HOOX · PYNE · DOCUMENTATION

API CLIENTS · BACKEND OPERATORS

# PRO API MANUAL

Flask Pro API, auth, /run contract, preview, backtest

Open-source Pine Script evaluation framework

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

# CONTENTS

# OVERVIEW



**title: "Pro API" description: "Flask Pro API for Pine evaluation — /run, preview, backtest, API keys, and the shared evaluate contract."**

# PRO API

The PYNE **Pro API** is a Flask HTTP service that runs Pine scripts over caller-supplied OHLCV, returns plots / series / strategy events / drawings / **alerts**, optional **L2 alert webhooks**, and gates thumbnail + backtest routes behind API keys. It is intentionally **not** the language server: different auth, scaling, and failure domains.

## Abstract

Where the LSP answers *static* editor questions, the Pro API answers *dynamic* evaluate questions:

```
POST JSON { script, data: OHLCV[], … }
  → backend.runtime.Runtime.run
  → { plots, series, plot_meta, events, drawings, alerts, script_id, run_id, … }
```

Free-tier evaluate (`POST /run`, `POST /run/batch`) validates bodies with hand-rolled schemas, enforces a 5 MB body cap, and restricts CORS. Pro routes under `/preview/*` and `/backtest/*` use `track_usage` (Bearer / ApiKey / query key). Chart PNGs are matplotlib Agg renders encoded as base64.

The same **evaluate contract** is mirrored by edge workers (pyne-worker / pine-worker) so AXIS and HOOX can swap hosts without redesigning payloads — see contract.

## Conceptual model

```
flowchart TB
  CLIENT[AXIS / CLI / curl] --> FLASK[backend.app Flask]
  FLASK --> AUTH[middleware auth + schemas]
  FLASK --> RUN["/run · /run/batch"]
  FLASK --> PREV["/preview/*"]
  FLASK --> BT["/backtest/*"]
  RUN --> RT[backend.runtime.Runtime]
  PREV --> CHART[chart_renderer]
  BT --> BACK[services.backtest]
  RT --> EVAL[CustomEvaluator + PineSeries]
  RT --> AST[parse ASDL]
```

## Interface surface

| Method | Path | Auth | Role |
|---|---|---|---|
| `GET` | `/` | none | Health + endpoint map |
| `POST` | `/run` | none (free) | Single-script evaluate |
| `POST` | `/run/batch` | none (free) | ≤8 scripts, shared OHLCV |
| `POST` | `/preview/chart` | API key + usage | Line / OHLCV PNG |
| `POST` | `/preview/indicator` | API key + usage | Expression series PNG |
| `POST` | `/backtest/quick` | API key + usage | Quick strategy metrics + equity PNG |
| `POST` | `/auth/create_key` | admin decorator | Mint `pyn_…` key |
| `GET` | `/auth/usage` | API key | Usage counters |
| `POST` | `/auth/validate` | none (body key) | Validate without consuming quota |

Local runner: `make run` → `python -m backend.app` (default `127.0.0.1:5002`).

## Tracks in this tab

1. App lifecycle — Flask app, CORS, limits, blueprints
2. Auth and keys — tiers, stores, decorators
3. Run · Preview · Backtest
4. Chart renderer

5. Runtime bridge — `Runtime` / series / evaluator glue
6. Contract — shared evaluate schema across Flask and workers

## Internals (map)

| Path | Role |
| --- | --- |
| `backend/app.py` | App factory surface, `/run*`, `/auth*`, CORS |
| `backend/api/preview.py` | Preview + backtest blueprints |
| `backend/middleware/*` | Auth, schemas, optional Redis/SQLite stores |
| `backend/runtime.py` | Bar-loop evaluate façade |
| `backend/evaluator.py` | `CustomEvaluator` bridge |
| `backend/series.py` | `PineSeries` history deque |
| `backend/services/*` | Charts + backtest simulation |

## See also

- Evaluate scripts (end user)
- Pro API usage guide
- Runtime hub
- LSP hub — editor path, not HTTP

# APP LIFECYCLE



**title: "App Lifecycle" description: "Flask application setup: MAX_CONTENT_LENGTH, CORS policy, blueprints, error handlers, and process entry."**

# APP LIFECYCLE

## Abstract

`backend/app.py` constructs a single global `Flask` app, applies security-minded defaults (body size, CORS allowlist), registers free and Pro routes, and exposes a module `__main__` for local development. There is no multi-app factory pattern today — import `app` for WSGI (gunicorn) or run the module for the built-in server.

## Conceptual model

```
flowchart TB
  BOOT[import backend.app] --> CFG[MAX_CONTENT_LENGTH + CORS]
  CFG --> ROUTES[register / · /run · /auth]
  ROUTES --> BP[register preview_bp + backtest_bp]
  BP --> ERR[404 / 500 JSON handlers]
  ERR --> SERVE{entry}
  SERVE -->|gunicorn| WSGI[app:app]
  SERVE -->|python -m backend.app| DEV[HOST/PORT env]
```

## Interface surface

### Process entry

```
make run
# equivalent
python -m backend.app
```

Environment:

| Variable | Default | Role |
|---|---|---|
| `HOST` | `127.0.0.1` | Bind address (localhost-first for dev) |
| `PORT` | `5002` | Listen port |
| `ALLOWED_ORIGINS` | `https://pynescript.ai`, `https://app.pynescript.ai`, localhost regex | CORS origins (comma-separated; regex allowed) |
| `API_KEY_STORE` | `/root/pynescript/data/api_keys.json` | JSON key store path (file-backed default) |
| `ADMIN_TOKEN` | unset | Documented for admin create_key (see auth) |

`debug=False` always on the dev runner.

### Health

`GET /` → JSON:

```
{
  "status": "healthy",
  "service": "pynescript-pro-api",
  "version": "1.0.0",
  "timestamp": 0,
  "endpoints": { "…": "…" }
}
```

### Hard limits

- **`MAX_CONTENT_LENGTH = 5 * 1024 * 1024`** — reject oversized bodies before JSON parse fills memory (audit 2026-07-05 / S1).

- CORS methods: `GET`, `POST` only.
- Allow headers: `Content-Type`, `Authorization`, `X-Admin-Token`.
- `supports_credentials=False`.

Localhost regex used in the default origin list:

```
^https?://(?:localhost|127\.0\.0\.1)(?::\d+)?$
```

Same-origin / no `Origin` header (curl, server-to-server) remains usable under flask-cors behavior.

### Blueprints

```
app.register_blueprint(preview_bp)   # url_prefix=/preview
app.register_blueprint(backtest_bp)  # url_prefix=/backtest
```

Defined in `backend/api/preview.py`.

### Error handlers

| Code | Body code | Message |
|---|---|---|
| 404 | `NOT_FOUND` | `Endpoint {path} not found.` |
| 500 | `INTERNAL_ERROR` | `Internal server error.` |

Both return JSON with `status: "error"`.

## Internals

| Path | Role |
|---|---|
| `backend/app.py` | App object, free routes, auth routes |
| `backend/api/preview.py` | Pro blueprints |
| `backend/middleware/schemas.py` | Request validation for `/run*` and auth |
| `backend/requirements.txt` | Flask, flask-cors, numpy, matplotlib, ... |

Recommended production: gunicorn/uvicorn-class WSGI with multiple workers; for multi-worker key consistency prefer SQLite/Redis stores over the default in-memory/file singleton assumptions — see auth.

## Invariants and edge cases

1. **Import side effects:** creating `app` configures CORS immediately from env.
2. **Body too large** → Flask 413 before route logic (not custom JSON).
3. **Unknown fields** on schema-validated routes → `UNKNOWN_FIELDS` 400 (`/run`, auth); preview routes currently use looser `get_json()` (schemas exist but are not always applied in handlers).
4. **Dev bind default is loopback** — set `HOST=0.0.0.0` only when intentional.

## Worked example — smoke test

```
curl -s http://127.0.0.1:5002/ | jq .status
# "healthy"
```

## Failure modes

| Symptom | Cause |
| --- | --- |
| Browser CORS errors | Origin not in `ALLOWED_ORIGINS` |
| 413 on large OHLCV | Exceeded 5 MB — downsample or paginate |
| Import errors for matplotlib | Missing backend deps — `pip install -r backend/requirements.txt` |

## See also

- Auth and keys
- Run endpoint
- Docker / CI

# AUTH AND KEYS



**title: "Auth and Keys" description: "API key model, tiers, require_api_key / track_usage decorators, JSON/SQLite/Redis stores, and admin create_key."**

# AUTH AND KEYS

## Abstract

Pro routes authenticate with a **raw API key** presented as `Authorization: Bearer …`, `Authorization: ApiKey …`, or `?api_key=`. Keys are minted as `pyn_` + url-safe secret, stored with a short `key_id` and SHA-256 hash metadata, and tracked against **monthly-ish call limits** by tier. Free `/run` does not require a key.

## Conceptual model

```
flowchart LR
  REQ[HTTP request] --> HDR{Authorization / query}
  HDR --> VAL[store.validate_key]
  VAL -->|miss| U401[401 UNAUTHORIZED]
  VAL -->|hit + over limit| U429[429 RATE_LIMITED]
  VAL -->|ok| G[g.api_key]
  G --> H[handler]
  H --> TRACK[increment_calls on success]
```

## Interface surface

### Tiers ( `_TIER_LIMITS` )

| Tier | `calls_limit` |
|---|---|
| `free` | `0` (treated as unlimited in `is_rate_limited` — limit 0 means “no cap”) |
| `hobby` | 5 000 |
| `pro` | 50 000 |
| `team` | 200 000 |
| `enterprise` | `inf` |

`calls_remaining()` returns `inf` when limit is 0.

### Endpoints

**`POST /auth/create_key`**

Body schema `CREATE_KEY_SCHEMA` : optional `tier` (default `"hobby"` ).

Response:

```
{
  "status": "success",
  "api_key": "pyn_…",
  "key_id": "12hexchars",
  "tier": "hobby",
  "message": "Store this API key securely. It will not be shown again."
}
```

Decorated with `@require_admin_token` :

- Requires non-empty `ADMIN_TOKEN` env (unset → **403**, minting disabled)
- Client must send matching `X-Admin-Token` (or `Authorization: Bearer …` / `AdminToken …` )
- Comparison uses `hmac.compare_digest`

**`GET /auth/usage`**

`@require_api_key` — returns `key_id` , `tier` , and usage block ( `calls_used` , `calls_limit` , `calls_remaining` , timestamps).

`POST /auth/validate`

Body: `{ "api_key": "…" }` — validates without incrementing usage. Returns tier info, `active`, `rate_limited`.

### Decorators

| Decorator | Behavior |
|---|---|
| `require_api_key` | Resolve key → 401 / 429 or set `g.api_key` |
| `track_usage` | Wraps `require_api_key`; increments calls when response status < 400 (tuple responses) or always for bare Response |
| `require_admin_token` | Fail-closed admin gate (`ADMIN_TOKEN` + `X-Admin-Token`) |

### Client presentation

```
curl -H "Authorization: Bearer $API_KEY" \
  -H "Content-Type: application/json" \
  -d @body.json \
  http://127.0.0.1:5002/preview/chart
```

## Internals

### `APIKey` dataclass

Fields: `key_id`, `key_hash`, `tier`, `calls_used`, `calls_limit`, `created_at`, `last_used`.

Helpers: `is_active()` (always True today), `is_rate_limited()`, `get_tier_info()`, `increment_calls()`.

### Stores

| Module | Use |
|---|---|
| `middleware/auth.py` → `APIKeyStore` + `get_key_store()` | Facade; selects backend via `STORE_BACKEND` |
| `middleware/key_store_sqlite.py` | Hash-only rows, WAL, multi-worker friendly |
| `middleware/key_store_redis.py` | Hash-only Redis; multi-replica |

`get_key_store()` backends (`STORE_BACKEND`):

| Value | Persistence |
|---|---|
| `json` (default) | File at `API_KEY_STORE`; raw keys as object keys (dev only) |
| `sqlite` | `API_KEY_STORE_SQLITE` (or `.db` beside JSON path); hash-only |
| `redis` | `REDIS_URL` required; hash-only |

Prod compose defaults to `sqlite` on `/data/api_keys.db` so gunicorn workers share state.

Key minting:

```
raw_key = "pyn_" + secrets.token_urlsafe(32)
key_id  = sha256(raw_key).hexdigest()[:12]
key_hash = sha256(raw_key).hexdigest()
```

### Schemas

`CREATE_KEY_SCHEMA`, `VALIDATE_KEY_SCHEMA` in `middleware/schemas.py` — strict types, reject unknown fields.

## Invariants and edge cases

1. **`/run` is unauthenticated** — rate-limit at reverse proxy if exposed publicly.
2. **Limit 0 means unlimited**, not "zero calls" — free tier limit value is 0 with that semantics.
3. **Usage increments only after handler returns** — failed auth never bills; handler 4xx under `track_usage` skips increment when returned as `(response, status)`.
4. **Revocation** exists on the store API (`revoke_key`); no public HTTP revoke route in `app.py`.
5. **Admin minting is fail-closed** — unset `ADMIN_TOKEN` or wrong `X-Admin-Token` → 403.

## Worked example

```
# create (requires ADMIN_TOKEN env + matching header)

curl -s -X POST http://127.0.0.1:5002/auth/create_key \
  -H "X-Admin-Token: $ADMIN_TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"tier":"hobby"}'

# validate
curl -s -X POST http://127.0.0.1:5002/auth/validate \
  -H 'Content-Type: application/json' \
  -d '{"api_key":"pyn_…"}'
```

## Failure modes

| Code | Meaning |
|---|---|
| `UNAUTHORIZED` | Missing/invalid key |
| `RATE_LIMITED` | `calls_used >= calls_limit` (finite limits) |
| `INVALID_TIER` | create_key with unknown tier string |
| `INVALID_KEY` | validate endpoint miss |

## See also

- App lifecycle
- Preview
- Security (DevOps)

# RUN



**title: "POST /run" description: "Free evaluate endpoints: single-script /run and multi-script /run/batch over shared OHLCV."**

## POST `/run` AND `/run/batch`

### Abstract

`/run` is the primary free evaluate entry: accept Pine source + bar list, optionally wire `request.*` data sources, execute via `Runtime`, and return plots/series/events/drawings/**alerts**. `/run/batch` runs up to eight scripts on the **same** OHLCV (AXIS multi-indicator), isolating per-script errors so one failure does not discard siblings. Optional **L2 webhooks** POST last-bar (default) `alert()` / `alertcondition()` firings to `webhook_url` or env `ALERT_WEBHOOK_URL`.

## Conceptual model

```
sequenceDiagram
  participant C as Client
  participant A as Flask /run
  participant V as validate RUN_SCHEMA
  participant D as resolve_request_sources
  participant R as Runtime.run

  C->>A: POST JSON
  A->>V: schema + required fields
  V->>D: optional data_source
  D->>R: script, ohlcv, mode
  R-->>C: success payload or EXECUTION_ERROR
```

## Interface surface

**`POST /run`**

### Request ( `RUN_SCHEMA` )

| Field | Type | Required | Default | Notes |
|---|---|---|---|---|
| `script` | string | yes | | Pine source |
| `data` | list | yes | | OHLCV bar objects |
| `symbol` | string | no | `"CHART"` | Syminfo / request wiring |
| `data_source` | string | no | `""` | `mock` / `ccxt` / `yahoo` / … |
| `data_options` | object | no | `{}` | Provider options |
| `mode` | string | no | `"auto"` | `"interpret"` \| `"compile"` \| `"auto"` |
| `inputs` | object | no | `{}` | Pine `input.*` overrides by title |
| `profiler` | bool | no | `false` | Per-line timing on interpret |
| `webhook_url` | string | no | `""` | L2 alert webhook (overrides `ALERT_WEBHOOK_URL` ) |
| `forward_alerts` | bool | no | `true` | When false, skip outbound webhook |
| `alert_last_bar` | bool | no | `true` | Webhook only last OHLCV bar firings |
| `alert_batch` | bool | no | `true` | One batch POST vs per-alert |

Unknown fields → `UNKNOWN_FIELDS` 400. Empty script/data still checked after schema with `NO_SCRIPT` / `NO_DATA` .

Bar shape (runtime expectation):

```
{ "time": 0, "open": 1, "high": 1, "low": 1, "close": 1, "volume": 1 }
```

Optional `bid` / `ask` per bar update runtime bid/ask state.

### Success response

```
{
  "status": "success",
  "plots": [],
  "series": {},
  "plot_meta": {},
  "events": [],
  "drawings": [],
  "alerts": [],
  "script_id": "16hex",
  "run_id": "16hex",
  "count": 0,
  "mode": "interpret",
  "data_source": "chart",
  "alert_forward": {
    "forwarded": 0,
    "failed": 0,
    "filter": "last_bar",
    "url": "https://hooks.example.com/pine",
    "batch": true
  }
}
```

| Field | Meaning |
|---|---|
| `plots` | Primary series (first plot) — backward compatible list |
| `series` | Named multi-plot map title → values per bar |
| `plot_meta` | Per-title color, linewidth, index |
| `events` | Strategy events with `script_id` / `run_id` stamps |
| `drawings` | Exported line/label/box registry for AXIS |
| `alerts` | `alert()` / true `alertcondition()` firings (interpret; empty on compile) |
| `alert_conditions` | Optional full condition evaluations (when present) |
| `alert_forward` | Present when a webhook URL was configured; delivery summary |
| `script_id` | `sha256(source)[:16]` |
| `run_id` | Per-`Runtime` instance id |

### Errors

| HTTP | code | When |
|---|---|---|
| 400 | `MISSING_FIELD` / `INVALID_FIELD` / `UNKNOWN_FIELDS` / `INVALID_BODY` | Schema |
| 400 | `NO_SCRIPT` / `NO_DATA` | Empty after defaults |
| 400 | `DATA_SOURCE_ERROR` | `resolve_request_sources` failed |
| 500 | `EXECUTION_ERROR` | Runtime returned `error` key |

## `POST /run/batch`

### Request ( `RUN_BATCH_SCHEMA` )

| Field | Type | Required | Notes |
|---|---|---|---|
| `scripts` | list | yes | strings or `{id, script}` objects |
| `data` | list | yes | Shared OHLCV |
| `symbol` / `data_source` / `data_options` / `mode` | optional | same as `/run` | |
| `webhook_url` / `forward_alerts` / `alert_last_bar` / `alert_batch` | optional | same L2 webhook fields as `/run` (applied per script result) | |

Hard cap: `RUN_BATCH_MAX_SCRIPTS = 8` → `TOO_MANY_SCRIPTS`.

**Response**

```
{
  "status": "success" | "partial",
  "results": [ { "id": "…", "status": "success|error", … } ],
  "count": 2,
  "ok": 1,
  "data_source": "chart"
}
```

HTTP **200** for partial success (per-script errors inside `results`). Envelope validation failures still 400.

## Internals

```
runtime = Runtime(symbol=str(symbol))
result = runtime.run(script, ohlcv, data_feed=…, data_provider=…, mode=…)
```

See runtime bridge for bar-loop details and compile mode.

Paths: `backend/app.py` (`run_pine_script`, `run_pine_script_batch`), `backend/middleware/schemas.py`, `backend/alert_forwarder.py`.

## Invariants and edge cases

1. **No API key** on these routes — protect at the edge if needed.
2. **Batch isolates exceptions** per script (`try/except` around `runtime.run`).
3. **Whitespace-only `data_source`** coerced to `None` → chart default.
4. **Compile mode** may omit alert side effects (`alerts: []`); use interpret/`auto` for alerts.
5. Schema rejects extras — clients must not send AXIS-only fields without updating schema.
6. **Webhook delivery is best-effort** — evaluate still returns `status: success` if the hook fails; check `alert_forward`.

## Worked example

```
curl -s http://127.0.0.1:5002/run \
  -H 'Content-Type: application/json' \
  -d '{
    "script": "//@version=5\nindicator(\"t\")\nplot(close)",
    "data": [
      {"time": 1, "open": 1, "high": 2, "low": 0.5, "close": 1.5, "volume": 10},
      {"time": 2, "open": 1.5, "high": 2.5, "low": 1.0, "close": 2.0, "volume": 12}
    ],
    "symbol": "TEST:DEMO"
  }'
```

## Failure modes

| Symptom | Cause |
|---|---|
| `UNKNOWN_FIELDS` | Typo'd property (e.g. `ohlcv` instead of `data`) |
| Parse errors in message | Invalid Pine — same engine as CLI |
| Batch `status: partial` | At least one script failed; inspect `results[i].message` |

## Alert webhooks (L2)

```
# Server default (optional)

curl -s http://127.0.0.1:5002/run \
  -H 'Content-Type: application/json' \
  -d '{
    "script": "//@version=5\nindicator(\"a\")\nif bar_index == last_bar_index\n    alert(\"fire\")\
    "mode": "interpret",
    "webhook_url": "https://hooks.example.com/pine",
    "data": [
      {"time": 1, "open": 1, "high": 2, "low": 0.5, "close": 1.5, "volume": 10},
      {"time": 2, "open": 1.5, "high": 2.5, "low": 1.0, "close": 2.0, "volume": 12}
    ]
  }'
```

Full engine rules and batch payload shape: Alerts.

## See also

- Alerts
- Contract
- Runtime bridge
- App lifecycle

# PREVIEW



**title: "Preview Endpoints" description: "Pro chart and indicator thumbnail routes under /preview with usage tracking."**

## PREVIEW ENDPOINTS

### Abstract

Preview routes render **PNG thumbnails** (base64) for desk UIs and docs. They require an API key via `@track_usage`, accept OHLCV-shaped **objects** (column arrays), and do **not** run the full `Runtime` bar loop for `/preview/chart` (data is plotted directly). `/preview/indicator` evaluates a small set of closed-form `ta.*` expressions in pure Python for the series, then plots.

## Conceptual model

```
flowchart LR
  KEY[API key] --> TRACK[track_usage]
  TRACK --> CHART["POST /preview/chart"]
  TRACK --> IND["POST /preview/indicator"]
  CHART --> R1[render_line_chart / render_ohlcv_chart]
  IND --> COMP[_compute_indicator]
  COMP --> R2[render_line_chart]
  R1 --> B64[base64 PNG]
  R2 --> B64
```

Blueprint: `preview_bp` with `url_prefix="/preview"` in `backend/api/preview.py`.

## Interface surface

### Auth

Both routes: `@track_usage` → must pass `require_api_key` and consume one call on success.

**`POST /preview/chart`**

**Body (documented; schemas exist as `PREVIEW_CHART_SCHEMA`)**

```
{
  "script": "optional string",
  "data": {
    "open": [], "high": [], "low": [], "close": [], "volume": []
  },
  "options": {
    "type": "line",
    "color": "#2196F3",
    "show_volume": false,
    "width": 600,
    "height": 300
  }
}
```

| Option | Default | Clamp |
|---|---|---|
| `type` | `"line"` | `"ohlcv"` selects candlestick renderer |
| `color` | `#2196F3` | line chart only |
| `width` | 600 | max 1200 |
| `height` | 300 | max 600 |
| `show_volume` | false | twin-axis volume on line chart |

**Success**

```
{
  "status": "success",
  "chart": "<base64 png>",
  "meta": {
    "type": "line",
    "bars": 252,
    "last_value": 0,
    "first_value": 0,
    "change": 0,
    "change_pct": 0,
    "rendered_at": 0
  },
  "tier_info": {}
}
```

### Errors

| code | When |
|---|---|
| `NO_DATA` | Missing `data` |
| `NO_CLOSE_DATA` | Empty `close` array |
| `RENDER_ERROR` | matplotlib / renderer exception |

Note: `script` is accepted for API symmetry but **not executed** on this path today — the chart is pure data visualization.

## `POST /preview/indicator`

### Body

```
{
  "expression": "ta.sma(close, 14)",
  "data": { "close": [/* … */] },
  "options": { "color": "#FF9800", "width": 600, "height": 300 }
}
```

Supported expression prefixes in `_compute_indicator`:

| Expression | Series |
|---|---|
| `ta.sma(…, period)` | SMA of close |
| `ta.ema(…, period)` | EMA |
| `ta.rsi(…, period)` | RSI |
| `ta.macd(…)` | MACD histogram (12/26/9) |
| other | falls back to raw `close` |

Parse failures in the helper fall back to plotting `close`.

## Internals

| Path | Role |
|---|---|
| `backend/api/preview.py` | Route handlers + indicator math |
| `backend/services/chart_renderer.py` | PNG encode |
| `backend/middleware/auth.py` | `track_usage` |

Schemas `PREVIEW_*` in `schemas.py` document intended validation; current handlers use `request.get_json() or {}` directly.

## Invariants and edge cases

1. **Data shape differs from `/run`:** preview uses **columnar dicts**; `/run` uses **list of bar dicts**.
2. **Not a full Pine interpreter** for indicator expressions — string prefix parsing only.
3. **Width/height clamps** protect worker memory/CPU.
4. Successful responses include `tier_info` from the authenticated key.

## Worked example

```
curl -s http://127.0.0.1:5002/preview/chart \
  -H "Authorization: Bearer $API_KEY" \
  -H 'Content-Type: application/json' \
  -d '{
    "data": {"close": [1,2,3,4,5], "volume": [10,10,10,10,10]},
    "options": {"type": "line", "show_volume": true}
  }' | jq '.meta'
```

## Failure modes

| Symptom | Cause |
|---|---|
| 401 | Missing key |
| 429 | Tier limit |
| Empty-looking PNG | Renderer empty-chart path when all values null |

## See also

- Chart renderer
- Auth and keys
- Run

# BACKTEST



**title: "Backtest Endpoint" description: "POST /backtest/quick — usage-tracked strategy metrics, equity curve, and optional mock OHLCV."**

## BACKTEST ENDPOINT

### Abstract

`POST /backtest/quick` is a Pro, usage-tracked route that runs a **simplified** strategy simulation over columnar OHLCV (or generated mock bars), returns trade lists + summary metrics, and embeds a base64 equity-curve PNG. It is optimized for speed/demo UX; it is not a full fidelity substitute for event-aware `Runtime` strategy evaluation on `/run`.

## Conceptual model

```
flowchart TD
  REQ[POST /backtest/quick] --> AUTH[track_usage]
  AUTH --> DATA{data or mock_data?}
  DATA -->|mock / empty close| MOCK[generate_mock_ohlcv]
  DATA -->|provided| OHLCV[columnar OHLCV]
  MOCK --> SIM[run_quick_backtest]
  OHLCV --> SIM
  SIM --> METRICS[equity, trades, summary]
  SIM --> PNG[render_equity_curve]
```

Blueprint: `backtest_bp`, prefix `/backtest`.

## Interface surface

### Request

```
{
  "script": "//@version=5\nstrategy(\"s\")…",
  "data": {
    "open": [], "high": [], "low": [], "close": [], "volume": []
  },
  "initial_capital": 10000.0,
  "mock_data": false,
  "mock_bars": 252
}
```

| Field | Default | Notes |
|---|---|---|
| `script` | required | Empty → `NO_SCRIPT` |
| `data` | `{}` | Columnar; may be omitted if `mock_data` |
| `initial_capital` | `10000` | Starting equity |
| `mock_data` | `false` | Force synthetic bars |
| `mock_bars` | `252` | Length of mock series |

If neither usable `close` nor `mock_data` → `NO_DATA`.

### Success


```
{
  "status": "success",
  "result": {
    "equity_curve": [10000, …],
    "trades": [
      {
        "entry_time": 0,
        "entry_price": 0,
        "exit_time": 0,
        "exit_price": 0,
        "direction": "long",
        "pnl": 0,
        "pnl_pct": 0,
        "size": 1
      }
    ],
    "summary": {
      "total_pnl": 0,
      "total_pnl_pct": 0,
      "sharpe_ratio": 0,
      "max_drawdown": 0,
      "max_drawdown_pct": 0,
      "win_rate": 0,
      "profit_factor": 0,
      "total_trades": 0,
      "winning_trades": 0,
      "losing_trades": 0,
      "avg_win": 0,
      "avg_loss": 0
    },
    "equity_chart": "<base64 png>"
  },
  "tier_info": {},
  "meta": {
    "bars": 252,
    "initial_capital": 10000,
    "completed_at": 0
  }
}
```


### Errors

| code | HTTP | When |
|---|---|---|
| `NO_SCRIPT` | 400 | Empty script |
| `NO_DATA` | 400 | No data and not mock |
| `BACKTEST_ERROR` | 500 | Exception in simulation |
| `UNAUTHORIZED` / `RATE_LIMITED` | 401 / 429 | Auth |

## Internals

`run_quick_backtest` → `run_backtest` in `backend/services/backtest.py`.

MVP simulation characteristics:

1. Optionally `parse(script)` (errors soft-ignored for MVP path).
2. Precompute long/short entry signals from dual SMA cross (10 vs 20) starting at bar 20.
3. Exit heuristics via RSI-like average and opposite signals / end of series.
4. Equity curve + trade PnL with optional commission/slippage parameters on the lower-level API.
5. Metrics: Sharpe (√252 scaling), max drawdown, win rate, profit factor.
6. Chart via `render_equity_curve`.

`generate_mock_ohlcv(n_bars)` produces synthetic columns for demos.

| Path | Role |
|---|---|
| `backend/api/preview.py` | `quick_backtest` route |
| `backend/services/backtest.py` | Simulation + metrics |
| `backend/services/chart_renderer.py` | Equity PNG |

## Invariants and edge cases

1. **Script content is lightly used** in the MVP sim — do not treat results as broker-accurate for arbitrary strategies. Prefer `/run` events for engine-faithful strategy traces.
2. **Columnar data**, same family as preview, not `/run` bar lists.
3. **Mock path** ignores incomplete user data when `mock_data` or empty close.
4. One usage increment per successful HTTP response under `track_usage`.

## Worked example

```
curl -s http://127.0.0.1:5002/backtest/quick \
  -H "Authorization: Bearer $API_KEY" \
  -H 'Content-Type: application/json' \
  -d '{
    "script": "//@version=5\nstrategy(\"demo\")\n// body optional for MVP",
    "mock_data": true,
    "mock_bars": 120,
    "initial_capital": 25000
  }' | jq '.result.summary'
```

## Failure modes

| Symptom | Cause |
|---|---|
| Implausible trades vs script logic | MVP signal model, not full evaluator |
| Empty trade list | No crosses in series / short history |
| Missing `equity_chart` | Renderer exception swallowed → empty string |

## See also

- Runtime bridge — faithful strategy events via `/run`

- Chart renderer
- Strategy builtins

# CHART RENDERER



**title: "Chart Renderer" description: "Matplotlib Agg PNG services: line charts, OHLCV candles, equity curves, base64 encoding."**

# CHART RENDERER

## Abstract

`backend/services/chart_renderer.py` is a headless plotting façade. It forces `matplotlib` into **Agg**, draws dark-theme charts for desk thumbnails, and returns **base64-encoded PNG** strings suitable for JSON transport. Callers are the preview and backtest routes; the module has no Flask dependency.

## Conceptual model

```
flowchart LR
  VAL[series / OHLCV / equity] --> PLT[matplotlib Figure Agg]
  PLT --> BUF[BytesIO PNG]
  BUF --> B64[base64 utf-8 string]
  B64 --> API[JSON chart / equity_chart fields]
```

## Interface surface

### `render_line_chart`

| Param | Default | Role |
|---|---|---|
| `values` | required | Y series; `None` → NaN gaps |
| `dates` | `None` | Optional x tick labels (≤ ~6 ticks) |
| `title` | `"Chart"` | Title string |
| `color` | `#2196F3` | Line + fill |
| `height` / `width` | 300 / 600 | Pixel intent → figsize `/100`, dpi 100 |
| `show_volume` | false | Twin axis bars from `ohlcv["volume"]` |
| `ohlcv` | `None` | Volume source when enabled |

Empty / all-null values → `_render_empty_chart` placeholder.

### `render_ohlcv_chart`

Candlestick + volume subplot pair from dict keys `open`, `high`, `low`, `close`, `volume`. Green/red body colors (`#4CAF50` / `#F44336`). Shared x-axis.

### `render_equity_curve`

Single series with profit/loss fill relative to zero baseline and a dashed reference at the first equity point. Title fixed `"Equity Curve"`.

### Return type

All public renderers → `str` (base64 PNG without data-URI prefix). Clients typically use:

```
{chart}
```

## Internals

| Detail | Implementation |
|---|---|
| Backend | `matplotlib.use("Agg")` before pyplot import |
| Theme | fig `#1E1E1E`, axes `#252526`, muted grids |
| Cleanup | `plt.close(fig)` after `savefig` |
| Empty state | Centered “No data available” text |

| Path | Role |
|---|---|
| `backend/services/chart_renderer.py` | Renderers |
| `backend/api/preview.py` | Consumers |
| `backend/services/backtest.py` | Equity chart hook |

## Invariants and edge cases

1. **Not interactive** — no GUI backend; safe for servers.
2. **NaNs** break lines at gaps (numpy nan).
3. **Candle body height** uses `+ 1e-9` to keep zero-range bodies visible.
4. **Thread safety:** matplotlib global state is historically process-local; multi-threaded gunicorn workers should avoid concurrent pyplot use on one worker or serialize renders.
5. Exceptions in callers often become HTTP `RENDER_ERROR` or empty equity chart strings.

## Worked example

```
from backend.services.chart_renderer import render_line_chart

b64 = render_line_chart([1.0, 1.2, 1.1, 1.4], title="demo", color="#A3E635")
assert isinstance(b64, str) and len(b64) > 100
```

## Failure modes

| Symptom | Cause |
|---|---|
| ImportError Agg | Incomplete matplotlib install |
| Huge base64 | High width/height — routes clamp preview sizes |
| Blank image | Empty input path succeeded with placeholder |

## See also

- Preview endpoints
- Backtest

# RUNTIME BRIDGE



**title: "Runtime Bridge" description: "backend.runtime.Runtime — bar-loop evaluate façade, PineSeries context, compile mode, and result packaging."**

# RUNTIME BRIDGE

## Abstract

`backend.runtime.Runtime` is the HTTP stack's **evaluate façade**. It owns symbol metadata namespaces (`syminfo`, `timeframe`, `barstate`, …), walks OHLCV bars, updates `PineSeries` for OHLC, visits a shared `CustomEvaluator` on the parsed AST each bar, drains strategy events, and packages multi-plot series for AXIS. Compile mode optionally swaps the AST walker for a Numba-backed subset engine.

This is the bridge between Flask handlers and the language core — not a second semantics.

## Conceptual model

```
flowchart TB
  SRC[Pine source] --> PARSE[parse mode=exec]
  OHLCV[bar list] --> LOOP[for each bar]
  PARSE --> LOOP
  LOOP --> SER[PineSeries.update OHLC]
  LOOP --> CTX[context bar_index time barstate]
  LOOP --> FILL[process_pending_orders]
  LOOP --> VISIT[evaluator.visit tree]
  VISIT --> PLOTS[plot_outputs]
  VISIT --> EV[strategy events]
  VISIT --> AL[alert firings]
  LOOP --> PACK[series map + plot_meta + drawings + alerts]
```

## Interface surface

### Construction

```
Runtime(symbol: str = "AAPL", run_id: str | None = None)
```

- Sets `syminfo.tickerid` / `name` / `prefix` (prefix from `EXCHANGE:SYMBOL` form).
- Generates `run_id` as `uuid4().hex[:16]` when omitted.

Helpers: `configure_footprint`, `update_bid_ask`.

**`run(source_code, ohlcv_data, data_feed=None, data_provider=None, mode="interpret")`**

| Arg | Role |
|---|---|
| `source_code` | Pine text |
| `ohlcv_data` | `list[dict]` with open/high/low/close/time[/volume/bid/ask] |
| `data_feed` / `data_provider` | `request.*` wiring; auto-resolved from chart bars when unset |
| `mode` | `"interpret"` (default) or `"compile"` |

#### Interpret path (default)

1. Resolve request sources (non-fatal on failure).
2. `parse(source_code, mode="exec")` — on failure `{ "error": "Parse Error: …" }`.
3. Init `PineSeries` for OHLC + context dict (timeframe daily defaults, barstate, chart colors).
4. Construct `CustomEvaluator(context=…, data_feed=…, data_provider=…)`, reset var declarations + drawing registry + `clear_alerts()`.
5. Per bar:
   - Update series + calendar fields from timestamp
   - Update barstate flags (`isfirst` / `islast` / history confirmed)
   - `process_pending_orders` when available
   - `evaluator.visit(tree)` — on failure `{ "error": "Runtime Error at bar …" }`
   - Lock pine defs after first bar (`_pine_defs_locked`) to avoid multi-dispatch blow-ups

- Drain strategy events; stamp `script_id` / `run_id`
- Capture plot outputs

6. Build `series` / `plot_meta` from all plot indices (disambiguate duplicate titles with `_2` suffixes).
7. Export drawings via `DrawingRegistry.export_for_api(bar_times)`.
8. Export alerts via `export_alerts_from_evaluator` (optional `alert_conditions`).

Return keys: `plots`, `series`, `plot_meta`, `events`, `drawings`, `alerts`, `count`, `script_id`, `run_id`, `mode` (+ optional `alert_conditions`).

#### Compile path

1. Import `pynescript.compiler.engine` — missing → error string.
2. Require `has_numba()` (object-mode may still run without pure Numba for some scripts).
3. `compile_script` + `compiled.run(opens, highs, lows, closes, volumes)`.
4. Pop internal keys (`__drawings`, `__events`, position meta).
5. Return similar envelope with `mode: "compile"`, `alerts: []`, plus `object_mode` / `generated_code` when present.

### `PineSeries` (`backend/series.py`)

History deque (default maxlen 1000), `series[0]` current / `series[n]` lookback, arithmetic on **current** values, `None` propagates as na-like absence. Negative indices raise — Pine does not allow them.

### Context namespaces

Defined in `runtime.py`: `Syminfo`, `Chartinfo`, `Timeframe`, `Barstate`, `Chart` — attribute names aim at Pine v5/v6 surface (`timeframe.isdaily`, `syminfo.isin`, …).

## Internals

| Path | Role |
|---|---|
| `backend/runtime.py` | `Runtime` |
| `backend/evaluator.py` | `CustomEvaluator` specialization |
| `backend/series.py` | `PineSeries` |
| `src/pynescript/ast/helper.py` | `parse` |
| `src/pynescript/ast/evaluator/**` | Builtin + strategy semantics |
| `src/pynescript/compiler/engine.py` | Compile mode |

Flask wiring: `backend/app.py` constructs a **fresh** `Runtime` per request (and per batch job) so `run_id` and drawing registries do not leak across users.

## Invariants and edge cases

1. **Parse once, visit many** — AST is not re-parsed per bar.
2. **Defs locked after bar 0** — performance invariant for large function tables.
3. **Drawing registry reset** at run start — isolation between requests.
4. **Primary `plots` list** is plot index 0 for backward compatibility; AXIS should prefer `series` + `plot_meta`.

5. **Errors are dicts**, not exceptions, for control-flow at the HTTP boundary (`"error" in result`).
6. Compile mode supports a **subset** (documented in engine — ta/math/plots family); unsupported scripts should use interpret.

## Worked example

```
from backend.runtime import Runtime

rt = Runtime(symbol="NASDAQ:AAPL")
out = rt.run(
    '//@version=5\nindicator("x")\nplot(close, "C")',
    [{"time": i, "open": 1, "high": 2, "low": 0.5, "close": 1.0 + i * 0.01, "volume": 1} for i in r
)
assert "error" not in out
assert out["count"] == 50
assert "C" in out["series"] or "plot_0" in out["series"]
```

## Failure modes

| Message prefix | Cause |
|---|---|
| `Parse Error:` | Grammar / syntax |
| `Runtime Error at bar` | Evaluator exception |
| `Order fill error at bar` | Broker sim pending orders |
| `Compile mode requires numba` | Missing dependency |
| `Compile Error:` / `Compiled Runtime Error:` | Engine path |

## See also

- Contract
- Run endpoint
- Series and history
- Strategy events

# CONTRACT



**title: "Evaluate Contract" description: "Shared request/response evaluate contract across Flask Pro API, pyne-worker, and pine-worker."**

# EVALUATE CONTRACT

## Abstract

The **evaluate contract** is the JSON dialect that lets AXIS, HOOX, CLI tools, and edge workers call *any* PYNE-compatible host interchangeably. Flask `POST /run` is the reference implementation; Python **pyne-worker** and TypeScript **pine-worker** aim for the same fields even when transport (Worker `fetch`, queue, etc.) differs.

This page freezes the semantic envelope — not transport auth headers.

## Conceptual model

```
flowchart LR
  subgraph Producers
    AXIS[AXIS]
    CLI[CLI / scripts]
    HOOX[HOOX mesh]
  end
  subgraph Hosts
    FLASK[Flask /run]
    PYW[pyne-worker]
    TSW[pine-worker]
  end
  AXIS --> FLASK
  AXIS --> PYW
  AXIS --> TSW
  CLI --> FLASK
  HOOX --> PYW
  HOOX --> TSW
```

**Invariant:** given the same script + OHLCV + mode, hosts should agree on plot series and strategy events within known parity limits (see numerical validation and pine-worker testing).

## Interface surface

### Request (single evaluate)

| Field | Type | Required | Description |
|---|---|---|---|
| `script` | string | yes | Full Pine source |
| `data` | `Bar[]` | yes | Chronological OHLCV |
| `symbol` | string | no | Default `CHART` / host-specific |
| `mode` | `"interpret"` \| `"compile"` | no | Default interpret |
| `data_source` | string | no | Optional external history provider id |
| `data_options` | object | no | Provider configuration |

**`Bar`**

| Field | Type | Required | Notes |
|---|---|---|---|
| `time` | number | recommended | Unix ms preferred |
| `open` | number | yes | |
| `high` | number | yes | |
| `low` | number | yes | |
| `close` | number | yes | |
| `volume` | number | no | Default host-dependent |
| `bid` / `ask` | number | no | Runtime bid/ask hooks |

### Response (success)

| Field | Type | Description |
|---|---|---|
| `status` | `"success"` | Flask sets this; workers may omit and use HTTP 200 only — **prefer including it** |
| `plots` | `(number\|null)[]` | Primary series (plot 0) |
| `series` | `Record<string, (number\|null)[]>` | Named multi-series |
| `plot_meta` | `Record<string, PlotMeta>` | Display metadata |
| `events` | `StrategyEvent[]` | Strategy / trade-like events |
| `drawings` | `Drawing[]` | line/label/box export |
| `alerts` | `AlertEvent[]` | `alert()` / true `alertcondition()` firings (interpret) |
| `alert_conditions` | object[] | optional condition evaluations |
| `alert_forward` | object | optional L2 webhook delivery summary |
| `count` | number | Bars evaluated |
| `script_id` | string | Stable hash prefix of source |
| `run_id` | string | Per-invocation id |
| `mode` | string | Echo effective mode |
| `data_source` | string | Flask echo of resolved source label |

#### `PlotMeta` (Flask)

```
{ "title": "C", "color": "#2196F3", "linewidth": 1, "index": 0 }
```

#### `StrategyEvent` (minimum)

Hosts should preserve:

```
{
  "type": "string",
  "bar_index": 0,
  "script_id": "…",
  "run_id": "…"
}
```

Additional fields follow evaluator `to_dict()` / TS port parity — treat unknown keys as forward-compatible.

#### `AlertEvent` (minimum)

```
{
  "message": "string",
  "freq": "once_per_bar",
  "bar_index": 0,
  "time": 0,
  "source": "alert",
  "script_id": "…",
  "run_id": "…"
}
```

See Alerts for frequency rules and webhooks.

### Response (failure)

Flask `/run`:

```
{
  "status": "error",
  "code": "EXECUTION_ERROR",
  "message": "Runtime Error at bar …"
}
```

Workers may use equivalent `{ error: string }` or HTTP 4xx/5xx with message body. Clients should accept:

1. Envelope `status === "error"` with `message`, or
2. Presence of top-level `error` string (raw `Runtime.run` shape).

### Batch evaluate (Flask extension)

`POST /run/batch`:

**Request:** `scripts: (string | {id, script})[]` (max 8) + shared `data` + same optional fields as single run.

**Response:**

```
{
  "status": "success" | "partial",
  "results": [ /* per-script success or error object with id */ ],
  "count": 0,
  "ok": 0,
  "data_source": "chart"
}
```

Workers may implement batch as N single evaluates; AXIS should not assume batch exists on every host.

## Internals — reference mapping

| Contract field | Flask source |
|---|---|
| `plots` / `series` / `plot_meta` | `Runtime.run` packing loop |
| `events` | `evaluator._strategy_state.drain_events()` → `to_dict()` |
| `drawings` | `DrawingRegistry.export_for_api` |
| `alerts` | `export_alerts_from_evaluator` (interpret) |
| `alert_forward` | `backend/alert_forwarder.maybe_forward_run_alerts` |
| `script_id` | `sha256(source)[:16]` |
| `run_id` | `Runtime._run_id` |

Schema enforcement on Flask free tier: `RUN_SCHEMA` / `RUN_BATCH_SCHEMA` in `backend/middleware/schemas.py` (strict, reject extras). Webhook fields: `webhook_url`, `forward_alerts`, `alert_last_bar`, `alert_batch`.

Tests: `tests/test_backend.py` (`test_run_success`, `test_run_exports_alerts`, webhook cases), `tests/test_alert_forwarder.py`.

## Divergences to remember

| Surface | Divergence |
|---|---|
| Preview / backtest | Columnar `data` dict, not `Bar[]`; not the evaluate contract |
| Compile mode | Extra keys `object_mode`, `generated_code` may appear |
| Auth | Flask free `/run` open; workers may require HOOX mTLS / API gateways |
| Error codes | Flask uses `code` enums; workers should map to stable strings when possible |

## Invariants and edge cases

1. **Bar order is chronological ascending** — hosts do not sort for you.
2. **`na` / missing** appear as JSON `null` in series arrays.
3. **Duplicate plot titles** get suffixes (`_2`, …) on Flask; clients should key on returned map keys, not assumed titles.
4. **Idempotent script_id** for identical source text across hosts.
5. **run_id is not stable** across retries — use for log correlation only.
6. Parity is best-effort on floating edges; pin fixtures when testing hosts against each other.

## Worked example — host-agnostic client sketch

```
type EvaluateRequest = {
  script: string;
  data: Array<{
    time: number;
    open: number;
    high: number;
    low: number;
    close: number;
    volume?: number;
  }>;
  symbol?: string;
  mode?: "interpret" | "compile";
};

type EvaluateSuccess = {
  plots: Array<number | null>;
  series: Record<string, Array<number | null>>;
  events: Array<Record<string, unknown>>;
  drawings: Array<Record<string, unknown>>;
  count: number;
  script_id: string;
  run_id: string;
  mode: string;
};
```

POST that body to Flask `/run` or the worker's evaluate route; branch on `status` / `error`.

## Failure modes

| Client bug | Symptom |
| --- | --- |
| Sending columnar preview data to `/run` | Schema `data must be a list` |
| Assuming batch on worker | 404 / unknown field |
| Ignoring `series` and only reading `plots` | Multi-plot scripts look single-series |

## See also

- Run endpoint
- Runtime bridge
- pine-worker
- Pro API usage (end user)
- Events