ identical semantics for every edge case.**
4. **Perf changes must keep bar-by-bar semantics** — no whole-script vectorization; prefer golden tests vs current oracle.
5. Update this annex when `missing_features.md` changes major status lines.

Worked examples

Detect multiline string support

```
from pynescript.ast.helper import parse, unparse
src = '''//@version=6
indicator("m")
s = """a
b"""
plot(1)
'''
tree = parse(src)
assert '"""' in unparse(tree) or "'''" in unparse(tree)
```

Footprint mock call

Prefer tests in `tests/test_v6_features.py` as executable specification for `request.footprint` methods.

Incremental TA parity smoke

```
.venv/bin/python -m pytest tests/test_ta_incremental.py -q
# Disable hot path if needed:
PYNE_TA_INCREMENTAL=0 python -c "from backend.runtime import Runtime; ..."
```

Failure modes

Symptom	Likely gap class
Lexer error on triple quotes	Stale generated lexer (should be fixed on main)
Import member missing	Library export registration
Empty footprint rows	Mock seed / data_feed not configured
Risk not blocking entry	Old runtime path; ensure risk kwargs applied
Runtime TIMEOUT on method-heavy scripts	Missing `_pine_defs_locked` on host (fixed 2026-07-28)
Corpus PARSE_FAIL mid-block	Truncated scrape — not a grammar hole

See also

- Implementation status
- Pine v6 surface
- Roadmap
- Compatibility
- Technical analysis (`ta.*`)

PINE V6 SURFACE

title: "Pine v6 surface inventory" description: "Summary of the full Pine Script™ v6 surface inventory — dispatch counts, namespaces, status schema — with pointer to the large source document."

PINE V6 SURFACE INVENTORY

Abstract

The **full surface inventory** enumerates every registered evaluator builtin, major series/variable, language construct, and known gap against Pine Script™ v6. It is deliberately large (tens of thousands of tokens when fully expanded). **This page is the navigable summary**; the exhaustive tables live in-repo at:

`docs/pine_v6_full_surface_inventory.md`

Generated snapshot referenced here: **2026-07-25**, from live `NodeLiteralEvaluator` dispatch, `builtin_metadata.json`, and design docs.

Do **not** paste the entire inventory into product docs — regenerate and link.

Conceptual model

```
flowchart TB
  subgraph Parse
    SRC[Pine source] --> LEX[Lexer]
    LEX --> PAR[Parser]
    PAR --> BLD[Builder]
    BLD --> AST[ASDL AST]
  end
  subgraph Eval
    AST --> VIS[NodeLiteralEvaluator]
    VIS --> BI[Builtin dispatch map]
    BI --> NS[ta / strategy / array / …]
  end
  BI --> INV[Inventory rows]
```

Interface surface

Status schema

Status	Definition
✅ implemented	Handler/parser path exists and is usable
🔄 partial	Mock/stub, incomplete semantics, or metadata-only
❌ missing	Expected on TV v6; not resolving
⚙️ by design	Intentionally not full TV platform
⬜ N/A	Editor/UI-only

Each inventory row also carries: **Name**, **Namespace**, **Kind**, **Metadata** (LSP JSON present?), **Source** (e.g. `dispatch`), **Notes**.

Summary counts (2026-07-25)

Metric	Count
Dispatch builtins (callable)	**640**
Dispatch partial-heuristic (stub/mock docstring)	8
Top-level namespaces	60
Official TV v6 function reference list	434 symbols — **0 missing** in dispatch

Top namespaces by dispatch keys

Namespace	Count
`ta`	159
`matrix`	74
`strategy`	70
`array`	56
`box`	31
`label`	28
`table`	26
`math`	24
`line`	22
`str`	19
`input`	14
`map`	11
`request`	11
`footprint`	9
`ticker`	9
(remaining namespaces)	smaller

Kinds covered

`function` · `series/var` · `constant` · `declaration` · `control` · `operator` · `type system` · `literal` · `semantics` · `export` · `import`

Internals

Path	Role
`docs/pine_v6_full_surface_inventory.md`	Full tables + architecture diagrams
`scripts/regenerate_inventory_summary.py`	Regeneration helper
`scripts/generate_builtin_metadata.py`	LSP metadata from live surface
Evaluator `_build_builtin_map()`	Source of dispatch truth

How to use the full file

1. Open `docs/pine_v6_full_surface_inventory.md` in the repo or Sphinx/site mirror.
2. Search by namespace (`## ta`, `strategy`, …).
3. Trust **dispatch** source rows for “is there a callable?”; read **Notes** for mock/feed caveats.
4. After large builtin PRs, regenerate summary counts before quoting them publicly.

Invariants & edge cases

1. **Inventory size is a feature** — summarization must not drop 🔄 rows when claiming coverage.
2. **Metadata presence ≠ semantic completeness.**
3. **Partial heuristic count (8)** is docstring-based — real partials may be higher.
4. **Language constructs** (control/export) may be parser-complete without every runtime edge.

Worked examples

Quote coverage honestly

> "Against the public Pine v6 function reference (434 symbols), pynescript registered **0 missing** dispatch entries as of 2026-07-25; 640 total callables across 60 namespaces. See `docs/pine_v6_full_surface_inventory.md`."

Programmatic check (sketch)

```
# regenerate after evaluator changes
python scripts/regenerate_inventory_summary.py
```

Failure modes

Mistake	Consequence
Pasting 134KB inventory into Mintlify page	Unreadable docs; broken UX
Citing counts without date	Stale marketing
Equating dispatch hit with TV gold	False precision claims

See also

- Implementation status
- Missing features
- Compatibility
- Runtime builtins

NUMERICAL VALIDATION

title: "Numerical validation" description: "Methodology and error bounds for PYNE technical and math builtins versus reference Pine Script™ implementations."

NUMERICAL VALIDATION

Abstract

PYNE's technical and mathematical builtins are validated for **numerical agreement** with reference Pine Script™ implementations under IEEE 754 realities. The long-form report (`docs/numerical_validation_report.md` , v1.0, Nov 2025) documents methodology, per-family error tables, and edge cases.

Headline (from that report): max observed relative error **0.0022%** (ADX, extreme volatility); average error **< 0.0005%**; deterministic ops (SMA, OBV, discrete math) often **exact**.

Treat percentages as **report-era measurements**, not eternal SLAs — re-validate after changing smoothers or bar-mode semantics.

Conceptual model

```
sequenceDiagram
  participant D as OHLCV dataset
  participant TV as Reference Pine
  participant PY as PYNE evaluator
  participant C as Comparator
  D->>TV: same bars
  D->>PY: same bars
  TV->>C: series out
  PY->>C: series out
  C->>C: abs / rel error stats
```

Interface surface

Acceptable thresholds (report)

Band	Relative error	Verdict
Excellent	< 0.001%	Pass
Good	< 0.01%	Pass
Acceptable	< 0.1%	Review
Unacceptable	≥ 0.1%	Fail

Metric definitions

- **Absolute error:** `|ref − pyne|`
- **Relative error:** `|ref − pyne| / |ref| × 100%` (guard zero refs)
- **Max / mean / std** over bars and scenarios

Category highlights

Family	Notes
Moving averages	SMA exact; EMA/WMA/… tiny FP error
Oscillators	RSI/stoch slightly higher smoothing error still ≪ 0.01%
Trend	ADX highest in report (0.0022% max)
Volume	OBV exact cumulative; MFI inherits typical-price FP
Stats	percentrank exact; variance/stdev excellent
Math	Discrete exact; transcendentals ~1e-15 class

Error distribution (report aggregate)

Error range	Share (approx.)
Exact 0	~38%
< 0.0001%	~46%
0.0001–0.001%	~14%
0.001–0.01%	~2%
≥ 0.1%	0%

Internals

Path	Role
`docs/numerical_validation_report.md`	Full tables + bias analysis
`tests/test_ta_indicators_*.py`	Executable TA tests
`tests/test_indicators.py` / builtins tests	Additional numeric guards
Bar-mode vs list-mode (`_pine_bar_mode`)	Scalar-per-bar vs full series — compare like with like

Methodology (report)

1. Generate synthetic 1k-bar OHLCV across regimes (trend, range, vol, gaps)
2. Validate with real histories (equities, crypto, FX samples)
3. Export reference outputs; run PYNE; element-wise compare
4. Search systematic bias; inspect tails

Invariants & edge cases

1. **na propagation** must match Pine truthiness — numerical compare should mask `na` pairs.
2. **Bar-mode scalars** vs full-series list mode can look like “bugs” if misaligned.
3. **Seeded mocks** (`request.seed`) stabilize stochastic feeds for regression.
4. **Recursive smoothers** accumulate FP differently across languages; bounds > bits.
5. **Extreme prices** (near-zero, huge) tested; still within excellent band in report.

Worked examples

Relative error helper

```
def rel_err(a: float, b: float) -> float:
    if a == 0 and b == 0:
        return 0.0
    denom = abs(a) if a != 0 else abs(b)
    return abs(a - b) / denom * 100.0
```

Run TA tests

```
pytest tests/test_ta_indicators_1.py tests/test_ta_indicators_2.py -q
```

Failure modes

Symptom	Investigation
Sudden ADX drift	Check Wilder smoothing / seed bars
SMA not exact	Off-by-one window or float input casts
Good unit tests, bad TV export compare	Timezone/session alignment of bars
Only bar 0 differs	Warm-up / `na` initialization

See also

- Compatibility
- Runtime technical builtins
- Implementation status

ROADMAP

title: "Roadmap" description: "PYNE forward plan — residual host parity, corpus tail, warm compile, and optional fidelity work."

ROADMAP

Abstract

The roadmap is a **living prioritization document**, not a contract. Source: `docs/ROADMAP.md` (updated **2026-08-01**). Historically “remaining” items — Jupyter, linter, data providers, strategy events, Pro API, LSP core, deep strategy runtime, drawing GC, **alert engine + L2 webhooks** — are **done**. What remains: **dual-host package-level Runtime unify**, **corpus execution tail**, **warm-compile productization**, residual **TA/series perf**, and optional **TV-oracle re-baselines**.

Conceptual model

```
flowchart LR
  DONE[Core parser + evaluator] --> A[Phase A performance]
  DONE --> B[Phase B DX polish]
  DONE --> C[Phase C integration depth]
  DONE --> D[Phase D transforms / scale]
  PW[pine-worker maturity] --> HOOX[HOOX trade mesh]
```

Interface surface

Status snapshot

Component	Status
Parser ANTLR4	✅
Evaluator	✅ incl. var/varip / reassign
Builtins / TA / collections	✅ broad
Strategy events + parity	✅
pine-worker (TS + converter)	✅ colocated tool (port ongoing)
Drawing/input/request	✅ (plots registry; data mock/feed)
Linter / Jupyter / data providers	✅
LSP advanced features	✅ core set
Numba compile path	✅ MVP
Full TV platform identity	⚙️ out of scope

Completed themes (do not re-plan as greenfield)

- Error message / typing hygiene
- `%%pinescript` Jupyter magic + sample data helpers
- `pynescript lint`
- Yahoo / AlphaVantage / CCXT + realtime datafeed wiring
- StrategyEvent emission + parity fixtures
- pine-worker extra tool + Python→TS converter

Remaining high-value work

ID	Item	Pri
H1	Dual-host Runtime (host surface + alerts ✅; package-level unify open)	P1 ⚙️
H2	Warm-compile product path (SLOs / prewarm / IR cache)	P1
C1	Corpus TIMEOUT / RUN_FAIL residual	P1
T1 / T2	Series caps + residual incremental TA	P2
F1 / F2	Optional TV-oracle re-baselines (ATR, pending fills)	P2
L1	v5↔v6 converter maturity	P3
L2	**Alert webhooks** (Pro API + pyne-worker)	**P3** ✅
L3	pine-worker (TS) full builtin parity	P3

Longer horizon (not next): ML wrappers, automatic refactor, parallel/distributed eval experiments.

Priority recommendation

Horizon	Focus
Short	H1 package unify residual, C1 corpus tail, H2 warm compile
Medium	T1/T2 series + residual TA; optional F1/F2 goldens
Long	L1 converter maturity; L3 TS pine-worker parity

Internals

Path	Role
`docs/ROADMAP.md`	Canonical roadmap prose
`docs/PROGRESS_REPORT.md`	Historical completion narrative
`docs/COMPILER_PLAN.md`	Compile/numba plan
`.opencode/plans/`	Engineering plans (strategy events, consolidation)
`pine-worker/`	TS port track

Invariants & edge cases

1. **Roadmap dates lag code** — always diff against `missing_features.md` and tests.
2. **Sister repos** (pyne-worker, hoox-setup, pine-worker packaging) may absorb “integration” items.
3. **AXIS is a separate product tree** (`docs/axis`) — chart UX roadmap is not PYNE-core.
4. Avoid reintroducing deleted backups (`builder.py.bak` , etc.) as “plan items.”

Worked examples

Track a roadmap item with a test

```
# pick a gap, write failing test first, implement, update missing_features.md
pytest tests/test_v6_features.py -k footprint -v
```

pine-worker maturity gate

```
cd pine-worker && bun test && bun run typecheck
# future: bun run parity once harness lands
```

Failure modes

Anti-pattern	Why it hurts
Planning features already ✅	Duplicated work
Roadmap without tests	Unverifiable “done”
Ignoring by-design limits	Infinite platform chase

See also

- Missing features
- Implementation status
- pine-worker
- Contributing

CONTRIBUTING

title: "Contributing" description: "Hard constraints, workflow, and style rules for contributors and agents working on PYNE."

CONTRIBUTING

Abstract

Contributions to PYNE are welcome when they respect the repo's **mechanical invariants**: generated code boundaries, src-layout packaging, dual console scripts, test corpus costs, and encryption secrets. This page distills `CONTRIBUTING.md` and `AGENTS.md` into an operational contract for humans and agents.

Conceptual model

```
flowchart LR
  FORK[Fork + branch] --> INST[make install / hatch]
  INST --> TEST[tests green]
  TEST --> EDIT[change code]
  EDIT --> LINT[ruff + mypy]
  LINT --> PR[Pull request]
```

Interface surface

Baseline workflow

1. Fork and branch from `main`
2. Install: `make install` or hatch envs
3. Run tests: `make test` / `hatch run test:test`
4. Implement with tests (prefer TDD for non-trivial behavior)
5. Lint/format: `make lint` / `make fmt` or `hatch run lint:style` + `lint:typing`
6. Open a PR with rationale + test evidence

Official short form also lives in root `CONTRIBUTING.md` (points at Sphinx docs historically).

Everyday commands

Goal	Command
Editable install + LSP	`make install`
Full tests	`make test`
LSP only	`make test-lsp`
Backend only	`make test-backend`
Lint / format	`make lint` / `make fmt`
Fast build sanity	`make build-check`
API	`make run`
LSP process	`make run-lsp`

Hard constraints

Grammar & generated code

- **Edit only** `src/pynescript/ast/grammar/antlr4/resource/*.g4` for grammar work
- **Never hand-edit** `…/antlr4/generated/` or `…/asdl/generated/`
- Regenerate via hatch `lint:gen-parser` / project scripts; see grammar guides under `.opencode/context/`

No stale backups

Do not recreate removed backups such as `builder.py.bak` or `evaluator/builtins/technical_refactored.py`. Live `builder.py` and `technical.py` are sole sources of truth.

Future annotations

Every new Python file must start with:

```
from __future__ import annotations
```

Enforced by ruff isort `required-imports`.

Console scripts

Script	Role
`pynescript`	Click CLI
`pynescript-lsp`	Language Server (pygls)

Do not conflate entrypoints in docs, packaging, or process managers.

Test corpus fixture

`tests/conftest.py` parametrizes `pinescript_filepath` over **every** `tests/data/builtin_scripts/*.pine`. A new test using that fixture may run hundreds of cases. Narrow with `--example-scripts-dir=...` when iterating.

`tests/data/library/` is reference material, not auto-parametrized.

Builtin metadata

`builtin_metadata.json` is **generated from code**. After adding builtins:

```
python scripts/generate_builtin_metadata.py
```

Do not hand-maintain the catalog long-term.

Secrets & crypto

- `scripts/build/.metadata.key` is **gitignored**
- CI must supply stable `CRYPTO_KEY` from `METADATA_KEY` / `_METADATA_KEY` for reproducible `.enc` blobs
- Never commit Fernet keys or API admin tokens

Internals (where to change what)

Want to...	Look at
Parse / unparse	`src/pynescript/ast/helper.py`
Add builtin	`src/pynescript/ast/evaluator/builtins/<ns>.py` + metadata generator
LSP feature	`src/pynescript/langserver/features/` + `server.py`
CLI subcommand	`src/pynescript/__main__.py`
Grammar	`resource/*.g4` then regenerate
Pro API route	`backend/api/preview.py`, `backend/app.py`
TS port	`pine-worker/`

Style

- **Ruff** line length 120; broad rule set in `pyproject.toml`
- **Black** target `py310` (via toolchain config)
- **Mypy** strict-ish; exemptions for generated grammar, `evaluator.builtins.*`, tests
- Voice for docs: academic nerdy-cool per `docs/WRITING.md` — no empty marketing adjectives

pine-worker contributions

- Bun for install/test/typecheck
- Python remains oracle; update parity fixtures via generator
- Converter stubs are starting points only

Failure modes for contributors

Mistake	Fallout
Editing generated ANTLR/ASDL	Overwritten / review hell
Skipping corpus cost awareness	10-minute “unit” tests
Hand-editing metadata JSON	Drift from dispatch
Committing `.metadata.key`	Secret leak; rotate immediately
PR without lint	CI red on ruff/mypy

See also

- Local development
- CI
- pine-worker
- Roadmap
- Root `AGENTS.md`, `CONTRIBUTING.md`, `CODE_OF_CONDUCT.md`

OVERVIEW

title: "pine-worker" description: "TypeScript Pine evaluator port colocated in pynescript — architecture, status, and relationship to the Python oracle."

PINE-WORKER

Abstract

pine-worker is a TypeScript implementation track for the Pine Script™ evaluator (AST types, visitor dispatch, series history, and eventually builtins + strategy events). It lives at `pine-worker/` inside the **pynescript** repository as an **extra tool**, not a replacement for the Python package.

Colocation goals:

- Share **parity fixtures** under `tests/fixtures/parity/`

- Keep the ANTLR **resource grammar** as single syntactic truth (TS parser target can be generated later)
- Allow cross-language porting in one checkout

Python (`pynescript` + `backend/`) remains the **semantic oracle**.

Conceptual model

```
flowchart TB
  G4[resource/*.g4 grammar] --> PY[Python ANTLR parser]
  G4 -.-> TS[Future ANTLR TS target]
  PY --> ASTPY[ASDL AST]
  ASTPY --> EVALPY[Python evaluator]
  EVALPY --> FIX[parity JSON fixtures]
  ZOD[Zod AST schemas] --> EVALTS[TS Evaluator]
  EVALTS --> FIX
  CONV[convert-python-to-ts.py] --> STUBS[TS builtin stubs]
  EVALPY --> CONV
```

Interface surface

Quick start (Bun)

```
cd pine-worker
bun install
bun test
bun run typecheck
```

`package.json` scripts:

Script	Action
`test`	`bun test`
`test:watch`	watch mode
`typecheck`	`tsc --noEmit`
`parity`	reserved — harness not fully wired

Dependencies: **Zod** (AST schemas), **TypeScript**; runtime assumes **Bun** for tests.

Current skeleton

Area	State
AST Zod schemas + types	Partial (mirrors Python ASDL intent)
Base evaluator + visitor dispatch	Partial
`PineSeries` historical access	Partial
Core NA / loop signal semantics	Partial
Full builtins	In progress
ANTLR TS parser	Not complete
Strategy event emission	Plan-driven (see opencode plans)
Parity harness `test/parity/`	Planned

Relationship to HOOX

When mature, pine-worker may be vendored/submoduled into **hoox-setup** as a Cloudflare Worker bound to trade-worker. Until then it develops here. Sister: **pyne-worker** (Python edge) is a different repo.

Internals

Path	Role
`pine-worker/src/ast/types.ts`	AST typing / Zod
`pine-worker/src/evaluator/evaluator.ts`	Visitor evaluator
`pine-worker/src/evaluator/series.ts`	Series model
`pine-worker/src/evaluator/types.ts`	NA, signals, registries
`pine-worker/test/*.test.ts`	Unit tests
`pine-worker/scripts/convert-python-to-ts.py`	Porting aid
`tests/fixtures/parity/`	Shared oracle fixtures
`.opencode/plans/*pine-worker*`	Strategy-events plans

Invariants & edge cases

1. **Python wins ties.** If TS and Python disagree, fix TS (unless Python is proven wrong — then fix both + fixtures).
2. **NA is a sentinel**, not JS `null` alone — truthiness differs from ordinary JS symbols.
3. **Do not fork grammar** ad hoc in TS; prefer shared resource `.g4`.
4. **Converter emits stubs, not proofs** — manual port + parity required.
5. License follows parent pynescript project.

Worked examples

Minimal evaluator unit (conceptual)

TS tests construct literal AST nodes (`Literal` , `Assign` , `Script`) and assert visitor results — see `pine-worker/test/evaluator.test.ts` .

Regenerate Python fixtures

```
python tests/fixtures/parity/generate_fixtures.py
```

Failure modes

Symptom	Cause	Fix
`bun test` fails on NA	JS truthiness	Use Pine truth helpers
Parity script no-ops	Harness not wired	Implement `test/parity` per plan
Drift from Python builtins	Manual port error	Re-run converter diff + fixtures

See also

- Converter
- Testing
- Runtime events
- Alerts (edge capture + L2 webhooks; see also **pyne-worker** Python edge host)
- Roadmap

CONVERTER

title: "Python → TypeScript converter" description: "Skeleton generator that turns pynescript builtin modules into TS stubs for manual semantic porting."

PYTHON → TYPESCRIPT CONVERTER

Abstract

`pine-worker/scripts/convert-python-to-ts.py` is a **porting accelerator**, not a verified transpiler. It reads Python builtin modules (AST of the Python source), emits TypeScript scaffolding with imports, function signatures, best-effort JSDoc from docstrings, and `TODO` bodies.

The human (or follow-on agent) still implements semantics to match the Python oracle and locks them with parity tests.

Conceptual model

```
flowchart LR
  PY[builtins/numeric.py] --> AST[ast.parse]
  AST --> MAP[type annotation map]
  AST --> DOC[docstring extract]
  MAP --> TS[numeric.ts skeleton]
  DOC --> TS
  TS --> HUMAN[manual body port]
  HUMAN --> PARITY[parity fixtures]
```

Interface surface

CLI

```
python pine-worker/scripts/convert-python-to-ts.py \
  src/pynescript/ast/evaluator/builtins/numeric.py

# redirect
python pine-worker/scripts/convert-python-to-ts.py \
  src/pynescript/ast/evaluator/builtins/numeric.py \
  > pine-worker/src/evaluator/builtins/numeric.ts

# namespace hint
python pine-worker/scripts/convert-python-to-ts.py \
  --namespace technical \
  --module src/pynescript/ast/evaluator/builtins/technical.py
```

Emitted properties

Feature	Behavior
Function signatures	Args + crude TS types from annotations
Docstrings	JSDoc blocks when present
Bodies	TODO / stubs — **not** executable ports
Types mapping	`int` / `float` → `number`, `str` → `string`, containers → broad `any[]` / `Record`

Explicit non-goals

- Full semantic translation of Python control flow
- NumPy / Pine series awareness
- Automatic parity proof
- Keeping output green without edits

Internals

Path	Role
`pine-worker/scripts/convert-python-to-ts.py`	Converter implementation
`src/pynescript/ast/evaluator/builtins/*.py`	Preferred inputs
`pine-worker/src/evaluator/`	Destination tree for ports

Type mapping (simplified)

The script applies regex-ish substitutions for common typing names (`Optional` , `List` , `Dict` , …) and strips generics crudely. Expect to **hand-fix** types toward Zod-validated AST nodes and `PineSeries` .

Invariants & edge cases

1. **Re-running clobbers manual bodies** if you redirect over a finished file — convert to a new path or diff carefully.
2. **Mixin methods / dynamic dispatch** in Python may not map 1:1 to TS functions.
3. **Default args and** `*args` need manual TS redesign.
4. Prefer converting **one namespace at a time** to keep reviewable diffs.

Worked examples

Port math helpers workflow

```
mkdir -p pine-worker/src/evaluator/builtins
python pine-worker/scripts/convert-python-to-ts.py \
  src/pynescript/ast/evaluator/builtins/numeric.py \
  > pine-worker/src/evaluator/builtins/numeric.ts
# implement TODOs
cd pine-worker && bun test
```

Compare against oracle

```
python tests/fixtures/parity/generate_fixtures.py
# add TS assertions consuming tests/fixtures/parity/json/*.json
```

Failure modes

Symptom	Fix
Empty output	Path wrong / no functions found
`any` everywhere	Annotations missing in Python source
Runtime divergence	Implement NA rules; don’t trust stubs
Import cycles in TS	Align module graph with evaluator design

See also

- pine-worker overview
- Testing & parity
- Runtime builtins

TESTING

title: "pine-worker testing" description: "Bun unit tests, shared parity fixtures, and the path to cross-implementation strategy event validation."

PINE-WORKER TESTING

Abstract

Testing for pine-worker has two layers:

1. **Local Bun unit tests** — AST/evaluator/series behavior in pure TS
2. **Cross-language parity** — Python oracle generates JSON fixtures; TS must match

Until the parity harness is fully wired (`bun run parity` is a placeholder), treat Python fixtures as the contract and unit tests as constructive proofs of the port's foundations.

Conceptual model

```
flowchart TB
  subgraph Unit
    T1[ast.test.ts]
    T2[evaluator.test.ts]
    T3[series.test.ts]
  end
  subgraph Parity
    PINE[parity/pine/*.pine] --> GEN[generate_fixtures.py]
    GEN --> JSON[parity/json/*.json]
    JSON --> PYT[Python tests]
    JSON --> TST[TS harness future]
  end
  Unit --> BUN[bun test]
```

Interface surface

Commands

```
cd pine-worker
bun install
bun test
bun test --watch
bun run typecheck
bun run parity   # prints note until harness lands
```

Unit test files

File	Focus
`test/ast.test.ts`	AST schema / types
`test/evaluator.test.ts`	Visitor eval, assign, literals
`test/series.test.ts`	Historical series indexing

Tests use `bun:test` (`describe` / `test` / `expect`).

Parity fixtures (shared with Python)

Path	Content
`tests/fixtures/parity/pine/*.pine`	Minimal strategy scripts
`tests/fixtures/parity/json/*.json`	Expected events / series snapshots
`tests/fixtures/parity/generate_fixtures.py`	Oracle generator
`tests/fixtures/parity/ohlcv.py`	Shared bars

Example scenarios: entry long/short, close, close_all, exit stop/limit, cancel, no-action, var counters.

Python-side consumers

Parity is also exercised from the main suite (e.g. strategy event tests). Regenerating fixtures:

```
python tests/fixtures/parity/generate_fixtures.py
```

Internals

Evaluator test style

Tests **hand-build AST node objects** (plain dictionaries with `type` discriminators) rather than parsing Pine text — because the TS parser target is incomplete. That decouples evaluator progress from lexer work.

NA and control-flow signals

`evaluator/types.ts` defines `NA`, `BreakSignal`, `ContinueSignal`, `ReturnSignal`. Unit tests should cover:

- arithmetic with NA → NA
- comparisons with NA → false (Pine rule)
- truthiness where NA is falsy (unlike some JS values)

Future parity harness

Intended location: `pine-worker/test/parity/` with `bun test test/parity --timeout 30000`. Design goal: load each JSON fixture, evaluate corresponding AST/script path, assert event lists match oracle.

Invariants & edge cases

1. **Never edit JSON fixtures by hand** to make TS pass — regenerate from Python or fix Python if oracle is wrong.
2. **OHLCV must match** between generators and consumers (`ohlcv.py`).
3. **Timeouts:** strategy loops can be heavy; parity script reserves 30s.
4. **Typecheck is mandatory** before claiming port milestones (`tsc --noEmit`).

Worked examples

Run a single unit file

```
cd pine-worker
bun test test/series.test.ts
```

Add a unit case for assignment

Follow patterns in `evaluator.test.ts`: build `Script` → `Assign` → `Identifier` load → `expect`.

Oracle refresh after Python strategy change

```
python tests/fixtures/parity/generate_fixtures.py
git diff tests/fixtures/parity/json/
pytest tests/test_strategy_events.py -q
```

Failure modes

Symptom	Cause	Fix
Fixture mismatch after Python fix	Stale JSON	Regenerate
TS passes units, fails parity	Missing strategy builtins	Port + converter
Flaky floats	Compare with tolerances	Mirror Python assert style
`parity` script always “not wired”	Expected until harness lands	Implement under `test/parity/`

See also

- pine-worker overview
- Converter
- Runtime events
- Compatibility