# ttTrader: The Complete Algorithmic Trading Platform

Configuration, exchanges, algorithms and indicators - the complete user manual

ttTrader

Generated 2026-09-12

# Contents

# Part I

# Part I - Getting started

# Chapter 1

# ttTrader: The Complete Algorithmic Trading Platform

**Version:** 0.1.0

## 1.1 Overview

ttTrader is a high-frequency trading platform designed for low-latency market data ingestion, order execution, and algorithmic strategy deployment across multiple centralised and decentralised exchanges. The system compiles a monolithic host process (`ttTrader.exe`) that dynamically loads exchange adapters, trading algorithms, and technical indicators as shared-library plugins at runtime.

The architecture is event-driven: all inter-module communication flows through a typed event bus. A layered cascading shutdown protocol ensures safe teardown of asynchronous WebSocket streams before plugin destruction.

### 1.1.1 What it gives you

- **One host, many venues and strategies.** Configure any mix of exchanges, algorithms and indicators; ttTrader loads and runs them together.
- **Simulation, paper and live from one file.** Switch the trading mode per run without editing your configuration.
- **Isolated multi-strategy accounts.** Each algorithm gets its own per-asset balance share, and only its own profit and loss moves that balance.
- **Deterministic, safe operation.** Clear startup validation and a staged, confirmation-driven shutdown.

---

## 1.2 Quick Start

```
# Run with a configuration file
ttTrader.exe <config.json>

# Override the trading mode for this run (live | paper | simulation | playback)
ttTrader.exe <config.json> --tradingMode paper
```

```
# Structured logging and market-data logging
ttTrader.exe <config.json> --log --logmarketdata
```

**Command line options:** `--config` (config file), `--tradingMode` (mode override selecting a config's `serverSettings` client set — one config file can carry sim / paper / live sets and the mode is switched per run), `--nocolor`, `--nosound`.

Logging is config-driven: the structured JSONL log family (orders, executions, positions, account, system) is controlled by the `"logging"` config block — see `03-architecture/logging.md`.

**Prerequisites:** - CMake 3.31+ - Visual Studio 2022 (17.4+) with C++23 support *or* Clang 18+ - Boost 1.x (thread, json, log) - OpenSSL - Intel oneAPI IPP - ttSubSystem (sibling project, pre-built) - Windows Credential Manager or Linux libsecret for API key storage

See `02-technical-stack/README.md` for full dependency versions and build-system details.

---

## 1.3 Documentation Map

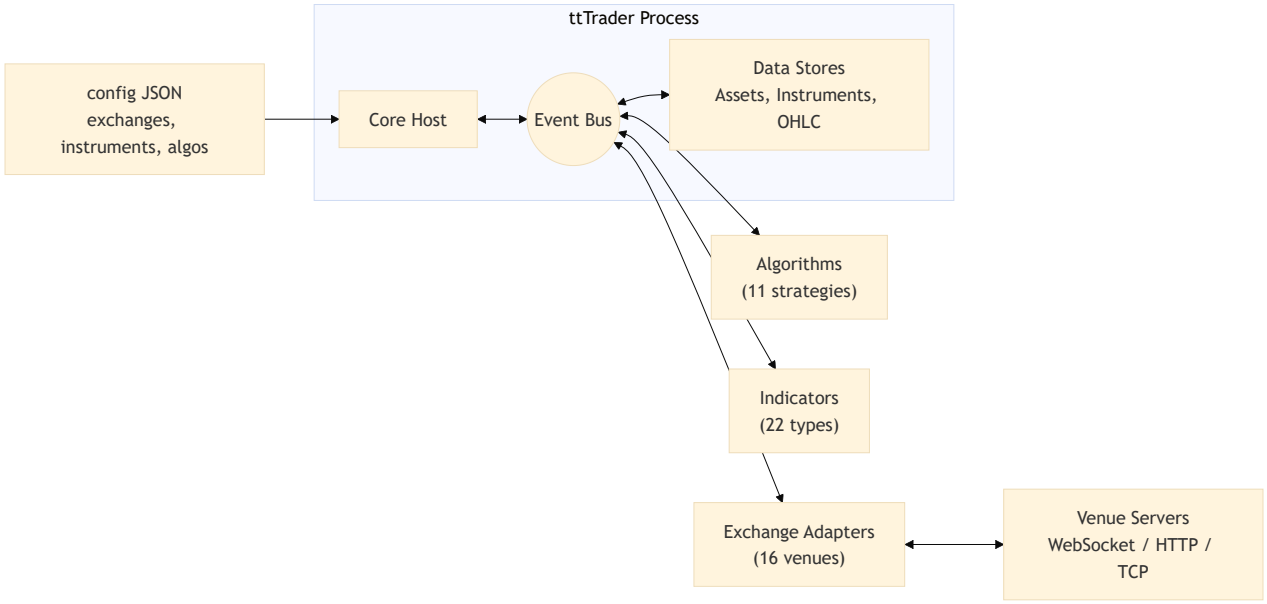
The canonical, curated index is the **ttTrader Manual**. The table below is a quick summary; the manual carries the full chapter and per-component tables.

| # | Section | Path | Contents |
|---|---|---|---|
| 0 | **Manual (index)** | `MANUAL.md` | Curated entry point: full TOC, per-exchange/-algo/-indicator indexes, known gaps |
| 1 | **System Protocol** | `01-system-protocol/` | Event bus, event ID ranges, cascading shutdown protocol, plugin lifecycle, RAII lifecycle templates |
| 2 | **Technical Stack** | `02-technical-stack/` | Third-party dependencies, compiler requirements, CMake presets, build flags, binary layout |
| 3 | **Architecture** | `03-architecture/` | Manager hierarchy, singleton stores, data flow from exchange to algo, strand-based async model |
| 4 | **Exchange Integrations** | `04-exchange-integrations/` | Adapter framework, protocol flags, connection lifecycle, creating a new adapter |
| 5 | **Trading Algorithms** | `05-trading-algorithms/` | `algoFramework_c` API, algo lifecycle, manager composition, creating a new algo |
| 6 | **Indicators** | `06-indicators/` | `indicatorCore_c` API, OHLC attachment, plugin wrappers, creating a new indicator |
| 7 | **Development Guide** | `07-development/` | Coding conventions, adding plugins, credential management, debugging, profiling with Tracy |

| # | Section | Path | Contents |
|---|---|---|---|
| — | **Credential Store** | `CredentialStore-README.md` | Windows Credential Manager, Linux libsecret, kernel keyring — how to store API keys securely |
| — | **Exchange README** | `../exchanges/README.md` | Per-adapter status, connection lifecycle Mermaid diagram, message classification, shutdown architecture |

---

## 1.4 System at a Glance



Exchange adapters, algorithms and indicators are plugins declared in the same configuration file; they all communicate through one event bus. See Configuration for the file layout and Balance shares and PnL isolation for how several strategies share one account.

---

### 1.4.1 Algorithm Intelligence and Portfolio Exposure

The algorithm intelligence panel shows real-time signal strength, position sizing and regime detection across all active strategies. The portfolio exposure view provides per-asset and per-algorithm risk breakdowns.

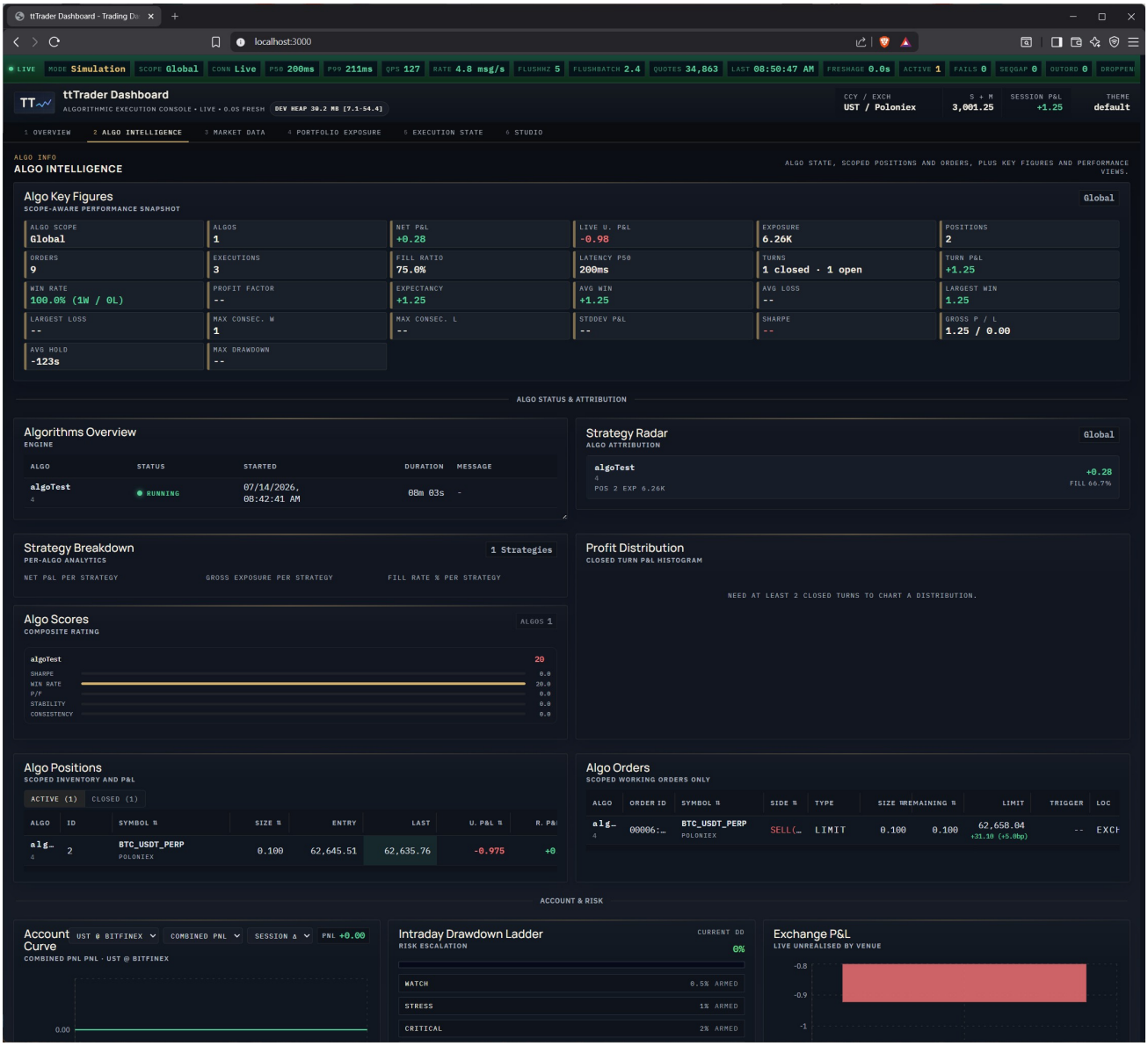



Figure 1.1: Algorithm intelligence panel

### 1.4.2 Trading Modes

ttTrader supports four trading modes selected per run via `--tradingMode` or the configuration file. All modes share the same event bus and plugin surface; only the execution backend changes.

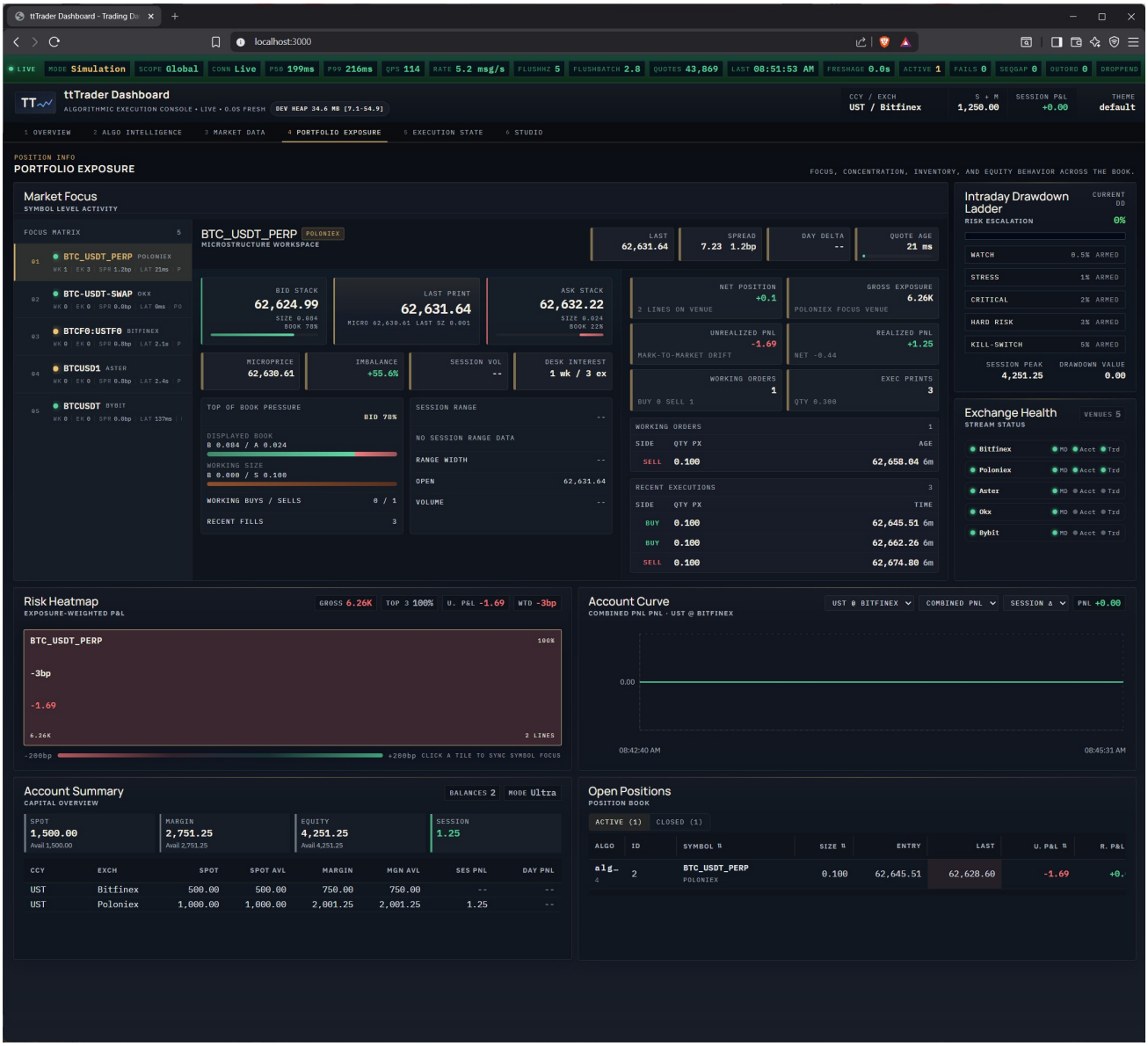
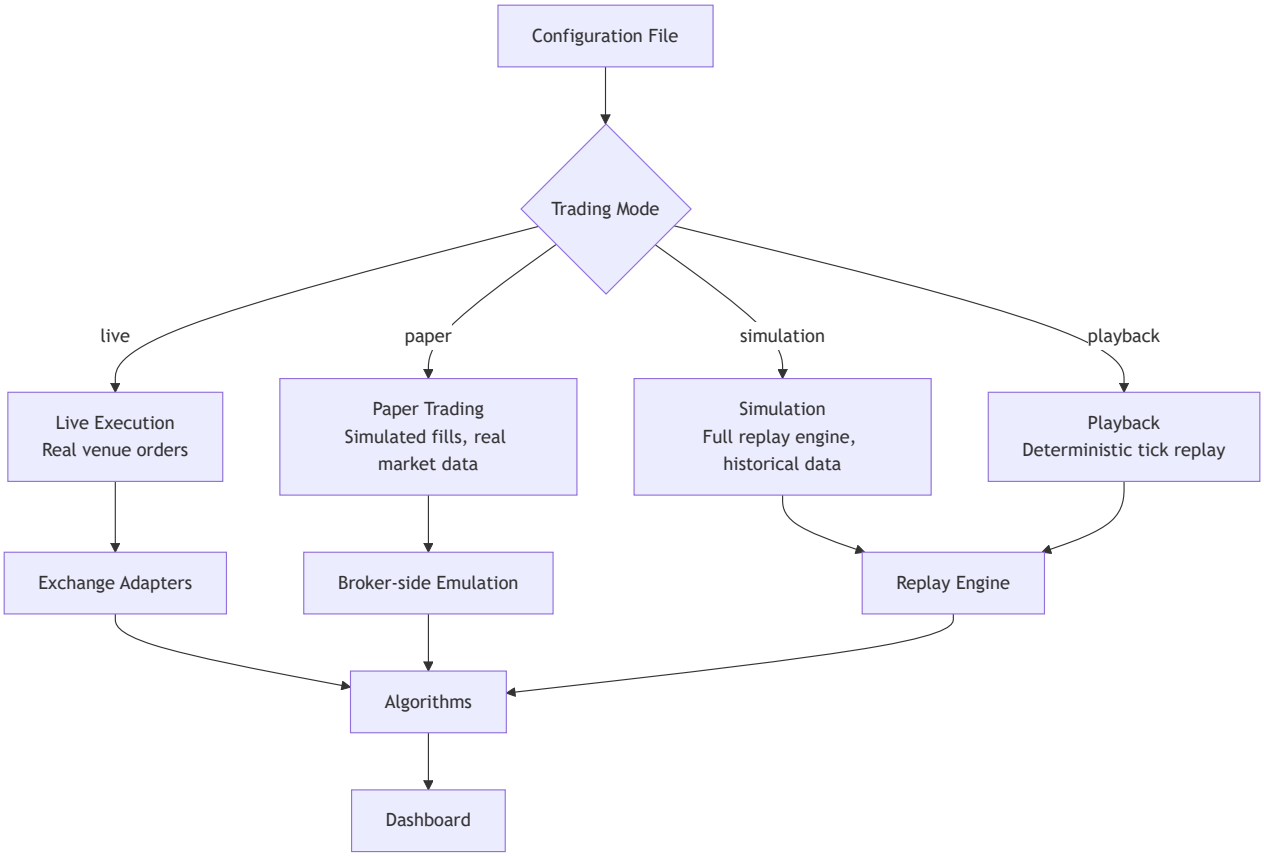



Figure 1.2: Portfolio exposure view


Configuration File
Trading Mode
live
paper
simulation
playback
Live Execution
Real venue orders
Paper Trading
Simulated fills, real
market data
Simulation
Full replay engine,
historical data
Playback
Deterministic tick replay
Exchange Adapters
Broker-side Emulation
Replay Engine
Algorithms
Dashboard

## 1.5 Dashboard

The ttTrader Dashboard (`ttTraderDash`) is an institutional-grade monitoring console that connects to a live trading stack over a single WebSocket feed. It presents orders, fills, positions, account state, market data, and execution-quality analytics across six purpose-built pages.

**Monitoring only.** The dashboard never sends order instructions beyond explicit desk risk controls (reduce-only, cancel-all, flatten-all), each with confirmation and ACK tracking.

### 1.5.1 Pages

| Page | Purpose |
|---|---|
| **Overview** | Desk KPI command bar: Net P&L, profit factor, fill quality, gross exposure, active orders, market-data latency, UI render p95 budget, alert log |
| **Algo Intelligence** | 26-metric progressive-disclosure grid, Strategy Radar with VWAP deviation, algo scoring, closed-turn profit distribution with Gaussian fit, account value curve (Live / 3H / 24H) |
| **Market Data** | Latency histograms (per-exchange overlay), latency-to-fill correlation, virtualized live quotes grid, price chart (1m / 5m / 1h), trade tape |
| **Portfolio Exposure** | Symbol focus workspace, vol-weighted risk heatmap with VaR 95 estimates, open positions, concentration tracking |
| **Execution State** | Markout analysis (T+1s/+5s/+30s/+60s), implementation-shortfall decomposition, order completion analytics, severity-scored exception rail, order blotter with lifecycle timeline |
| **Studio** | Record live sessions to `.ttpb` files and replay with transport controls (play/pause/step/seek, speed selection, loop) |

### 1.5.2 Key Capabilities

- **Full order-type visibility** — stop market/limit, trailing stops (live trailed trigger, delta, direction), GTD expiry countdowns, GAD deferred activation. The `loc` field identifies lifecycle ownership: LOCAL, BROKER, EXCHANGE, or MIXED.
- **TCA suite** — slippage vs. decision price, post-fill markout analysis, per-order implementation-shortfall decomposition, per-algo VWAP deviation.
- **Latency observability** — Gaussian-fit latency histograms, pending-ack aging, latency-to-fill correlation, UI render p95 to separate feed slowness from frame drops.
- **High-frequency rendering** — rAF-batched quote ingestion (200ms max-wait), virtualized tables, cell-level change flash, retention caps protecting instruments with open positions.
- **Session-scale memory** — tiered equity retention (3 min live / 3h session / 24h day) and tiered candles (1m/2h, 5m/24h, 1h/7d).
- **Unified alert model** — INFO to log, ELEVATED to exception rail, CRITICAL to persistent acknowledgment-required banner.

### 1.5.3 Architecture

The dashboard is a React 18 + TypeScript + Vite application using Tailwind CSS, Recharts, native WebSocket transport, and IndexedDB archival. It speaks the v3 structured-envelope protocol (`{ seq, t, d }` frames with strict single-key contract). Realized P&L, position size, fees, equity, and timestamps are backend-computed; the frontend computes only live unrealized P&L from BBO.

WebSocket auto-discovery: `VITE_WS_URL -> ws(s)://<host>:18181 -> ws://[IP_ADDRESS]:18181 -> ws://localhost:18181 -> ws://localhost:9001`.

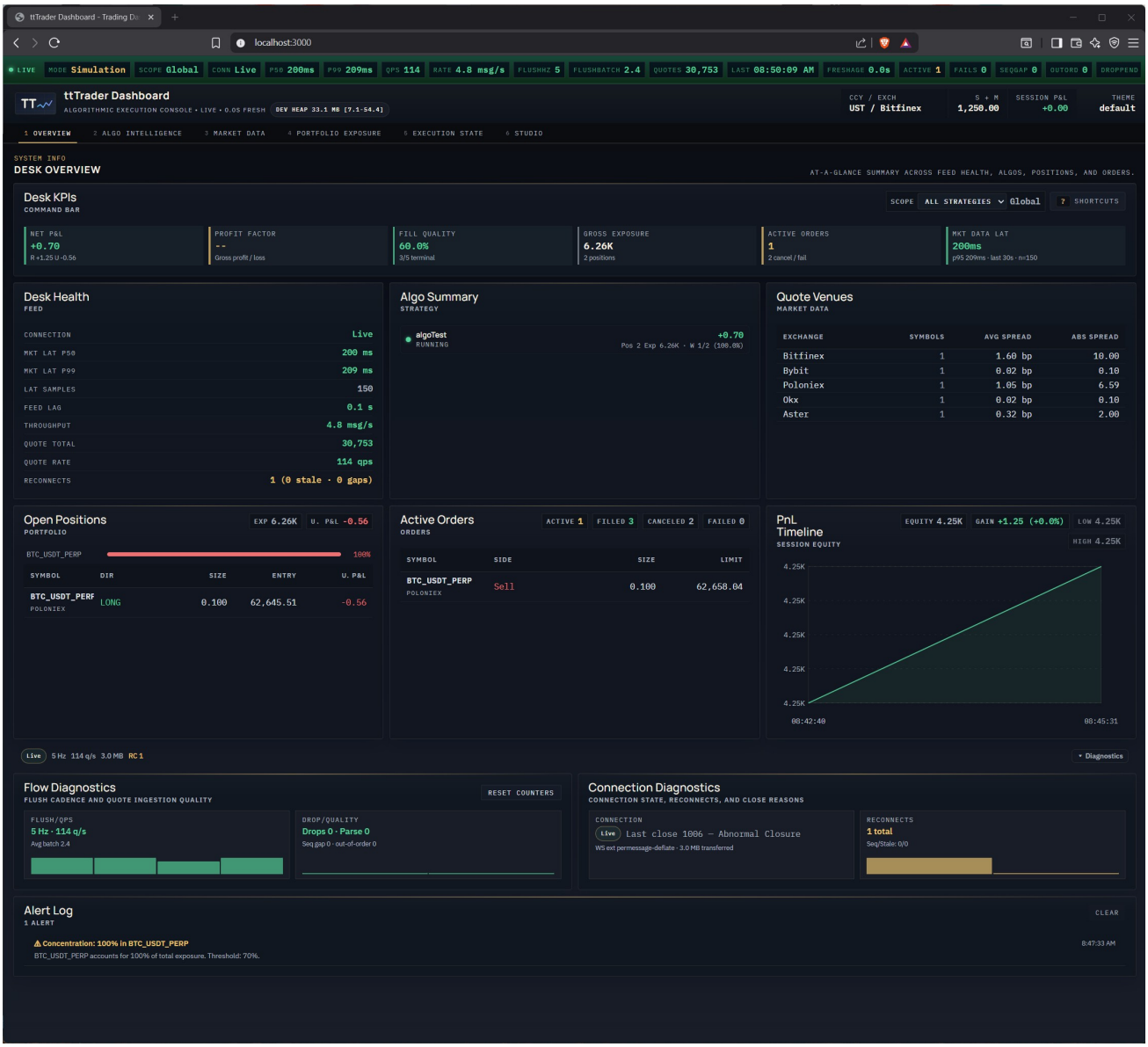



Figure 1.3: Dashboard overview

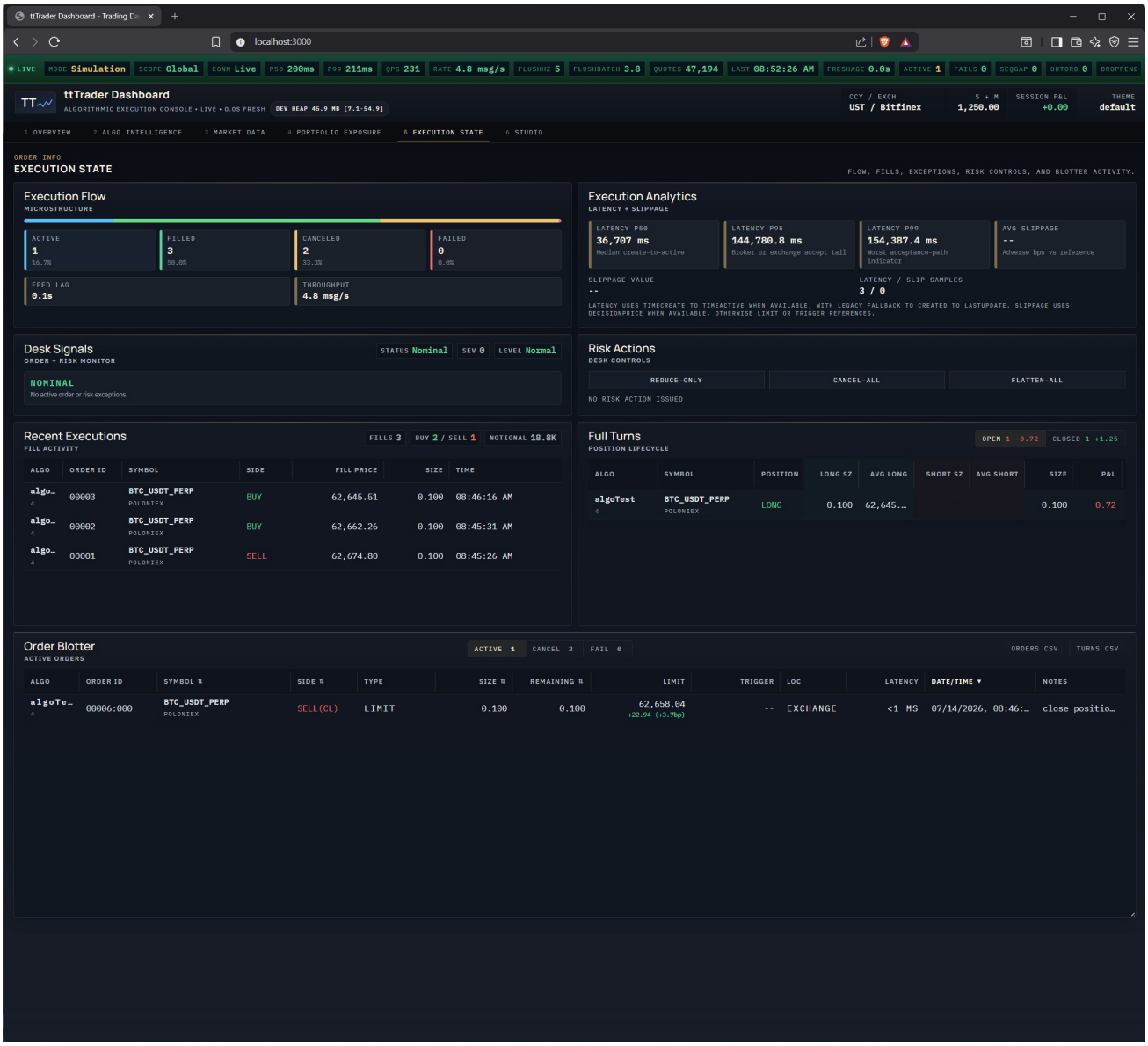



Figure 1.4: Execution state and TCA

## 1.6 Core Concepts

### 1.6.1 Event-driven architecture

Every inter-module interaction is an event posted to a typed event bus (`ttSubSystem/eventSystemBase.h`). No module holds direct pointers to another module's internals. Events carry two 64-bit payloads (`EV_VALUE = intptr_t`).

```
::postEvent( g_evHandles.get( ESRV_EXCHANGE ), EEV_GENERAL_EXIT_REQUEST );
```

Event ID ranges are spaced by `0x100` to avoid collisions:

| Prefix | Module | Range start |
|---|---|---|
| `EEV_GENERAL_*` | System / lifecycle | `EV_USER + 0x1000` |
| `EEV_2EXCHG_*` | Exchange plugins | `EV_USER + 0x1200` |
| `EEV_2ALG_*` | Trading algorithms | `EV_USER + 0x1300` |
| `EEV_2MD_*` | Marketdata manager | `EV_USER + 0x1600` |
| `EEV_2ORD_*` | Order manager | `EV_USER + 0x1700` |
| `EEV_2WSS_*` | WebSocket server | `EV_USER + 0x1800` |
| `EEV_2PB_*` | Playback engine | `EV_USER + 0x1900` |
| `EEV_2MGR_EXCHG_*` | Exchange manager | `EV_USER + 0x2000` |

Full reference: `01-system-protocol/`.

### 1.6.2 Manager lifecycle

Managers are created with the factory pattern (`new T`) and registered with the global handle table (`g_evHandles`) so other modules can address them. The `managerLifecycle_c<T>` RAII template wraps the pattern:

```
managerLifecycle_c<managerExchange_c> l_exchangeManager( "exchange", ESRV_EXCHANGE );
// init() called in constructor, exit() + delete in destructor
```

### 1.6.3 Stores

Stores are singletons (`singleton_c<T>` mixin) that hold runtime state loaded from JSON configuration. They follow the same init/exit pattern as managers, wrapped by `storeLifecycle_c<T>`:

```
storeLifecycle_c<storeExchange_c> l_exchangeStore( "exchange", &l_jsonConfig );
```

### 1.6.4 Plugin system

Exchanges, algorithms, and indicators are compiled as shared libraries and loaded at runtime through `storePlugin_c`. Each plugin exposes three C-linkage functions:

```
void*  createPlugin();           // returns eventHandler_c* (or algo/indicator base)
void   destroyPlugin( void* );   // cleanup
struct pluginConstInfo_s* getPluginConstInfo();  // name, type, description
```

### 1.6.5 Shutdown protocol

The exit sequence is a strict cascading waterfall, gated by confirmation events:

```
stop algos -> marketdata down -> order manager down -> exchange exit
-> plugin disconnect -> websocket down -> fTraderTerminated = true
```

Each step posts a confirmation event before the next step proceeds. The RAII destructor chain destroys stores and plugins only after `fTraderTerminated` is set, guaranteeing that all asynchronous WebSocket streams are fully torn down.

A three-layer defense prevents the exchange plugin from being destroyed while a WebSocket stream is still alive:

| Layer | Mechanism |
|---|---|
| Send guard | `_onEventRequestExit` checks `ENWS_STATUS_CONNECTED` before calling `_networkSend` |
| Deactivation gate | Only `_onEventCheckExit` may call `_setModuleActive(false)` — `_serverDisconnected` defers |
| Destructor safety | `~webSocketClient_c()` synchronously tears down any live stream via `_finalizeDisconnect()` |

Full specification: `01-system-protocol/`.

---

# 1.7 Conventions

## 1.7.1 Naming

| Element | Convention | Example |
|---|---|---|
| Classes | `lowerCamelCase_c` | `exchgConnectionBase_c` |
| Structs | `lowerCamelCase_s` | `exchangeConfigInfo_s` |
| Enums | `lowerCamelCase_e` | `executionMode_e` |
| Typedefs | `suffix _t` | `decimal_t` |
| Members | `m_` prefix | `m_fExitRequest` |
| Parameters | `p_` prefix | `p_fActive` |
| Locals | `l_` prefix | `l_ui32Module` |
| Statics | `s_` prefix | `s_ui64SubscriptionId` |
| Globals | `g_` prefix | `g_runtimeStatus` |
| Constexpr | `ce_` prefix | `ce_strName` |

## 1.7.2 Financial types

- **`decimal_t` is mandatory** for all trading, accounting, and market-data calculations. Derived from ttSubSystem, it is a 128-bit fixed-point type.
- `float`/`double` are permitted only for external API interop, physics simulations, or highly optimised numerical kernels. Every exception must be documented with a comment explaining the justification.
- Conversions between `decimal_t` and `float`/`double` must be explicit; no implicit casting.

## 1.7.3 Memory allocation

- `new`/`delete` and `malloc`/`free` are **forbidden** in application code.
- Use `st_malloc`/`st_free` (debug fast path) or `lt_malloc`/`lt_free` (low-latency production path) for dynamic allocation.
- Prefer stack allocation. Reserve vectors when size is known. Avoid unnecessary copies; prefer move semantics.
- `std::vector` is the default container. `std::array` for fixed sizes. `std::unordered_map` for lookup.

### 1.7.4 Concurrency

- RAII locks. Minimise lock scope. Avoid shared mutable state.
- No locks in hot execution paths. Prefer lock-free structures.
- `std::atomic` only when necessary; prefer immutable data.

### 1.7.5 Layout

- clang-format for all automatic formatting (`.clang-format` at project root).
- Lines <= 192 characters. If a line exceeds this, wrap with one argument per line.
- 1 empty line between functions in headers. 3 empty lines between functions in source files.
- Prefer early exits (guard clauses) to reduce nesting.

---

## 1.8 Exchange Integrations

| Exchange | Marketdata | Trading | Account | Notes |
|---|---|---|---|---|
| Binance | [x] | [x] | [x] | Futures stream style |
| Bitfinex | [x] | [x] | [x] | Reference adapter; BBO derived from book |
| Bybit | [x] (V5) | Skeleton | Skeleton | Public marketdata channels |
| Bitget | [x] (+ SBE binary) | Skeleton | Skeleton | Low-latency binary decode path |
| Kraken | [x] (WS v2) | Skeleton | Skeleton | Instrument-based channel model |
| OKX | [x] | Disabled | Disabled | Channel-based WebSocket |
| Aster | [x] (perps) | Disabled | Disabled | Trade/BBO/Book topics |
| Bitstamp | [x] | — | — | Diff-order-book with local merge |
| Interactive Brokers | [x] | [x] | [x] | Socket protocol (not WebSocket) |
| Paradex | [x] (perps) | — | — | Public trade/book channels |
| Hyperliquid | Header | — | — | — |
| Lighter | Header | — | — | DEX-layer protocol with ZK crypto |
| Phemex | Header | — | — | — |
| Poloniex | Header | — | — | — |

An adapter is a single header (`exchangeProtocol.h`). Capabilities are declared via `constexpr bool` flags. See `04-exchange-integrations/`.

### 1.8.1 Funding-rate feed

Perpetual/derivative venues expose a funding-rate feed that algos subscribe per instrument with `"funding": true` in the algo config's `instruments[]` entry. The feed delivers `singleFundingInfo_s` frames (rate payable at the next settlement, predicted next rate, settlement time and interval, plus the update reason: snapshot / rate change / settlement) through the `algoFramework_c::onFundingUpdate()` callback. Three transports are used, selected per venue and hidden behind the `Protocol::marketdata` hooks:

| Transport | Venues |
|---|---|
| Dedicated WS channel (`buildSubscribeFunding`) | Binance (`@markPrice`), OKX (`funding-rate`), Bybit (`tickers.`), Bitget (`ticker`), Aster (`@markPrice`), Hyperliquid (`activeAssetCtx`, hourly), Kraken futures (`ticker`, hourly), Phemex (`ticker_p`, 1e8-scaled), Poloniex (`ticker`) |
| Throttled REST poll (60 s, `buildFundingPoll`) | Lighter (`/api/v1/funding-rates`, per currency), dYdX (`/v4/perpetualMarkets`), Paradex (`/v1/markets`) |
| Extraction from existing frames (`extractFunding`) | Aevo (ticker frames) |

Not modeled (declared `fMarketdataFunding = false`): Bitfinex (positional status payload needs live verification), Bitstamp (spot only), Interactive Brokers (CME dated futures have no funding concept).

The certification suite covers the feed as the `Marketdata Funding` test (`CERTT_MD_FUNDING`): it verifies the subscribe-ack binding immediately and never waits for a settlement frame.

---

## 1.9 Trading Algorithms

| Algorithm | Type | Description |
|---|---|---|
| `algoTest` | Reference | Order lifecycle cycle test (create -> modify -> cancel) |
| `algoExchangeTest` | Certification | 27-test-case exchange certification suite |
| `algoMCT` | Production | Microstructural Coherence Trading — density-matrix purity signals with pyramiding |
| `algoMLD` | Production | Online-learning logistic model with regime-based sizing and deterministic champion fallback |
| `algoHUNT` | Production | Liquidity-hunt reversion with realtime NN meta-controller and counterfactual parameter memory |
| `algoHarvest` | Production | Level-reaction harvester: 5m fractal levels, session/slide gate, breath-cut, tiered peak ratchet, funding filter |

The algorithm framework (`algoFramework_c`) provides composable managers for assets, positions, orders, risk, timers, and stop-loss. Virtual callbacks deliver market-data ticks, order lifecycle events, and timer notifications.

Order lifecycle uses a single typed enum — `orderProcessingState_e` — split into active states (`idle`, `active`, pipeline create/modify/cancel hops) and absorbing terminal states (`executed`, `canceled`, `failed`, `expired`, `deleted`). Outgoing orders travel `algo -> manager -> exchange` and move `eProcessingState` through the matching hop; outcomes settle in `eState`. Numeric values of the legacy lifecycle states (1–7) are frozen for WebSocket wire compatibility.

See `05-trading-algorithms/README.md` and `03-architecture/README.md`.

---

## 1.10 Indicators

| Indicator | Type | Plugin |
|---|---|---|
| EMA | Exponential Moving Average | `indAvgEMA` |
| HMA | Hull Moving Average | `indAvgHMA` |
| SMA | Simple Moving Average | — |
| WMA | Weighted Moving Average | — |
| ZLHMA | Zero-Lag HMA | — |
| ATR | Average True Range | — |
| Average Bar Range | Bar range average | — |
| Bollinger Bands | Volatility bands | — |
| MACD | Moving Average Convergence/Divergence | — |
| RSI | Relative Strength Index | — |
| Trading Range | Range indicator | — |

Indicators attach to OHLC data series and expose per-bar `decimal_t` values through a uniform accessor interface.

See `06-indicators/README.md`.

---

## 1.11 Development Guide

### 1.11.1 Adding a new exchange adapter

1. Create `exchanges/exchg{Name}/` with `exchangeProtocol.h` and `CMakeLists.txt`.
2. Implement `exchangeProtocol_s` with `constexpr bool` capability flags.
3. Provide message classification, market-data parsers, and request builders.
4. Reference `exchgBitfinex/exchangeProtocol.h` as the canonical skeleton.

### 1.11.2 Adding a new trading algorithm

1. Create `algos/algo{Name}/` with `algoCore.h/.cpp`, `algoConfig.h/.cpp`, and `CMakeLists.txt`.
2. Derive from `algoFramework_c` and implement the required virtual callbacks.
3. Override `onMarketdataTick/BBO/Book`, `onOrder*`, and timer methods as needed.

### 1.11.3 Adding a new indicator

1. Implement a class deriving from `indicatorCore_c` in `include/ttTrader/indicator/`.
2. (Optional) Wrap it as a plugin DLL under `indicators/ind{Name}/`.

### 1.11.4 Coding conventions

All C++ code must follow the rules documented in the global instruction files (`00-formatting` through `90-hft`). The most important constraints are:

- **`decimal_t` for all financial math.** No `float`/`double` in trading logic.
- **`st_malloc`/`lt_malloc` for dynamic allocation.** No `new`/`delete`.
- **RAII for locks and resources.** No raw ownership.
- **Early exits (guard clauses).** Minimise nesting depth.
- **Rule of Zero.** Let the compiler generate special member functions.
- **No heap allocation in hot paths.**
- **Tests must be deterministic.** No shared state between tests.

---

# Chapter 2

# Storing API Keys & Secrets in the OS Credential Vault

Instead of hardcoding secrets in configuration files or environment variables, ttTrader can read API keys and secrets directly from your operating system's built-in credential store.

## 2.1 Supported Platforms

| Platform | Credential Store | CLI Tool |
|---|---|---|
| Windows | Windows Credential Manager | `cmdkey.exe` (built-in) |
| Linux | GNOME Keyring / KDE Wallet (via libsecret) | `secret-tool` (libsecret-tools package) |
| Linux (headless) | Kernel Keyring | `keyctl` (keyutils package) |

- No external vault server required.
- Secrets are encrypted at rest, sealed to your OS user account.
- Windows: secrets persist across reboots.
- Linux libsecret: persists across reboots (if keyring daemon auto-starts).
- Linux kernel keyring: does NOT survive reboots (unless using persistent keyring `@p`).

---

## 2.2 Naming Convention

Use a hierarchical, per-platform namespace matching how each OS tool works natively:

| Config reference | Windows target | Linux (secret-tool) | Linux (kernel keyring) |
|---|---|---|---|
| `cred:ttTrader/binance/apiKey` | `ttTrader/binance/apiKey` | `application=ttTrader, account=binance/apiKey` | `ttTrader:binance/apiKey` |
| `cred:ttTrader/binance/apiSecret` | `ttTrader/binance/apiSecret` | `application=ttTrader, account=binance/apiSecret` | `ttTrader:binance/apiSecret` |
| `cred:ttTrader:bybit/apiKey` | (Windows uses /) | — (secret-tool uses separate attributes) | `ttTrader:bybit/apiKey` |

The `cred:` target in your ttTrader config is the raw keyring key name — use the separator convention of the platform you're on.

## 2.3 Config Reference Syntax

`apiKeyRef` / `apiSecretRef` in `*.cfg` files accept three prefixes, resolved by `storeExchange_c::_resolveSecretRef` (`src/storeExchange.cpp`):

| Prefix | Meaning | Example |
|---|---|---|
| `cred:{name}` | OS credential store; `{name}` is the raw keyring target | `cred:ttTrader/lighter/account/apiSecret` |
| `env:{VAR}` | environment variable | `env:TT_BINANCE_API_KEY` |
| `vault://...` | **not supported** — legacy HashiCorp Vault syntax, removed with the Vault CLI | — |

### 2.3.1 Pitfalls

- **Never write `cred://...`** (double slash). The prefix match is a literal `cred:`, so everything after it is passed to the OS store as the target name. `cred://ttTrader/lighter/account/apiSecret` looks up `//ttTrader/lighter/account/apiSecret`, which is never stored, and `CredReadW` fails (`credentialStore.cpp`). The correct form is a single colon and no leading slashes: `cred:ttTrader/lighter/account`
- **`vault://secret/{name}` is dead config.** It was replaced by the native OS credential store; the app logs `failed to resolve apiKeyRef/apiSecretRef for <exchange>/<client>` and aborts. Migrate those entries to `cred:{name}` (or `env:{VAR}`).
- **The target must exactly match what was stored.** `readCredential` uses the name verbatim, so case and separators matter (`ttTrader/...`, not `tt-trader/...`).

---

## 2.4 Windows: Adding Secrets

### 2.4.1 Step 1 — Store a credential

Open **PowerShell** or **Command Prompt**:

```
cmdkey /generic:"ttTrader/exchange/account/apiKey" /user:"api-key" /pass:"YOUR_API_KEY_HERE"
cmdkey /generic:"ttTrader/exchange/account/apiSecret" /user:"api-secret" /pass:"YOUR_API_SECRET_HER
```

- `/user` is required by cmdkey but not used by ttSubSystem — any non-empty string works.
- `/pass` is the actual secret value.
- The target name uses forward slashes (`/`) matching the Credential Manager format.

### 2.4.2 Step 2 — Verify

**List a specific credential**:

```
cmdkey /list:ttTrader/exchange/account/apiKey
```

**List all ttTrader credentials at once** (PowerShell):

```
cmdkey /list | Select-String "ttTrader"
```

Expected output (either method):

```
Currently stored credentials for ttTrader/exchange/account/apiKey:
    Target: ttTrader/exchange/account/apiKey
```

```
Type: Generic
User: api-key
```

### 2.4.3 Step 3 — Update

Re-run the same `cmdkey /generic` command — it overwrites the existing entry.

### 2.4.4 Step 4 — Delete

```
cmdkey /delete:ttTrader/exchange/account/apiKey
```

### 2.4.5 Optional — Use the GUI

1. Open **Control Panel** -> **Credential Manager** -> **Windows Credentials**.
2. Click **Add a generic credential**.
3. Internet or network address: `ttTrader/exchange/account/apiKey`
4. User name: any string (e.g., `api-key`)
5. Password: your API key
6. Click **OK**.

---

## 2.5 Linux (Desktop): Adding Secrets via libsecret

### 2.5.1 Prerequisites

Install `libsecret-tools`:

| Distribution | Command |
|---|---|
| Ubuntu / Debian | `sudo apt install libsecret-tools` |
| Fedora / RHEL | `sudo dnf install libsecret` |
| Arch | `sudo pacman -S libsecret` |

Ensure your keyring daemon is running: - **GNOME**: `gnome-keyring-daemon` (auto-started on login) - **KDE**: `kwalletd5` or `kwalletd6`

### 2.5.2 Step 1 — Store a credential

```
secret-tool store --label="ttTrader Binance API Key" application ttTrader account binance/apiKey
```

You will be prompted to enter the secret. Paste your API key and press Enter.

```
secret-tool store --label="ttTrader Binance API Secret" application ttTrader account binance/apiSec
```

### 2.5.3 Step 2 — Verify

```
secret-tool lookup application ttTrader account binance/apiKey
```

Prints the stored secret to stdout.

### 2.5.4 Step 3 — Update

```
secret-tool clear application ttTrader account binance/apiKey
secret-tool store --label="ttTrader Binance API Key" application ttTrader account binance/apiKey
```

### 2.5.5 Step 4 — Delete

```
secret-tool clear application ttTrader account binance/apiKey
```

---

## 2.6 Linux (Headless): Adding Secrets via Kernel Keyring

The kernel keyring is available on every Linux system, even without a desktop environment.

### 2.6.1 Prerequisites

Install `keyutils`:

```
sudo apt install keyutils   # Debian/Ubuntu
sudo dnf install keyutils   # Fedora/RHEL
```

### 2.6.2 Step 1 — Store a credential

```
keyctl add user ttTrader:binance/apiKey "YOUR_API_KEY" @u
keyctl add user ttTrader:binance/apiSecret "YOUR_API_SECRET" @u
```

- @u = user keyring (persists for the session, lost on logout/reboot)
- @p = persistent keyring (survives reboots if kernel has `CONFIG_PERSISTENT_KEYRINGS=y`)

### 2.6.3 Step 2 — Verify

```
keyctl show @u
```

### 2.6.4 Step 3 — Read a secret

```
keyctl print $(keyctl search @u user ttTrader:binance/apiKey)
```

### 2.6.5 Step 4 — Delete

```
keyctl purge -p user ttTrader:binance/apiKey
```

### 2.6.6 Config reference format

When using kernel keyring, your config reference uses colon separators:

`cred:ttTrader:binance/apiKey`

---

## 2.7 Server Environments

If neither libsecret's D-Bus daemon nor the kernel keyring fit your server setup, use the existing `env:` prefix:

```
export TT_BINANCE_API_KEY="your-key"
export TT_BINANCE_API_SECRET="your-secret"
```

Reference in ttTrader config: `env:TT_BINANCE_API_KEY` and `env:TT_BINANCE_API_SECRET`.

---

## 2.8 Security Notes

- **Windows**: Credentials are AES-256 encrypted with your Windows account password as the derivation key. Only decryptable while you are logged in.
- **Linux (libsecret)**: Secrets are encrypted by the keyring daemon using your login password. Items in the default collection unlock at login.
- **Linux (kernel keyring)**: Secrets reside in kernel memory and are not written to disk (except persistent keyrings, which are encrypted on disk).
- **Process isolation**: Another process running as your user can read your credentials from the OS vault (same trust model as environment variables).
- **Service accounts**: If ttTrader runs as a Windows Service or Linux systemd unit under a different user account, store credentials for *that* account separately.
- **Cold startup**: On Linux headless servers using kernel keyring (@u), secrets are lost on reboot. Either use @p (persistent keyring) or the `env:` fallback.

# Part II

# Part II - Configuration

# Chapter 3

# Configuration

Reference for ttTrader run-configuration files: the top-level schema, the component sections, trading modes, credential references, per-asset balance shares, and the validation that aborts a bad config. The files are parsed at startup by the singleton stores in `src/stores/` and the exchange/ohlc loaders.

## 3.1 Where config files live

| Kind | Location | Example |
|---|---|---|
| Combined run config | exe directory (not in the repo) | `ttTraderDebug/algoHUNT-DELEV-QIS-HERD-MIRE-lighter-btc-perp.json` |
| Per-algo standalone config | `algos/<algoName>/config/` | `algos/algoHUNT/config/algoHUNT-lighter-btc-perp.json` |
| Per-exchange adapter config | `exchanges/exchg<Name>/exchange.cfg` | `exchanges/exchgLighter/exchange.cfg` |
| Repo default helper configs | `config/` | `config/algoExchangeTest.cfg` |

Run one with:

```
ttTrader.exe <config.json> [--tradingMode <live|paper|simulation|playback>] [--log] [--logmarketdata]
```

## 3.2 Format

- JSON with `//` line comments allowed (full-line comments are stripped by the parser; inline URL `//` is preserved).
- Most objects accept an `"enabled": true|false` flag; disabled entries are skipped, so one file can carry several variants.
- Unknown keys are generally ignored; malformed *values* fail the load (see Validation).

## 3.3 Top-level keys

| Key | Type | Purpose |
|---|---|---|
| `tradingMode` | string | `live` / `paper` / `simulation` / `playback`; selects the exchange `serverSettings` set and the order manager. `--tradingMode` overrides it |
| `cancelOpenOrdersOnExit` | bool | cancel resting open orders during shutdown |
| `cancelCloseOrdersOnExit` | bool | cancel resting close orders during shutdown |
| `closePositionsOnExit` | bool | flatten open positions during shutdown |
| `plugins` | array | plugin instances to load (`type`: `exchange` or `algo`) |
| `assets` | array | asset (balance) definitions |
| `exchanges` | array | exchange instances and their client sets |
| `instruments` | array | tradable instruments and their OHLC/indicator series |
| `algos` | array | algorithm instances |
| `logging` | object | optional structured-jsonl logging switches (`logRoot`, per-stream flags, levels, rotation, retention) |

## 3.4 Trading modes

`live` / `paper` / `simulation` / `playback` (`src/app/commandLine.cpp`, `configSystem.h`). The raw lowercased string selects the matching `serverSettings` set in `storeExchange_c`; `playback` falls back to the `simulation` set if it has none. Only the selected client set is parsed, so unselected production credentials never leave the credential store.

## 3.5 plugins

```
"plugins": [
    { "id": "exchgLighter", "enabled": true, "type": "exchange", "plugin": "exchgLighter" },
    { "id": 10, "enabled": true, "type": "algo", "plugin": "algoHUNT" }
]
```

An algo's `algoId` (in the `algos` section) must match a loaded `algo` plugin; exchange `id` must match an `exchanges` entry.

## 3.6 assets

Global balance definitions. An algo subscribes to the assets it lists (see below); balances arrive from the venue and drive sizing.

```
{ "id": "assetLighterUSDC", "enabled": true, "exchgId": "exchgLighter",
  "name": "USDC", "simMargin": 3000, "simSpot": 0 }
```

`simMargin` / `simSpot` seed the simulated margin/spot balances in simulation mode.

## 3.7 exchanges

```
{
    "id": "exchgLighter",
    "enabled": true,
```

```
    "serverSettings": {
        "simulation": [ { "type": "market", "clientUrl": "wss://...", "logPattern": "LGT-webs:0" }
        "paper":      [ { "type": "market", "clientUrl": "https://..." },
                        { "type": "trading", "clientUrl": "https://...", "clientId": 190, "keyIndex
                          "chainId": 300, "apiSecretRef": "cred:ttTrader/lighterTestnet/trading/api
                        { "type": "account", "clientUrl": "wss://...", "apiSecretRef": "cred:ttTrad
    }
}
```

- `serverSettings` maps a mode name (`live`/`paper`/`simulation`/`playback`) to its own client array; unknown keys warn, non-array values fail the load. Without `serverSettings`, a flat `"clients"` array loads the same way.
- Client entry `type` is `market` / `trading` / `account`.
- Common fields: `clientUrl`, `clientId`, `keyIndex`, `chainId`, `framePing`, `logPattern`, and credential references `apiKeyRef` / `apiSecretRef` / `apiPassphraseRef` (`cred:<scope>/<path>`).
- `tlsInsecureSkipVerify` is accepted at the exchange level.

## 3.8 instruments

```
{
    "id": "instLighterBTC", "enabled": true, "exchgId": "exchgLighter",
    "idAtExchg": "BTC", "info": "Lighter BTC/USDC perpetual future",
    "ohlc": [
        { "id": "instLighterBTC-1m", "source": "close", "timeFrame": 1,
          "historyDepth": 512, "gapFillEmpty": true,
          "indicators": [
              { "id": "instLighterBTC-1m-atr14", "type": "atr", "params": { "period": 14 } }
          ] }
    ]
}
```

- `idAtExchg` is the venue's symbol/market id; `exchgId` links to an exchange.
- `ohlc[]` declares OHLC series. `source` is `close` (or `trades`), `timeFrame` is minutes, `historyDepth` the retained bars (project limit 512), `gapFillEmpty` carries the last close through no-trade bars, and `indicators[]` attaches Indicator-V2 instances (`id`, `type`, `params`). An algo references these by id — see ../06-indicators/.

## 3.9 algos

```
{
    "algoId": 10, "id": "algoHUNT", "enabled": true, "name": "algoHUNT", "info": "algoHUNT",
    "parameter": { "instTradeId": "instLighterBTC", "ohlcId": "instLighterBTC-5m",
                   "risk": { "type": "PERCENT", "value": 0.20 }, "stateFile": "state/algoHUNT-BTCUS
    "assets": [ { "id": "assetLighterUSDC", "balanceFraction": 0.20 } ],
    "instruments": [ { "instrumentId": "instLighterBTC", "enabled": true, "book": 10,
                       "funding": false, "ohlc": [ "instLighterBTC-5m" ] } ]
}
```

- `algoId` matches the loaded algo plugin; `id`/`name`/`info` are labels.
- `parameter` is algo-specific (read in `onSetParams()` / the algo's `algoConfig.cpp`); `stateFile` is framework-managed persistence.
- `assets` lists the asset balances this algo may use and its share of each (see below).

- `instruments` lists the venue instruments, the order-book depth to subscribe (book), whether to subscribe funding, and the OHLC series to consume.

## 3.10 Balance shares and PnL isolation

Each algo declares its share **per asset** in its `assets` list:

```
"assets": [ { "id": "assetLighterUSDC", "balanceFraction": 0.20 } ]
```

`storeAlgo_c` resolves every share after all enabled algos are parsed:

- a configured `balanceFraction` is taken verbatim; `0` means listed but read-only (consumes no balance);
- entries without a value split the remaining budget (`1 - sum(explicit)`) equally;
- an over-allocated asset (`sum > 100 %`), or an unconfigured entry left with no budget, is a fatal config error: log + `requestApplicationExit()`;
- a bare asset id (`"assetLighterUSDC"`) is the same as an omitted value.

Sizing then uses a **first-snapshot allocation** (`startAvailable x share`) plus only that algo's own realized PnL — never the live account balance, so a losing algo cannot size against a winner's profit. Mid-run deposits/withdrawals are detected by `managerMain_c` and rescaled proportionally. Full behaviour: `../05-trading-algorithms/README.md#balance-shares-and-pnl-isolation`.

## 3.11 Validation and failure behaviour

A malformed config fails the load with a logged error (and, for balance-share errors, `requestApplicationExit()`):

- no enabled exchange;
- duplicate algo `id`;
- unknown plugin / plugin load failure;
- malformed exchange `serverSettings` (non-array set) or invalid client entry;
- invalid asset id, unknown asset reference, or duplicate asset in one algo;
- invalid `balanceFraction` (non-decimal, `< 0` or `> 1`), over-100 % for an asset, or an unconfigured entry with no budget left.

## 3.12 Minimal example

```
{
    "tradingMode": "simulation",
    "cancelOpenOrdersOnExit": true,
    "cancelCloseOrdersOnExit": true,
    "closePositionsOnExit": true,
    "plugins": [
        { "id": "exchgLighter", "enabled": true, "type": "exchange", "plugin": "exchgLighter" },
        { "id": 10, "enabled": true, "type": "algo", "plugin": "algoHUNT" }
    ],
    "assets": [
        { "id": "assetLighterUSDC", "enabled": true, "exchgId": "exchgLighter",
          "name": "USDC", "simMargin": 3000, "simSpot": 0 }
    ],
    "exchanges": [
        { "id": "exchgLighter", "enabled": true,
          "serverSettings": { "simulation": [
              { "type": "market", "clientUrl": "https://mainnet.zklighter.elliot.ai:443", "logPatte
```

```
    ],
    "instruments": [
        { "id": "instLighterBTC", "enabled": true, "exchgId": "exchgLighter", "idAtExchg": "BTC",
          "ohlc": [ { "id": "instLighterBTC-5m", "source": "close", "timeFrame": 5,
                      "historyDepth": 512, "gapFillEmpty": true,
                      "indicators": [ { "id": "instLighterBTC-5m-atr14", "type": "atr", "params": {
    ],
    "algos": [
        { "algoId": 10, "id": "algoHUNT", "enabled": true, "name": "algoHUNT", "info": "algoHUNT",
          "parameter": { "instTradeId": "instLighterBTC", "ohlcId": "instLighterBTC-5m",
                         "atrIndicatorId": "instLighterBTC-5m-atr14" },
          "assets": [ { "id": "assetLighterUSDC", "balanceFraction": 1.0 } ],
          "instruments": [ { "instrumentId": "instLighterBTC", "enabled": true, "book": 10,
                             "ohlc": [ "instLighterBTC-5m" ] } ] }
    ]
}
```

# Chapter 4

# config/

repo-level default configs for manual runs from the repository root (`.build/<preset>/ttTrader.exe --config config/<file>.cfg` runs with the repository root as working directory).

- `algoExchangeTest.cfg` — quick exchange certification run (algoExchangeTest against exchgLighter, simulation mode).
- `instrumentsIbkr.cfg` — IBKR instrument lookup helper (see algos/algoMLD/config/algoMLD-ibkr-nq-front.json header for the lookup procedure).

algo runtime configs live with their algo in `algos/<algoName>/config/`; per-exchange adapter configs live in `exchanges/exchg<Name>/exchange.cfg`. writable artifacts (logs, algo state, playback recordings) are created relative to the exe directory, never in the repository root.

Algo buying power is declared **per asset** in the algo's `assets` list (`{ "id": "assetLighterUSDC", "balanceFraction": 0.20 }`), not in `parameter`. `storeAlgo_c` resolves the shares across all enabled algos (unconfigured entries split the remaining budget equally; an over-allocated asset or an unconfigured entry with no budget aborts startup), and each algo then sizes from its first-snapshot allocation plus only its own realized PnL. See ../docs/05-trading-algorithms/README.md.

# Part III

# Part III - Exchanges

# Chapter 5

# Exchanges Overview

## 5.1 Adapter quick reference (September 2026)

All 16 adapters are **protocol-complete** and ship two configs each: `exchange.cfg` (the venue's exchanges entry) and `algoExchangeTest.cfg` — a complete certification run config for that venue (simulation mode, live marketdata, BTC-USD-PERP instrument + collateral asset + scenario wiring). Both live in the adapter folder `exchanges/exchg<Name>/`; insert the vault keys per `docs/exchange-certification-plan.md` and run. Certification state per venue, including the next pending gate, is tracked in `docs/exchange-certification-plan.md`. The `Kline` column marks whether the venue streams candles natively over websocket (`ws`), seeds the series over REST/TWS only (`seed`), or exposes no kline endpoint at all (–); see "Exchange-native kline streaming" below.

| Venue | Tier | Kline | Auth + private transport | Order ops (native / emulated) | Cert instrument | Keys |
|---|---|---|---|---|---|---|
| Lighter *(reference)* | — | ws | Poseidon2 / Goldilocks schnorr (L1 key); WS account_all feed + signed HTTP sendTx | native: stops, modify, GTD; emulated: trailing (hybrid ratchet), GAD | `BTC-USD` · USDC | 1 key |
| Poloniex | T0 [x] | ws | HMAC-SHA256 WS auth + signed HTTP; WS orders/trade channels + HTTP | native: create/cancel; emulated: stops, GTD, modify (cancel+replace) | `BTC_USDT_PERP` *(verify)* · USDT | 2 keys |
| Bitfinex | T0 [x] | ws | HMAC-SHA384 WS auth; single auth WS (`on/ou/oc`, chan-0 `te/tu`) | native: modify; emulated: stops, GTD/GAD | `tBTCF0:USTF0` · USDT | 2 keys |
| Interactive Brokers | T0 [x] | seed | TWS api session (clientId per sub-manager, no keys); TWS wire over raw tcp (`PLACE_ORDER` / `ORDER_STATUS`, one socket per module) | native: stops, modify (trailing ratcheted via modify); emulated: GTD/GAD | CME NES conid `908100745` (paper acct) · USD (paper) | no keys |
| Bybit | T1 [x] | ws | HMAC-SHA256 `GET/realtime` frame (both sockets); private ws streams + trade ws order entry | native: amend; emulated: stops, GTD/GAD | `BTCUSDT` (linear) · USDT | 2 keys |

| Venue | Tier | Kline | Auth + private transport | Order ops (native / emulated) | Cert instrument | Keys |
|---|---|---|---|---|---|---|
| Kraken Futures | T1 [x] | ws | REST challenge -> HMAC-SHA512 ws subscribe; openOrders ws stream + v3 signed HTTP | native: amend; emulated: stops, GTD/GAD | `PF_XBTUSD` · USD | 2 keys |
| OKX | T1 [x] | ws | login frame (HMAC-SHA256 + passphrase); private ws: orders/account/fills + ws ops | native: amend; emulated: stops, GTD/GAD | `BTC-USDT-SWAP` · USDT | 3 keys |
| Bitget | T1 [x] | ws | login frame + ACCESS-* headers; private ws channels + v2 mix HTTP | native: modify-order; emulated: stops, GTD/GAD | `BTCUSDT` · USDT | 3 keys |
| Binance | T1 [x] | ws | HMAC query signing (X-MBX-APIKEY); user-data ws (listenKey auto-minted at startup) | native: modify (PUT); emulated: stops, GTD/GAD | `BTCUSDT` · USDT | 2 keys |
| Aster | T2 [x] | ws | Binance-compatible HMAC; binance-style user-data ws (auto listenKey) | native: —; emulated: stops, GTD, modify | `BTCUSDT` · USDT | 2 keys |
| Bitstamp | T2 [x] | seed | v2 HMAC headers + v3 web-token ws; private-my_orders ws + v2 signed HTTP | native: —; emulated: stops, GTD, modify | no perps -> `BTC/USD` spot flows · USD | 3 keys |
| Paradex | T2 [x] | seed | STARK-curve SNIP-12 sigs -> JWT bearer; jsonrpc private ws + signed HTTP | native: modify; emulated: stops, GTD/GAD | `BTC-USD-PERP` · USDC | 2 keys |
| Hyperliquid | T3 [x] | ws | EIP-712 L1 action ECDSA (secp256k1); address-keyed user streams + signed POST /exchange | native: modify; emulated: stops, market, GTD/GAD | `BTC` · USDC | 2 keys |
| Phemex | T3 [x]* | ws | RSA/HMAC per endpoint class | native: —; emulated: all private flows | `BTCUSDT` · USDT | marketdata only |
| dYdX | T3 (new) | ws | EIP-712 ShortTermOrder + DYDX-* HMAC headers; v4_accounts ws (HMAC-signed subscribe) + signed HTTP | native: — (modify lands later); emulated: stops, GTD, modify | `BTC-USD` · USDC | 3 keys |
| Aevo | T3 (new) | — | EIP-712 Order + AEVO-KEY/SECRET HMAC headers; op-based private ws (orders/fills) + signed HTTP | native: —; emulated: stops, GTD, modify | `BTC-PERP` · USDC | 3 keys |

* Phemex is the one adapter still marketdata-only: its private stack needs RSA-SHA256 login + scaled-integer handling and lands as a dedicated change set after the first certification wave.

Findings worth knowing before certification:

- **Correlation is uniform**: every new stack correlates via the decimal client order id echoed by the venue (orderLinkId/clOrdId/clientOid/ client_id/newClientOrderId) resolved through the shared order store — no

adapter-local maps except Bitfinex's 32-bit cid indirection.

- **External fills** (liquidation/ADL) are logged and dropped everywhere; positions stay local truth derived from attributed executions.
- **Capability truth** lives in each adapter's `ce_capabilities`; simulation ignores it entirely (broker-side emulation).
- **Known testnet-phase verification items** are listed per venue in `docs/exchange-certification-plan.md` §4 (wire-field drift risk in parsers written before first live contact).

## 5.2 Exchange-native kline streaming (September 2026)

Every venue that exposes a kline feed streams candles natively: the adapter subscribes the venue's candle channel, classifies the frame as `EMSG_CANDLE` and feeds the series through `ohlcInfo::ingestStreamCandle` — candle series configured with `source: exchange` consume venue-streamed OHLC only, never candles generated locally from trades/BBO. The `algoExchangeTest` Marketdata Candles test validates exactly this stream (finalized/priced/monotonic closes) and self-skips for venues without a ws kline feed.

| Venue | ws candle channel | notes |
|---|---|---|
| Binance | `<symbol>@kline_<interval>` | dedicated candle socket; `x` flag marks venue-final bars |
| Aster | `<symbol>@kline_<interval>` | binance-compatible schema |
| Bybit | `kline.<interval>.<SYMBOL>` | symbol rides the topic, rows carry no instrument field |
| OKX | `candle<bar>` (`candle1m`, …) | `confirm` flag marks venue-final bars |
| Bitget | `candle<bar>` (`candle1m`, …) | `confirm` flag marks venue-final bars |
| Kraken | spot v2 `ohlc` channel | rides the second market ws client (wss://ws.kraken.com/v2); futures socket carries no candle feed |
| Bitfinex | public `candles` (`trade:<interval>:<pair>`) | one stream per instrument; chanId -> timeframe map bound on the ack |
| Poloniex | `candles_<timeframe>` (`candles_minute_1`) | one shared channel per timeframe; symbol rides the frame |
| Lighter | `candle/{MARKET_INDEX}/{resolution}` | resolution suffix derives the close time; up to two candles per rollover |
| Hyperliquid | `{type: "candle", coin, interval}` | history via POST /info `candleSnapshot` |
| Phemex | `kline_p.subscribe` | perp kline; the `kline.subscribe` object form is rejected (6001, live-verified) |
| dYdX | `v4_candles` | channel id `<market>/<resolution>`; live pushes carry a candles array |

Venues without a ws kline feed seed the series over REST/TWS and, where noted, aggregate live windows locally:

- **Bitstamp** — the ws v2 exposes no ohlc/kline channel: candles seed from REST only, live windows aggregate from trades/BBO (candle test self-skips).
- **Paradex** — `/v1/markets/{symbol}/candles` REST only, no ws channel.
- **InteractiveBrokers** — `REQ_HISTORICAL_DATA` over the TWS api wire.

- **Aevo** — the venue exposes no kline endpoint at all (natural skip).

Final-flag semantics: Binance (x), OKX and Bitget (`confirm`) mark venue-final bars on the wire; every other feed carries no final flag, so frames are provisional and the series watermark closes the bars.

## 5.3 Architecture (May 2026)

Each exchange now follows a **2-file design**:

```
exchanges/{Name}/
  exchangeProtocol.h    – struct exchangeProtocol_s with all wire-format logic + constexpr bool flags
   CMakeLists.txt         – build rules
```

Shared infrastructure in `shared/exchanges/`: - `exchgPlugin.cpp` / `exchgPlugin.h` — instantiates manager types, event maps, DLL entry points - `exchgManagerMainT.h` — template main manager (orchestrates sub-managers) - `exchgManagerMarketdataT.h` — template marketdata manager - `exchgManagerAccountT.h` — template account manager - `exchgManagerTradingT.h` — template trading manager - `exchgConnectionBase.h` — base class for websocket/http connection lifecycle

Venue capabilities are declared in a single constexpr `exchgCapabilities_s ce_capabilities` inside `exchangeProtocol_s` and exported across the DLL boundary via `getExchgCapabilities()` (see "Order-type capabilities" below). Protocol-behaviour flags remain individual constexpr bools (`ce_fUsesBinaryData`, `ce_fBookUpdatesBbo`, ...).

### 5.3.1 Structural rules

- Keep all protocol logic inside `exchangeProtocol.h`; no separate `.cpp` should be required.
- Use `// --- section ---` comments to separate each logical group.
- Keep public protocol methods together and place private helper functions at the end.
- Avoid `handleXXXX` names in protocol structs; use `processXXXX` consistently.
- Use `static constexpr` for flags and constants, and keep them grouped at the top.
- Put `static` runtime state at the bottom of the struct, with `inline` definition after the struct.
- Prefer empty stubs over missing functions for account/trading hooks so the shared managers compile against a complete protocol surface.

### 5.3.2 Example skeleton

```
struct exchangeProtocol_s
{
    // --- identity & capabilities ---
    static constexpr const char* ce_strName = "Example";
    static constexpr bool ce_fUsesNetworkWebSocket = true;

    // venue capability truth: exported at plugin load (getExchgCapabilities)
    // and consumed by the order manager; must mirror exactly what the wire
    // code in this file implements
    static constexpr exchgCapabilities_s ce_capabilities{
        .fMarketdata = true,
    };
    static_assert( exchgCapabilities_s::isValid( ce_capabilities ), "exchgCapabilities_s: invalid c

    // --- capability helpers ---
    static bool canExitAccount() { return true; }

    // --- error handling / logging ---
```

```
    static void processError( const jsonValue_c& );
    static bool shouldLogMessage( const exchgMessageClassify_s& p_result, logDirection2_e p_eDirect

    // --- message classification ---
    static exchgMessageClassify_s classifyMessage( const jsonValue_c& p_jsonValue );

    // --- session / control handlers ---
    static void processServerInfo( const jsonValue_c& p_jsonValue );
    static void processSubscribed( const jsonValue_c& p_jsonValue );
    static void processUnsubscribed( const jsonValue_c& p_jsonValue );

    // --- marketdata parsing ---
    static void processInstrumentDetails( const jsonValue_c& p_jsonValue );
    static std::vector<exchgInstrumentMsg_s> processUpdateTrades( const jsonValue_c& p_jsonValue );
    static void processTrade( const jsonValue_c& p_jsonValue, instrumentInfo_s* p_instrumentPtr );
    static void processUpdateBook( const jsonValue_c& p_jsonValue, instrumentInfo_s* p_instrumentPt
    static void processBookSnapshot( const jsonValue_c& p_jsonValue, instrumentInfo_s* p_instrument

    // --- account / trading parsing ---
    static void processWalletUpdate( const jsonValue_c& );
    static void processPositionUpdate( const jsonValue_c& );
    static void processOrderUpdate( const jsonValue_c& );
    static void processExecutionUpdate( const jsonValue_c& );

    // --- request builders ---
    static netBuffer_s* buildInstrumentDetailsRequest();
    static netBuffer_s* buildSubscribeMarketdataTrade( instrumentInfo_s* p_instrumentPtr );
    static netBuffer_s* buildSubscribeMarketdataBook( instrumentInfo_s* p_instrumentPtr );
    static netBuffer_s* buildUnsubscribeTrade( instrumentInfo_s* p_instrumentPtr );
    static netBuffer_s* buildUnsubscribeBook( instrumentInfo_s* p_instrumentPtr );
    static void buildHistoricalDataRequest( netBuffer_s* p_requestPtr, instrumentInfo_s* p_instrume

    // --- historical parsing ---
    static ohlcHistoricalInfo_s* processHistoricalData( const jsonValue_c& p_jsonValue, instrumentI

private:
    // --- private helpers ---
    static instrumentInfo_s* _getInstrumentFromChannel( const jsonValue_c& p_jsonValue );
    static const jsonValue_c* _extractBids( const jsonValue_c& p_jsonValue );

    // --- static protocol state ---
    static uint64_t s_ui64SubscriptionId;
};

inline uint64_t exchangeProtocol_s::s_ui64SubscriptionId = 1;
```

This gives a unique, repeatable structure for all exchange adapters in `exchanges/`.

## 5.4 HFT design rule

- when there is a choice between two API/function variants, always use the lower-latency variant.
- prefer direct websocket market-data channels over polling/http.
- avoid feeds that are explicitly aggregated or throttled unless no lower-latency option exists.

The same rule extends to private traffic (order create/cancel/modify, account/trading subscriptions): whenever

a venue exposes both a private WebSocket channel and a REST endpoint for an operation, the adapter must use the WebSocket path; a venue's own binary stream protocol counts as the lowest-latency path (IBKR orders ride the TWS api raw-tcp wire); signed HTTP is reserved for venues without a stream equivalent (Lighter `sendTx`, Poloniex `/v3/trade/order`) and bulk operations that are HTTP-only (fill-history gap replay, open-order/balance snapshots during startup recovery).

## 5.5 Connection Lifecycle

The diagram below covers the full lifecycle shared by all exchange adapters: WebSocket connect -> server handshake -> instrument discovery -> channel subscribe -> streaming data -> heartbeat keepalive -> unsubscribe -> disconnect -> reconnect.

### 5.5.1 Lifecycle states at a glance

| Step | Trigger | Key virtual / Protocol method | Notes |
|---|---|---|---|
| **Connect** | `start()` | `_serverConnected()` | Sets host details, posts `ENWE_P2S_CONNECT` |
| **Server Info** | `"event":"info"` JSON | `Protocol::processServerInfo()` | Only when `usesServerInfo = true` (Bitfinex) |
| **Instrument Discovery** | `_serverConnected()` | `Protocol::processInstrumentDetail()` / `processInstrumentSnapshot()` | HTTP GET or WebSocket instrument channel |
| **Subscribe** | instruments valid | `Protocol::buildSubscribe()` -> `processSubscribed()` | Message ID encoded with `ECMD_INSTRUMENT_SUBSCRIBE` |
| **Streaming Data** | push from exchange | `_processTrade()` / `_processUpdateBbo()` / `_processBook()` / `_processArray()` | Classified by `_classifyMessage` -> dispatched by `_dispatchClassifiedMessage` |
| **Heartbeat** | timer (every 2 ticks) | `Protocol::buildHeartbeat()` / `buildHeartbeatRaw()` | Ping/pong auto-handled by base class |
| **Unsubscribe** | `stop()` or instrument removal | `Protocol::buildUnsubscribe()` -> `processUnsubscribed()` | Clear chanId mappings, posts `EEV_2EXCHG_INTERNAL_CHECK_EXIT` |
| **Disconnect** | `_onEventCheckExit` | `_serverDisconnected()` -> `_bookClearAllState()` | Resets all marketdata + subscription state |
| **Reconnect** | unexpected disconnect | `_onEventTimer` increments counter -> `ENWE_P2S_CONNECT` | Loops back to Connect step |

## 5.6 Shutdown Architecture

The shutdown protocol follows a layered approach with a single deactivation gate to prevent the exchange plugin from being destroyed while WebSocket streams are still alive.

### 5.6.1 Shutdown sequence (top-level)

1. `managerMain_c::_onEventExitRequest` stops algos
2. algos confirm stop -> marketdata exit request posted
3. marketdata confirms down -> order manager exit request posted
4. order manager confirms down -> `EEV_GENERAL_EXIT_REQUEST` posted to exchange manager
5. exchange manager + plugins shut down (see below)
6. exchange confirms down -> WebSocket server exit request posted
7. WebSocket server confirms down -> `EEV_GENERAL_EXIT` -> `fTraderTerminated = true`
8. processing loop returns, RAII objects destroyed

Only after step 8 do the store lifecycle destructors destroy the plugin objects. Every step waits for a confirmation event before advancing; no step is skipped.

### 5.6.2 Plugin-level shutdown (within step 5)

### 5.6.3 Three-layer defense against premature destruction

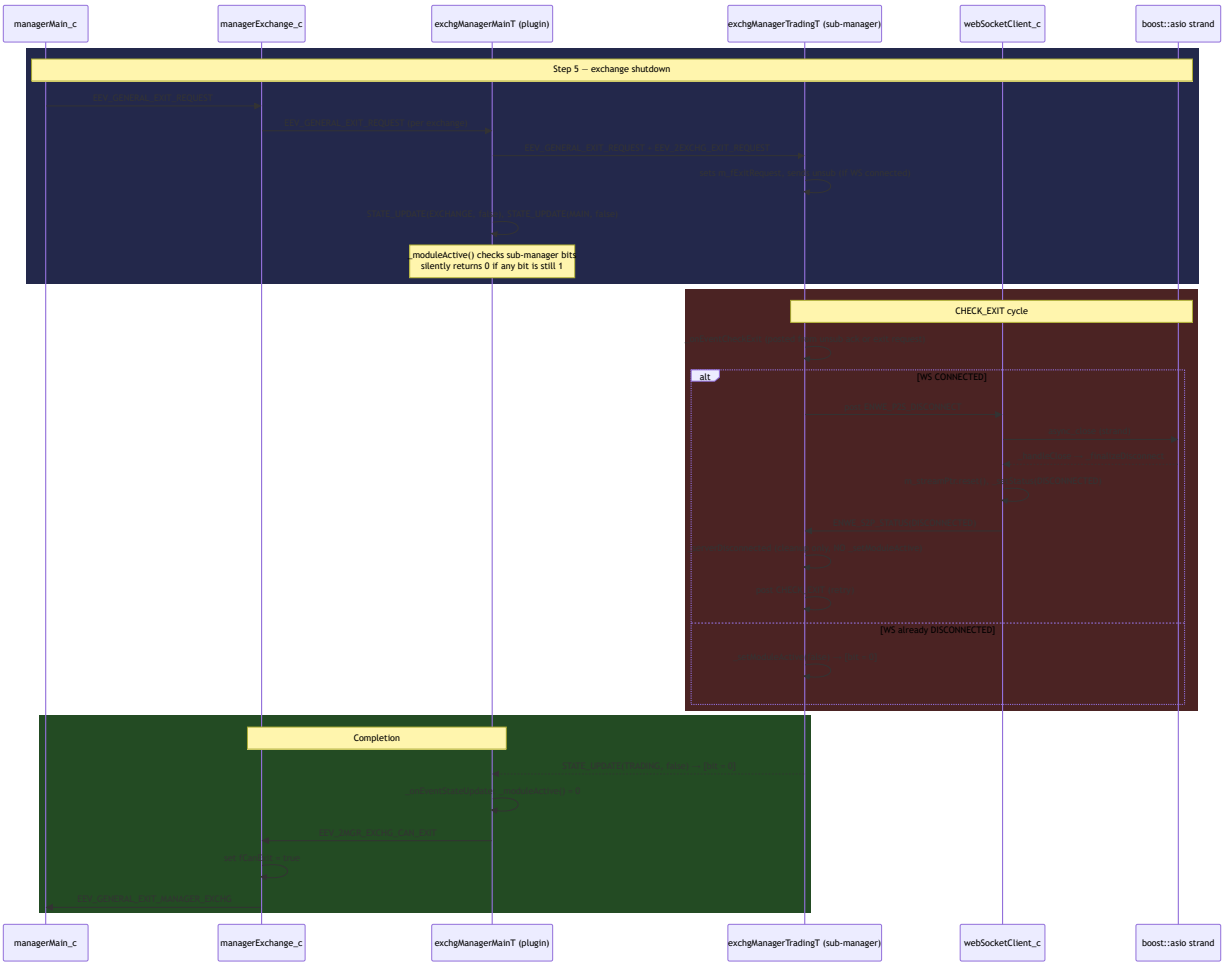
| Layer | Location | Mechanism |
|---|---|---|
| Send guard | `_onEventRequestExit` in all three managers | Checks `ENWS_STATUS_CONNECTED` before calling `_networkSend`. If no WS is connected, skips the unsub send and posts `CHECK_EXIT` directly. |
| Deactivation gate | `_serverDisconnected` in all three managers | Does NOT call `_setModuleActive(false)`. Local cleanup only (subscription state reset, force unsub). `CHECK_EXIT` is posted when `m_fExitRequest`, but the module bit remains 1 — the exit chain keeps waiting. |
| Destructor safety | `webSocketClient_c::~webSocketClient_c` | If `m_streamPtr` is not null (async close never completed, or forced termination), calls `_finalizeDisconnect()` synchronously to tear down the asio stream. |

### 5.6.4 Why the deactivation gate matters

`_serverDisconnected` fires on **any** WebSocket disconnection — including those that occur before the exit chain reaches the "check exit" stage. If it called `_setModuleActive(false)` directly, the module bit would be cleared early, allowing the exchange manager to post `CAN_EXIT` and destroy the plugin while the WebSocket stream (`m_streamPtr`) is still alive. The assert `_ASSERT(nullptr == m_streamPtr)` — a real guard against use-after-free or double-close — would fire because `async_close` on the asio strand had not completed by the time the destructor ran.

By restricting module deactivation to `_onEventCheckExit` — which only runs **after** the exit protocol has been initiated and the WS status transitions through the normal DISCONNECTING -> DISCONNECTED lifecycle — the exit chain is guaranteed to wait until the stream has been fully torn down.

### 5.6.5 Exit event flow (across managers)



### 5.6.6 Design rules

- `_setModuleActive(false)` may only be called from `_onEventCheckExit`.
- `_serverDisconnected` must NOT call `_setModuleActive(false)` — it clears local state and posts `CHECK_EXIT` (when `m_fExitRequest`), deferring the module deactivation to the gate.
- `_serverConnected` must call `_setModuleActive(true)` so the exit protocol knows a sub-manager may need cleanup.
- The destructor of `webSocketClient_c` must be defensive: any live stream must be torn down synchronously via `_finalizeDisconnect()`. All strand-based async completion handlers check for null `m_streamPtr` and return early, making the synchronous cleanup safe.

## 5.7 Architecture: Message Classification

As of May 2026, all exchange adapters use a unified message-type classification system:

1. **Base class** `exchgConnectionBase_c` provides a single dispatch switch in `_receivedDataObject` that routes every received JSON object by type (trade, BBO, book, ping, control, etc.). This one dispatch serves all three manager types (marketdata, trading, account).

2. **Template** `exchgManagerMarketdataT` chains base-class generic handling ("id", "op", "event") with exchange-specific routing via `Protocol::classifyMessage()`.

3. **Protocol structs** (`exchangeProtocol_s` in each adapter) implement `classifyMessage(json) -> exchgMessageClassify_s`. All exchange-specific JSON field logic ("topic", "e", "arg.channel", etc.) lives exclusively in each exchange's `exchangeProtocol.h` — never in `shared/exchanges/`.

4. **Fine-grained handlers** (`_processTrade`, `_processUpdateBbo`, `_processBook`, `_processControl`, etc.) replace the old catch-all `_processObject` 6-step if-chain.

5. **Private traffic** (trading/account managers) uses the same classification; order/execution/wallet/position messages route into the order-manager hub — see `docs/01-system-protocol/notification-pipeline.md`.

### 5.7.1 Message type enum

| Enum value | Purpose |
|---|---|
| `EMSG_SUBSCRIBE_ACK` | Subscription confirmed (via "id field") |
| `EMSG_UNSUBSCRIBE_ACK` | Unsubscription confirmed |
| `EMSG_PING` / `EMSG_PONG` | WebSocket keepalive (base class auto-responds) |
| `EMSG_CONTROL` | Exchange-specific control/ACK (routed to `preProcessObject`) |
| `EMSG_TRADE` | Public trade data |
| `EMSG_BBO` | Best bid/offer ticker |
| `EMSG_BOOK` | Order book snapshot or delta |
| `EMSG_CANDLE` | Exchange-native kline frame (venue-streamed OHLC, see "Exchange-native kline streaming") |
| `EMSG_INSTRUMENT` | Instrument metadata (Kraken WS path) |
| `EMSG_SERVER_INFO` | Server version/info (Bitfinex) |
| `EMSG_EXECUTION` / `EMSG_ORDER_UPDATE` / `EMSG_POSITION` | Private trading/account streams (routed through the order-manager hub) |

## 5.8 Order-type capabilities (stop / trailing stop)

Order-type and time-in-force capabilities are **declared by each adapter** in a single constexpr `exchgCapabilities_s ce_capabilities` beside the wire code that implements them (`exchangeProtocol.h`). They are exported across the DLL boundary once at plugin load (`getExchgCapabilities`, resolved by `storePlugin_c` beside `getPluginConstInfo`) and imported by `storeExchange_c` into the runtime consumption fields of `exchangeInfo_s` that the order managers and order handlers read.

- There are **no capability JSON keys anymore.** A legacy key (`nativeStopOrders`, `nativeTrailingStops`, `nativeOrderModify`, `nativeGoodTilDate`, `nativeGoodAfterDate`) inside an `exchanges[]` entry fails the config load with an error naming the key.
- Invalid combinations are compile-time errors: every adapter static_asserts `exchgCapabilities_s::isValid()` (trailing requires plain stops; stops/GAD require trading).
- The flags describe the remote venue for **production only**: in simulation (and playback) the platform itself is the venue, so the order manager always emulates stops broker-side, ratchets trailing stops locally and enforces the GTD/GAD boundaries itself — independent of declared capabilities.

| `fNativeStopOrders` | `fNativeTrailingStops` | Behavior of the order manager |
|---|---|---|
| `false` (default) | `false` (default) | All stops simulated broker-side: trigger evaluated locally on every BBO, order converts to plain MARKET/LIMIT on crossing and only then hits the exchange. Trailing delta ratcheted locally (one-way) and published to the algo/frontend on every move. |
| `true` | `false` | Plain stops native. Trailing stops = hybrid: placed as a native plain stop, the manager ratchets the trigger via exchange modify requests (one modify in flight per order, coalesced by its processing state). Used for Lighter. |
| `true` | `true` | Everything native: the protocol must consume `orderLimit_s::decTrailingDelta` on the wire. No adapter declares this today (InteractiveBrokers uses the hybrid ratchet over native modify) — flag an exchange `true/true` only together with its protocol change. |
| `false` | `true` | **Invalid.** Compile-time `static_assert` failure in the adapter. |

## 5.9 Time-in-force capabilities (GTD / GAD)

Declared in the same `ce_capabilities` instance:

| Flag | true | false (default) |
|---|---|---|
| `fNativeGoodTilDate` | The exchange pulls/cancels the order itself at its expiry (e.g. Lighter: chain-side GTD, cancel arrives as an order update; on-chain minimum and maximum expiry window apply at validation). | The order manager expires the order at the boundary: broker-side orders flip to `expired` locally; orders resting on the exchange are pulled through a normal cancel request. The exchange GTD window is not enforced at validation. |
| `fNativeGoodAfterDate` | Reserved for future protocol pass-through of `dtGoodAfterDate`. | The order manager holds the order in the deferred queue and re-submits it at activation (current behavior inherent to the flow). A GTD expiring before the GAD activation expires the order right out of the queue. |

## 5.10 Order modify capability

`fNativeOrderModify` steers whether the order manager sends in-place modify requests or emulates modification via cancel + replace:

- native (`true`, e.g. Lighter): `_orderModifyAlgo` passes through to `buildOrderModify`; partial fills do not apply.
- emulated (`false`): leg 1 cancels the venue order, the cancel ack resumes into a replace with the staged values clamped onto what is still open (partial-fill safe). No cancel/failed notification reaches algo/frontend for either leg. See `docs/01-system-protocol/notification-pipeline.md`.

Notes:

- A trailing stop is encoded as `EOIT_STOP_MARKET/EOIT_STOP_LIMIT` with `orderInfo_s::decTrailingDelta != 0` (see `orderInfo_s::isTrailingStop()`); there is no separate wire order type.
- The hybrid ratchet is one-way: a sell trigger only moves up, a buy trigger only moves down, both aligned to the instrument price grid.
- `protocol buildOrderModify` must carry the trigger on the wire for the hybrid mode (Lighter: `TriggerPrice`).

## 5.11 Current adapter status

Reality check against the trading/account integration plan (tiering: T0 repairs -> T3 last). "Skeleton" = empty stub functions behind the standard protocol surface.

| Exchange | Marketdata | Trading | Account | Tier / notes |
|---|---|---|---|---|
| Lighter | Implemented | Implemented (create/cancel/modify via sendTx) | Implemented (wallet + execution parsers) | Reference implementation; untouched by the plan |

| Exchange | Marketdata | Trading | Account | Tier / notes |
|---|---|---|---|---|
| Poloniex | Implemented | Create/cancel real (HTTP), execution parsing real over the private `trade` channel; modify emulated (cancel + replace) | Wallet real; asset fan-out through hub | T0 complete — sim cert next |
| Bitfinex | Implemented | Plain limit/market orders real over the auth WS (on/oc/ou); stops/GTD emulated | Wallet real (HMAC-SHA384 WS auth); fan-out through hub | T0 complete — sim cert next |
| InteractiveBrokers | Implemented (tick messages over the TWS api raw-tcp wire) | Order create/cancel/modify real over the TWS wire (PLACE_ORDER/ORDER_STATUS/EXECUTION_DATA, one api connection per sub-manager); native plain stops, hybrid trailing ratchet via modify | REQ_ACCT_DATA + REQ_ACCOUNT_SUMMARY + REQ_POSITIONS on the account cache, wallet through hub | T0+T2 complete — G1 sim [x] and G2 paper [x] certified 2026-08-28 (`cert-g1-sim.md`, `cert-g2-paper.md`); L2 book re-enables with the entitlement |
| Binance | Implemented (futures stream style) | Create/cancel/modify real (signed HTTP); pushes real on the user-data stream (listenKey minted at startup, url auto-attached) | ACCOUNT_UPDATE parsing real; wallet via the stream | T1 complete — sim cert next |
| Bybit | Implemented (V5 public marketdata) | Create/cancel/amend real over the trade ws; plain limit/market | Wallet/order/execution streams real over the private ws | T1 complete — sim cert next (spec capture in adapter README) |
| Kraken | Implemented (futures public ws feeds) | Signed HTTP sendorder/amendorder/cancelorders; native amend | Challenge-auth ws open_orders stream (placements/executions/cancels) | T1 complete — sim cert next (spec capture in adapter README) |
| OKX | Implemented (marketdata only) | WS ops order/cancel-order/amend-order; native amend | Login + orders/account/fills channels on the private ws | T1 complete — sim cert next (spec capture in adapter README) |
| Bitget | Implemented (futures public marketdata + SBE binary path) | Signed HTTP v2 mix endpoints (place/cancel/modify) | Login + orders/account/fill channels on the private ws | T1 complete — sim cert next (spec capture in adapter README) |
| Aster | Implemented (perps marketdata only) | Create/cancel real (signed HTTP, binance-compatible); modify emulated | ACCOUNT_UPDATE parsing real; listenKey minted at startup like Binance | T2 complete — sim cert next |

| Exchange | Marketdata | Trading | Account | Tier / notes |
|---|---|---|---|---|
| Bitstamp | Implemented (trade + diff_order_book) | Create/cancel real (v2 signed HTTP); modify emulated (no venue amend) | private-my_orders stream over the v3 web token (order states + fill transactions) | T2 complete — sim cert next; re-verify v3 canonicalization at testnet |
| Paradex | Implemented (WebSocket streaming, perps) | Create/cancel/modify real (SNIP-12 signed HTTP) | JWT ws orders/fills channels real | T2 complete — sim cert next (crypto offline-tested; wire fields re-verify at testnet) |
| Hyperliquid | Implemented (trades + l2Book ws) | Limit order/cancel/modify real (signed POST /exchange, L1 actions); market/trigger orders land with testnet hardening | orderUpdates/userFills streams keyed by derived wallet address | T3 code-complete — sim cert next (spec capture in adapter README) |
| Phemex | Implemented | Not implemented — RSA/HMAC signing module lands as a dedicated change set after the first certification wave | Not implemented | T3 — marketdata only |

Per-venue configs ship inside each adapter folder: `exchanges/exchg{Name}/exchange.cfg` (ready-to-paste entries, credentials via `cred:` refs only — see `docs/exchange-certification-plan.md` §2 for the vault insertion commands).

Capability bits per adapter live in each `exchangeProtocol_s::ce_capabilities`; they gate account/trading sub-manager startup (`exchgManagerMainT::_onEventConnect`) and steer order-manager emulation.

## 5.12 Marketdata feed characteristics

### 5.12.1 Binance

- trade: `<symbol>@trade` (realtime event feed)
- bbo: `<symbol>@bookTicker` (realtime top-of-book updates)
- book: `<symbol>@depth5|10|20@100ms`
- kline: `<symbol>@kline_<interval>` on the dedicated candle socket (x flag closes the bar)
- compression/aggregation: no adapter-side compression; book feed is exchange-throttled to 100ms updates.

### 5.12.2 Bybit (V5)

- trade: `publicTrade.<SYMBOL>`
- bbo: `tickers.<SYMBOL>` (contains bid1/ask1)
- book: `orderbook.<depth>.<SYMBOL>` with snapshot+delta merge
- kline: `kline.<interval>.<SYMBOL>` (symbol rides the topic)
- depth mapping in adapter: requested depth -> `1`, `50`, `200`, or `500`
- compression/aggregation: no adapter-side compression; cadence depends on Bybit topic push behavior.

### 5.12.3 Aster

- trade: <symbol>@trade
- bbo: <symbol>@bookTicker
- book: <symbol>@depth<5|10|20>@100ms
- depth mapping in adapter: requested depth maps to 5/10/20 (max level support 20)
- kline: <symbol>@kline_<interval> (binance-compatible schema)
- compression/aggregation: no adapter-side compression; cadence depends on exchange topic push behavior.

### 5.12.4 Bitget

- trade: publicTrade with instType=usdt-futures
- bbo: books1
- book: books50 (or books1 when local depth is 1)
- kline: candle<bar> (candle1m, ...) with confirm-flag finalization
- binary fast path: SBE payload support with template dispatch (trade/bbo/book templates)
- compression/aggregation: no adapter-side compression; updates are processed as push events.

## 5.13 Cross-adapter performance conventions

- exchange utility functions are centralized in each adapter's exchangeBase class (instead of free functions) to enable reuse across marketdata/account/trading managers.
- binary decode helpers are centralized in sbeDecoder_c (ttSubSystem) and used by adapters that support SBE binary feeds.
- variable-length binary symbol decode uses direct buffer pointers on the hot path to avoid string allocation/copy overhead where safe.
- shared instrument lookup uses hashed lookup paths to avoid per-message linear scans on symbol mapping.
- timestamp values passed to datetime_t::FromUnixTime are normalized to milliseconds at adapter boundaries.
- exchange adapter helpers are implemented as class members (private/protected), not free functions.
- parsing still validates bounds before data access to keep low-latency behavior without weakening safety checks.

### 5.13.1 Kraken (WebSocket v2)

- instrument metadata: instrument snapshot
- trade channel: trade
- bbo channel: ticker
- book channel: book (depth mapped to 10/100/1000)
- kline: spot v2 ohlc channel on the second market ws client (futures socket carries no candle feed)
- compression/aggregation: no adapter-side compression; local book is merged from snapshot/update stream payloads.

### 5.13.2 OKX

- trade channel: trades
- bbo channel: bbo-tbt (chosen for lower latency)
- book channel: books with snapshot+update handling
- kline: candle<bar> (candle1m, ...) with confirm-flag finalization
- compression/aggregation: no adapter-side compression; feed semantics follow selected OKX channel.

### 5.13.3 Bitfinex

- trade channel: trades

- book channel: book with `prec=P0`, `freq=F0`, `len=1|25`
- bbo: derived from best bid/ask of the local maintained book state
- kline: public `candles` channel (`trade:<interval>:<pair>`), one stream per instrument
- compression/aggregation: adapter requests Bitfinex bulk update mode (`conf` flag) to reduce overhead; still processed as incremental marketdata updates.

### 5.13.4 InteractiveBrokers

- trade and bbo are sourced from TWS TICK_PRICE/TICK_SIZE messages (bid 1/0, ask 2/3, last 4/5) merged into a two-sided overlay on the marketdata connection
- book is reserved for REQ_MARKET_DEPTH when the BookTrader entitlement lands (the subscribe builder returns null until then, so the suite skips the case)
- kline: seeded from TWS `REQ_HISTORICAL_DATA` over the api wire; no live candle feed
- compression/aggregation: no adapter-side compression; update cadence depends on IB market data service and entitlement.

## 5.14 Symbol format notes

- each exchange adapter still expects its own `strSymbolAtExchg` format.
- see each exchange-local README for examples and constraints.

# Chapter 6

# Aevo

Marketdata: implemented (perps WebSocket streaming: trades, throttled orderbook, book-ticker) Account: implemented (wallet discovery via GET /account; hmac-authenticated ws orders/fills channels) Trading: implemented (EIP-712-signed HTTP order create/cancel; modify surface present but not declared native) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading` true; no native stops/GTD/GAD and no native modify). Plain limit/market only. See docs/01-system-protocol/notification-pipeline.md.

## 6.1 Requirements capture

Sources (fetched September 2026): api-docs.aevo.xyz (llms.txt index; pages for GET /markets, POST /orders, POST+DELETE /orders/{order_id}, GET /auth, GET /account, the ws channel pages trades/orderbook-throttled/book-ticker/ orders/fills and websocket-authentication), github.com/aevoxyz/aevo-sdk (eip712_structs.py, aevo.py, order examples). Re-verify wire fields at testnet sign-off.

### 6.1.1 Crypto stack (`aevoAuth.h`)

- HMAC-SHA256 (offline-tested by the `AevoAuth` suite against RFC 4231 vectors 1-3) for the ws one-off auth signature
- EIP-712 typed hashing over keccak256: domain separator over `EIP712Domain(string name,string version,uint256 chainId)` (canonical assembly, no 0x1901 in the preimage), Order struct hash per the SDK definition order; canonical EtherMail spec vector pinned in the test (domain separator, digest)
- secp256k1 deterministic-k ECDSA via `CEip712Signing` (subsystem; same swapped-argument call shape as hyperliquidAuth.h), 65-byte r||s||v hex signature with 0x prefix
- signed Order struct (SDK `eip712_structs.py::Order`, definition order): `Order(address maker,bool isBuy,uint256 limitPrice,uint256 amount,uint256 salt,uint256 instrument,uint256 timestamp)`
- signing domain (SDK `CONFIG`): mainnet `{name:"Aevo Mainnet", version:"1", chainId:1}`, testnet `{name:"Aevo Testnet", version:"1", chainId:11155111}`; selected from the `testnet` marker in the client URL

### 6.1.2 Endpoints & flow

| Purpose | URL |
|---|---|
| Public ws | `wss://ws.aevo.xyz` |

| Purpose | URL |
|---|---|
| Private ws | same socket; one-off auth op frame, then orders/fills subscriptions |
| Wallet discovery | GET /account — the account field is the wallet/maker address |
| Orders | POST /orders, cancel DELETE /orders/{order_id}, edit POST /orders/{order_id} |
| Instruments | GET /markets?instrument_type=PERPETUAL (the venue rejects PERP with INSTRUMENT_TYPE_INVALID — first G1 run, fixed 2026-09-06) |

REST private requests carry AEVO-KEY + AEVO-SECRET headers (openapi securitySchemes on every private endpoint + aevo.py rest_headers).

WS frames are op-based: requests {"op":"subscribe|unsubscribe|auth", "data":[...]}, subscription payloads {"channel":"...","data":{...}}. Public channels used: trades:INSTRUMENT, orderbook-100ms:INSTRUMENT (initial type:"snapshot" frame, then diffs; a zero size removes the level; rows are [price, amount, iv] strings — iv is dropped for perps), book-ticker:INSTRUMENT (answers arrive on the ticker:INSTRUMENT channel per the docs). Private channels: orders, fills (subscribed as plain channel strings after the auth ack).

Startup sequence: account ws defers -> http client fetches GET /account ({"account":...} routes through EMSG_AUTHENTICATION -> processLoginAck stashes the wallet) -> deferred socket connects -> one-off hmac auth frame ({"op":"auth","data":{key,timestamp,signature}}, signature = HMAC_SHA256( secret, "key,timestamp,ws,auth,")) -> chained orders / fills subscriptions (buildFollowupSubscribes). A private-socket reconnect re-arms the auth frame and the subscription chain (serverConnected resets the followup state on wss clients).

Timestamps are unix nanoseconds everywhere on the ws (dateTime_t auto-detect converts); order signing timestamps are unix seconds.

### 6.1.3 Credentials

- apiKeyRef: Aevo api key (AEVO-KEY)
- apiSecretRef: Aevo api secret (AEVO-SECRET + ws HMAC key material)
- apiPassphraseRef: order-signing key private key (64 hex chars) — the passphrase is not used by the verified header scheme, so the slot carries the EIP-712 signing key instead (see uncertainties)

### 6.1.4 Order correlation

Aevo order ids are 32-byte hex hashes of the signed order payload (0x...), not uint64. The adapter keys orders by the first 8 bytes of the hash (orderInfo_s::infoId.ui64ExchgId) and keeps a ui64ExchgId -> "0x..." registry in the trading channel: create acks bind it, cancel/modify targets read from it, private pushes resolve through the order store by the same 8-byte key. salt = our decimal client order id (uniqueness entropy per the SDK; the venue does not echo it, and there is no client-order-id field on POST /orders). Edit responses carry a NEW order id — the registry re-binds and the stored exchg id is replaced; foreign or stale pushes are logged and dropped.

The venue numeric instrument id (needed by the signed Order struct and the order payload) rides the free-form runtimeInfo.strMarketName slot, filled during instrument discovery.

### 6.1.5 No kline/candle endpoint

Aevo exposes no OHLC/kline REST or ws endpoint (only /mark-history and /index-history, which are not candle series). No candle hooks are implemented (buildSubscribeCandle/processUpdateCandle/ build-

HistoricalDataRequest/processHistoricalData absent) — the candle certification test is a natural skip. Historical series seeding falls back to the framework default (see exchgManagerMarketdataT _onEventLoadHistorical).

### 6.1.6 Trade-flow sparsity and bbo timestamp (observed at first live G1 run)

- The adapter subscribes trades:BTC-PERP correctly, but the venue's perp trade flow is **intermittent**: zero trade records during the 21:57 first-pass window (2026-09-06), trades present ~40 minutes later. algoExchangeTest.cfg runs failFast: false so a trade-silent window costs only test 01, not the whole suite.
- **bbo timestamp**: the per-market book-ticker ticker object carries no timestamp; the venue sends it on the envelope's data object in **nanoseconds** ("timestamp":"1788726900294129831"). The adapter stashes the envelope value and normalizes to milliseconds at the boundary — the first run failed test 02 ("bbo: missing timestamp") before this was fixed.

## 6.2 Testnet

Full testnet environment (separate signing domain + chain id):

| Purpose | URL |
|---|---|
| WS | wss://ws-testnet.aevo.xyz |
| REST | https://api-testnet.aevo.xyz |

The signing domain flips on the testnet marker in the client URL (Aevo Testnet / chainId 11155111, sepolia). Point the cfg clientUrls at the testnet hosts for the run.

## 6.3 Uncertainties (unverified against a live venue)

- wallet discovery is one-shot per process: if the GET /account round trip fails at the network level the adapter logs the error but does not re-issue it (Paradex jwt-mint parity); a restart re-arms it. a retry hook is a central-framework decision.
- ws ack frame shapes: the docs do not document the response to auth / subscribe ops (the SDK sends them and never parses the reply). The classifier routes every frame with an op or success field to the account-side control chain and ignores unrelated acks; if the real ack carries neither field it lands in the unknown-message log instead.
- the task-brief REST header scheme (AEVO-TIMESTAMP, AEVO-KEY, AEVO-PASSPHRASE, AEVO-SIGNATURE) could not be verified in any official source; the openapi + SDK both use AEVO-KEY + AEVO-SECRET. If the venue starts requiring the passphrase scheme, the credential mapping must change (the signing key currently rides apiPassphraseRef).
- market-order price convention: the SDK uses limit_price = 0 for market sells and 2^256-1 for market buys; the SDK's own market-buy path contains a scaling bug (limit_price * 10^6 overflows uint256), so the unscaled ceiling value is emitted and needs a testnet verdict.
- book-ticker subscriptions answering on the ticker: channel is taken from the docs' response example only; if the venue echoes the request channel name instead, both heads are routed anyway.
- private channel payloads (orders/fills) match the documented object shapes; unknown order_status values are logged, not asserted.
- fees: /markets carries no fee structure (unlike Paradex), so the runtimeInfo fee defaults stay until GET /account fee_structures parsing is wired centrally.
- ce_i32MaxBookDepth 50 is a retention cap choice; the venue streams the full book and documents no depth limit.
- iv element in orderbook rows is assumed absent/zero for perps and is dropped without validation.

# Chapter 7

# Aster

2-file design: exchangeProtocol.h + CMakeLists.txt

Shared infrastructure in shared/exchanges/ - exchgPlugin.cpp/h, exchgManagerMainT.h, exchgManager*T.h templates

Protocol: exchangeProtocol_s with constexpr bool flags

Marketdata: implemented (perps marketdata only) Account: stubs Trading: stubs Tier: T2 of the integration plan (Binance-style HMAC auth)

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata=true` only); account/trading sub-manager startup is gated on those bits. See docs/01-system-protocol/notification-pipeline.md.

## 7.1 exchangeProtocol_s organization

Use the shared protocol block ordering so adapters stay consistent and easy to maintain: `exchanges/exchgLighter/exchang` is the functional reference implementation; `exchanges/exchgBitfinex/exchangeProtocol.h` the layout skeleton. All adapters should keep the same block order and expose empty account/trading stubs when a complete shared-manager skeleton is required. Instrument discovery and historical parsing are intentionally separate from live marketdata parsing.

1. identity and capability flags
2. message logging and classification
3. session/control parsers (`processServerInfo`, `processSubscribed`, `processUnsubscribed`, `processLoginAck` when used)
4. marketdata parsers (`processInstrumentDetails`, `processTrade`, `processBbo`, `processUpdateBook`, `processBookSnapshot`)
5. account/trading parsers
6. request builders (instrument discovery, subscribe/unsubscribe, heartbeat/auth, historical)
7. historical response parser
8. private helper methods
9. static protocol state

Naming convention: protocol receive/ack handlers use `processXXXX` (not `handleXXXX`).

## 7.2 Historical data

`buildHistoricalDataRequest` fetches the binance-compatible `/fapi/v1/klines` with `limit=_OHLC_ITEM_REQUEST + 2` (512). The fapi klines cap of 1500 applies here as on Binance (error -1130 beyond it); 512 fits with no clamp.

## 7.3 Testnet

No public testnet known for the Aster fapi-compatible stack. Gate G2 falls back to minimum-size production validation; confirm with the venue whether a demo environment has been added before relying on production for first contact.

# Chapter 8

# Binance

2-file design: exchangeProtocol.h + CMakeLists.txt

Shared infrastructure in shared/exchanges/ - exchgPlugin.cpp/h, exchgManagerMainT.h, exchgManager*T.h templates

Protocol: exchangeProtocol_s with constexpr bool flags

Marketdata: implemented Account: stubs (tier T1 — full private stack is new work, not a repair) Trading: stubs (tier T1 — full private stack is new work, not a repair)

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (fMarketdata=true only); account/trading sub-manager startup is gated on those bits. See docs/01-system-protocol/notification-pipeline.md.

## 8.1 exchangeProtocol_s organization

Use the shared protocol block ordering so adapters stay consistent and easy to maintain: `exchanges/exchgLighter/exchang` is the functional reference implementation; `exchanges/exchgBitfinex/exchangeProtocol.h` the layout skeleton. All adapters should keep the same block order and expose empty account/trading stubs when a complete shared-manager skeleton is required. Instrument discovery and historical parsing are intentionally separate from live marketdata parsing.

1. identity and capability flags
2. message logging and classification
3. session/control parsers (`processServerInfo`, `processSubscribed`, `processUnsubscribed`, `processLoginAck` when used)
4. marketdata parsers (`processInstrumentDetails`, `processTrade`, `processBbo`, `processUpdateBook`, `processBookSnapshot`)
5. account/trading parsers
6. request builders (instrument discovery, subscribe/unsubscribe, heartbeat/auth, historical)
7. historical response parser
8. private helper methods
9. static protocol state

Naming convention: protocol receive/ack handlers use `processXXXX` (not `handleXXXX`).

## 8.2 Historical data

`buildHistoricalDataRequest` fetches `/fapi/v1/klines` with `limit=_OHLC_ITEM_REQUEST + 2` (512). fapi klines reject `limit > 1500` with error -1130 ("Data sent for parameter 'limit' is not valid") — the original shared

default of 1538 tripped exactly that reject. 512 fits the cap with no clamp; keep the venue cap in mind if the shared default ever grows.

## 8.3 Testnet

Binance USD(S)-M futures testnet with separate test credentials:

| Purpose | URL |
|---|---|
| Public/user ws | `wss://stream.testnet.binancefuture.com/ws` |
| REST (orders, listenKey) | `https://testnet.binancefuture.com` |

The listenKey mint flow is identical (header-only POST). Keys come from testnet.binancefuture.com.

# Chapter 9

# Bitfinex

Bitfinex uses the standard exchange adapter layout and serves as the layout reference for a marketdata + account + trading exchange adapter (the functional reference implementation is Lighter):

```
exchanges/Bitfinex/
  exchangeProtocol.h
  CMakeLists.txt
```

## 9.1 Implementation status

- Marketdata: implemented
- Account: implemented (HMAC-SHA384 WS auth + wallet parsing; wallet updates route through the order-manager hub)
- Trading: plain limit/market orders over the authenticated ws — create `[0,"on",...]`, cancel `[0,"oc",...]`, native in-place modify `[0,"ou",...]`; order state via `os`/`on`/`ou`/`oc`, executions via chan-0 `te`/`tu` (correlated through the static cid map); stops/GTD/GAD emulated by the order manager

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading`, native modify true). Config: `exchange.cfg` in this folder. See docs/01-system-protocol/notification-pipeline.md.

## 9.2 Shared infrastructure

Shared manager/template infrastructure lives in `shared/exchanges/`:

- `exchgPlugin.cpp` / `exchgPlugin.h`
- `exchgManagerMainT.h`
- `exchgManagerMarketdataT.h`
- `exchgManagerAccountT.h`
- `exchgManagerTradingT.h`
- `exchgConnectionBase.h`

## 9.3 exchangeProtocol_s organization

`exchangeProtocol.h` is organized into logical blocks to keep all adapters similar and maintainable:

1. Identity and capabilities
2. Message classification and logging filter
3. Server/session handlers (server info, subscribe/unsubscribe, auth ack)
4. Marketdata parsers (instrument details, trade, BBO, book)

5. Account/trading parsers
6. Request builders (instrument discovery, subscribe/unsubscribe, heartbeat/auth, historical)
7. Historical response parsing
8. Private helper methods
9. Static state

## 9.4 Testnet

Bitfinex does not offer a public testnet for the production websocket API. Gate G2 falls back to minimum-size production validation; exercise on/ou/oc packet handling carefully on the first live pass and record deviations here.

# Chapter 10

# Bitget (v2)

Marketdata: implemented (futures public channels + SBE binary decode) Account: implemented (login -> orders/account/fill channels on the private ws) Trading: implemented (signed HTTP v2 mix endpoints; modify via modify-order) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading` true; no native stops/GTD/GAD). Plain limit/market only. See docs/01-system-protocol/notification-pipeline.md.

## 10.1 Requirements capture (bitget v2)

Source: bitgetlimited.github.io/apidoc/en/mix (websocket private + trade REST). Captured August 2026; re-verify before testnet sign-off.

### 10.1.1 Endpoints

| Purpose | URL |
|---|---|
| Public ws (+ SBE binary) | `wss://ws.bitget.com/v2/ws/public` |
| Private ws | `wss://ws.bitget.com/v2/ws/private` |
| REST trading | `https://api.bitget.com/api/v2/mix/order/*` |

### 10.1.2 Authentication

- login frame: `{"op":"login","args":[{"apiKey","passphrase", "timestamp":<ms>,"sign"}]}` with `sign = base64( HMAC-SHA256( secret, timestampMs + "GET/user/verify" ) )`
- REST headers: `ACCESS-KEY`, `ACCESS-SIGN` = base64( HMAC-SHA256( secret, timestampMs + "POST" + requestPath + body ) ), `ACCESS-TIMESTAMP`, `ACCESS-PASSPHRASE`
- passphrase is the third credential — config key `apiPassphraseRef`

### 10.1.3 Streams (private ws, instType USDT-FUTURES)

- `orders`: `data[]` with `orderId`, `clientOid` (= our client id), `status` (`new`, `partial_fill`, `filled`, `canceled`)
- `account`: per marginCoin balances (`accountAvailable`, `accountEquity`, `locked`) -> routed through the hub
- `fill`: executions with `fillId`, `price`, `size`, `fillFee`

### 10.1.4 Order entry (signed HTTP)

place-order / cancel-order (explicit `orderIdList/clientOidList`) / modify-order (`orderId`, `newSize`, `newPrice`). Ack `code == "00000"` binds the venue id; authoritative state arrives on the orders channel.

## 10.2 Testnet

Bitget offers **demo trading** (separate demo product set + session flag) rather than distinct hostnames. Pin the exact mechanism (demo header / product prefix) against bitgetlimited.github.io/apidoc during G2 prep and record it here before running the scenario set.

# Chapter 11

# Bitstamp

Marketdata: implemented (trade channel + diff_order_book with local book merge) Account: implemented (v3 web-token auth -> private-my_orders stream) Trading: implemented (v2-signed HTTP order new/cancel; modify emulated) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading` true; no native stops/GTD/GAD/amend). Plain limit/market only. See docs/01-system-protocol/notification-pipeline.md.

## 11.1 Requirements capture

Source: www.bitstamp.net/api — v2 REST signing, v3 real-time services (web token authorization), websocket push documentation. Captured August 2026; **the v3 signature canonicalization must be re-verified against live docs before testnet sign-off** (verb\npath\nquery\ncontent-type\nnonce\ntime- stamp\nversion\nbody).

### 11.1.1 Endpoints

| Purpose | URL |
|---|---|
| Public ws | `wss://ws.bitstamp.net` |
| Private ws channels | `private-my_orders` (order states + fill transactions) |
| REST order entry | `https://www.bitstamp.net/api/v2/order/new/` etc. |
| WS auth token | `POST /api/v3/web_tokens/authorization/` |

### 11.1.2 Authentication

- REST v2 headers: `key`, `X-AUTH-SIGNATURE` = hex( HMAC-SHA256( secret, nonce + customer_id + api_key [+ body] ) ), `X-AUTH-NONCE`, `X-AUTH-TIMESTAMP` (µs); urlencoded bodies
- WS: one-time token from the v3 authorization endpoint, then `{"event":"bts:subscribe","data":{"channel":"pri my_orders", "auth_token":...}}`
- credentials: apiKeyRef / apiSecretRef / **customer id via `apiPassphraseRef`** (third credential slot)

### 11.1.3 Streams

- `private-my_orders`: array of live orders with `id`, `status` (`open`, `in_process`, `finished`, `canceled`) and, on completion, `transactions[]` with price/amount/fee per fill. Correlation runs through the store by venue order id (bound at the create ack).

### 11.1.4 Order entry (signed HTTP)

limit: `POST /api/v2/order/new/` (`pair`, `amount`, `price`, `type`); market: `/order/new/market/` (no price). cancel by venue `id`. No venue amend — modify is emulated cancel+replace.

## 11.2 Testnet

Bitstamp does not offer a public testnet. Gate G2 falls back to minimum-size production validation - prioritize verifying the v3 web-token signature canonicalization on the very first call (a wrong canonicalization fails cleanly with an authentication error, which is a safe probe).

# Chapter 12

# Bybit (v5)

2-file design: exchangeProtocol.h + CMakeLists.txt

Shared infrastructure in shared/exchanges/ - exchgPlugin.cpp/h, exchgManagerMainT.h, exchgManager*T.h templates

Marketdata: implemented Account: implemented (wallet/order/execution streams over the private ws) Trading: implemented (order entry over the trade ws; native amend) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading`, native modify true). Plain limit/market only: stops, trailing and GTD/GAD are emulated by the order manager. See docs/01-system-protocol/notification-pipeline.md.

## 12.1 Requirements capture (from official v5 docs)

Source: bybit-exchange.github.io/docs/v5 — `ws/connect`, `websocket/trade/guideline`, `websocket/private/{order,ex`
Captured August 2026; re-verify on protocol changes.

### 12.1.1 Endpoints

| Purpose | URL |
|---|---|
| Public marketdata | `wss://stream.bybit.com/v5/public/linear` (per category) |
| Private stream | `wss://stream.bybit.com/v5/private` |
| Order entry | `wss://stream.bybit.com/v5/trade` |
| REST fallback / discovery | `https://api.bybit.com` |

### 12.1.2 Authentication (both private sockets, identical scheme)

- signature = hex( HMAC-SHA256( apiSecret, `"GET/realtime" + expiresMs` ) ), expiresMs = now + 5000
- frame: `{"op":"auth","args":[apiKey, expiresMs, signature]}`
- acks differ per socket:
  - private: `{"success":true,"op":"auth"}`
  - trade: `{"retCode":0,"retMsg":"OK","op":"auth"}`
- heartbeat: client `{"op":"ping"}` every ~20 s (adapter sends at 18 s); pong carries `"op":"pong"`

### 12.1.3 Streams (private socket, all-in-one topics)

Subscribe once after auth: `{"op":"subscribe","args":["wallet","order","execution"]}`.

- **wallet**: `data[0].coin[]` with `coin`, `walletBalance`, `locked`, `equity`; account-wide USD fields (`totalEquity`, `totalAvailableBalance`). No snapshot on subscribe. Balance = walletBalance, available = walletBalance - locked.
- **order**: `data[]` with `orderId`, `orderLinkId` (= our client id), `orderStatus` (`New`, `PartiallyFilled`, `Filled`, `Cancelled`, `Rejected`, `Deactivated`, `Triggered`, `Untriggered`), `leavesQty`, `cumExecQty`, `updatedTime`. A cancel racing a fill can deliver two messages for one order — the hub's terminal-state checks make the second idempotent.
- **execution**: `data[]` with `execId`, `orderId`, `orderLinkId`, `execPrice`, `execQty`, `execFee`, `execTime`. Multiple executions may arrive in one message.

Positions are parsed as signals only — positions stay local truth derived from executions routed through the order-manager hub.

### 12.1.4 Order entry (trade socket)

```
{"header":{"X-BAPI-TIMESTAMP":"<ms>"},
 "reqId":"<our client id>",
 "op":"order.create"|"order.amend"|"order.cancel",
 "args":[{category:"linear", symbol, side:"Buy"|"Sell",
          orderType:"Limit"|"Market", qty, price?, timeInForce?,
          orderId? (amend/cancel), orderLinkId?}]}
```

- ack = accepted only; final state arrives via the private `order` topic.
- correlation needs no local map: `reqId` and `orderLinkId` both carry our decimal client order id; the store resolves them directly.
- amend requires the venue `orderId` -> unbound orders cannot be amended natively (the manager's cancel+replace covers that window).
- `retCode != 0` on an op response is surfaced as a reject through the corresponding order pipeline event.

## 12.2 exchangeProtocol_s organization

Use the shared protocol block ordering so adapters stay consistent and easy to maintain: `exchanges/exchgLighter/exchang` is the functional reference implementation; `exchanges/exchgBitfinex/exchangeProtocol.h` the layout skeleton. All adapters should keep the same block order and expose empty account/trading stubs when a complete shared-manager skeleton is required. Instrument discovery and historical parsing are intentionally separate from live marketdata parsing.

1. identity and capability flags
2. message logging and classification
3. session/control parsers (`processServerInfo`, `processSubscribed`, `processUnsubscribed`, `processLoginAck`)
4. marketdata parsers (`processInstrumentDetails`, `processTrade`, `processBbo`, `processUpdateBook`, `processBookSnapshot`)
5. account/trading parsers
6. request builders (instrument discovery, subscribe/unsubscribe, heartbeat/auth, historical)
7. historical response parser
8. private helper methods
9. static protocol state

Naming convention: protocol receive/ack handlers use `processXXXX` (not `handleXXXX`).

## 12.3 Testnet

Separate credentials (create them on testnet.bybit.com).

| Purpose | URL |
|---|---|
| Public marketdata | `wss://stream-testnet.bybit.com/v5/public/linear` |
| Private stream | `wss://stream-testnet.bybit.com/v5/private` |
| Order entry | `wss://stream-testnet.bybit.com/v5/trade` |
| REST | `https://api-testnet.bybit.com` |

Same wire formats as production; swap the four client URLs and the vault entries (e.g. `ttTrader/bybit-testnet/account/*`).

# Chapter 13

# dYdX v4 (dYdX Chain)

Marketdata: implemented (perps websocket: v4_trades / v4_orderbook / v4_candles) Account: implemented (v4_accounts credentials-authenticated ws channel) Trading: implemented (signed REST POST /v4/orders create, DELETE /v4/orders/{hash} cancel; no native modify) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading` true; no native stops/GTD/GAD/modify). Plain limit/market only; market orders ride the short-term IOC path. See docs/01-system-protocol/notification-pipeline.md.

## 13.1 Requirements capture

Sources (fetched September 2026): docs.dydx.exchange (indexer http/ws pages are JS-rendered — deep links not fetchable), the official v4 clients on GitHub (dydxprotocol/v4-clients: v4-client-js `socket-client.ts`, v4-client-py-deprecated `chain_helpers.py` + `dydx_composite_client.py`, `utility.py`), and the task wire spec. Re-verify the flagged fields at testnet sign-off.

### 13.1.1 Verified wire facts (source-checked)

| Fact | Source |
|---|---|
| WS envelope `{"type":"connected"\|"subscribed"\|"channel_data"\|"unsubscribed"\|"error","channel":...,"id":...,'` | v4-client-js socket-client.ts |
| Subscribe frame `{"type":"subscribe","channel":"v4_trades","id":"BTC-USD"}` (+ `v4_orderbook`, `v4_candles`, `batched` optional) | v4-client-js socket-client.ts |
| Server pushes the plain text `PING`, client answers text `PONG` (no JSON frame) | v4-client-js socket-client.ts (current version) |
| Candle resolutions `1MIN 5MINS 15MINS 30MINS 1HOUR 4HOURS 1DAY` (plural MINS/HOURS — note the task wire spec said 5MIN/15MIN/30MIN/4HOUR singular) | v4-client-js socket-client.ts |
| `GET /v4/time`, `GET /v4/height` (indexer block height) exist | v4-client-py-deprecated utility.py |

| Fact | Source |
|---|---|
| Market math: `quantums = floor(size * 10^-atomicResolution / stepBaseQuantums) * stepBaseQuantums`, min stepBaseQuantums; `subticks = floor(price * 10^(atomicResolution - quantumConversionExponent + 6) / subticksPerTick) * subticksPerTick`, min subticksPerTick | v4-client-py-deprecated chain_helpers.py |
| Order flags `SHORT_TERM = 0`, `LONG_TERM = 64` (not 0x40); market = SHORT_TERM + IOC; limit GTT = LONG_TERM; `SHORT_BLOCK_WINDOW = 20` | v4-client-py-deprecated chain_helpers.py |
| `EIP712_SIGNATURE` subsystem wrapper: `CEip712Signing::sign(domain, msg, key)` digests `0x1901 \|\| msg \|\| domain` (secp256k1, low-s) | ttSubSystem src/eip712Signing.cpp |
| Indexer market fields: `clobPairId`, `atomicResolution`, `quantumConversionExponent`, `subticksPerTick`, `stepBaseQuantums`, `tickSize`, `stepSize`, `minOrderSize`, `oraclePrice`, `status` | v4-client-py place_order_message consumption of the indexer response |

### 13.1.2 Assumed / task-spec wire facts (NOT source-verified — uncertainties)

The official v4 clients place orders via the validator (cosmos tx), not via `POST /v4/orders`. The REST order path below follows the task wire spec (the hosted dYdX v4 trading api gateway). The docs site pages for it were JS-rendered and could not be fetched during implementation.

1. **EIP-712 domain**: `EIP712Domain(string name,string version,uint256 chainId)` with name `"dYdX"`, version `"1"`, chainId `6627239` (mainnet default, config-pinnable via the `"chainId"` client key, lighter-style). The exact dydx-chain EIP-712 domain chain id value is unverified.
2. **ShortTermOrder type string**: canonical on-chain form `ShortTermOrder(bytes32 orderId,int32 subaccountId,uint32 clientId,uint32 clobPairId,string side,uint64 quantums,uint64 rawPrice,uint8 orderFlags,uint32 goodTilBlock,string execution)`; `LongTermOrder(... uint32 goodTilBlockTime ...)` for GTT. UNVERIFIED which of `goodTilBlock` (block height) vs `goodTilBlockTime` (unix seconds) the REST gateway binds — the adapter passes the seconds deadline (`now + goodTilTimeInSeconds`) into the `goodTilBlock` slot for short-term orders. `GET /v4/height` is the fallback hardening hook if the gateway requires block heights.
3. **POST /v4/orders body**: `{"order":{"orderId","subaccountId","clientId","clobPairId","side","quantums` — field names/types unverified; `quantums`/`rawPrice` ride as JSON strings (uint64 can exceed the 2^53 JSON-number safe range).
4. **Signature hex**: `"0x" + r(32B) + s(32B) + v(1B)` (65-byte ethereum signature form), `signatureType = 2` (DYDX_SIGNED EIP-712).
5. **Request signing** (REST headers + accounts subscribe): `signature = base64(HMAC-SHA256(base64decode(secret), timestamp || method || requestPath || body))`, ISO8601 timestamp, headers `DYDX-API-KEY / DYDX-TIMESTAMP / DYDX-SIGNATURE / DYDX-PASSPHRASE`. Task-spec (matches the dYdX indexer signing convention); the exact string-to-sign canonicalization is unverified.
6. **v4_accounts subscribe frame**: walletAddress + includeOnlyAddresses + `apiKeyCredentials {key, secret(b64), passphrase(b64)}` + signature + timestamp. The ws requestPath/method used for the signature (`WEBSOCKET_SUBSCRIBE` + `/v4/ws/accounts`) is an adapter choice, flagged. Note: the current official v4-client-js socket client exposes only `v4_subaccounts` keyed by `address/subaccountNumber` WITHOUT credentials; the credentials-based v4_accounts channel follows the task wire spec / the v3 accounts-channel pattern and must be verified at testnet.

7. **Cancel**: `DELETE /v4/orders/{orderHash}` (path-parameter form). If the gateway expects `DELETE /v4/orders` with a body of `{"orderId":...}` the adapter target string changes in one place (`buildOrderCancel`).
8. **Order id layout** (constructed pre-signature): 32 bytes = `orderFlags(1B LE) || clobPairId(2B LE) || clientId(4B LE) || wallet address(20B) || subaccount number(5B BE)`. UNVERIFIED (derived from the v4-chain OrderId serialization shape); correlation never depends on it — the venue-returned hash is authoritative and re-binds into the side map.
9. **Orderbook update semantics**: `channel_data` carries only the changed price levels with their new aggregate sizes; a size of `""` (or `"0"`) removes the level. Removal-by-empty-string is the common indexer convention but is not source-verified — a wrong reading would corrupt book state only until the next reconnect snapshot.
10. **v4_candles push shape — RESOLVED 2026-09-07 (first live G1 run)**: the live push carries `contents.candles` as an **array** of candle objects (each `{"startedAt":...,"ticker":"BTC-USD","resolution":"1MIN", "open":...,"high":...,"low":...,"close":...,"baseTokenVolume":..., "usdVolume":...,"trades":...}`), NOT the documented `market/resolution` keyed object; no explicit finality flag — a candle whose close time has passed is treated as final. The parser now accepts both forms.
11. **Status values** `OPEN | FILLED | CANCELED | UNTRIGGERED | BEST_EFFORT_CANCELED` with `FILLED -> canceled` retirement (the fills in the same push account the size — house Paradex CLOSED+FULLY_FILLED precedent).
12. **REST error bodies**: treated as errors when the body carries `errors`, `error` or `message` members; the human text rides `message/error`.

### 13.1.3 Crypto stack (`dydxAuth.h`)

- HMAC-SHA256 request signing over OpenSSL (`HMAC()`; OpenSSL::SSL is linked — note the LINUX block of the CMakeLists adds it, unlike Paradex).
- base64 decode/encode helpers (RFC 4648 alphabet; secrets are base64).
- EIP-712 typed-data primitives (standard encoding, keccak via `CHashing::keccak`): domain separator `EIP712Domain(string name,string version,uint256 chainId)`, struct hash = `keccak(typeHash || enc(field)…)`.
- ECDSA via `CEip712Signing::sign(structHash, domainSeparator, key)` — the subsystem digests `0x1901 || p_h32Msg || p_h32Domain`, so the argument order above yields the standard `0x1901 || domainSeparator || structHash` digest (same trick as hyperliquidAuth.h). low-s normalized, v in 27/28.
- Private key parsing + address derivation follow hyperliquidAuth.h exactly (hash32_t stores the key bytes reversed; `memoryToHex(true)` re-reverses to canonical hex).

### 13.1.4 Endpoints & flow

| Purpose | URL |
|---|---|
| Public ws | `wss://indexer.dydx.trade/v4/ws` |
| Marketdata REST | `https://indexer.dydx.trade/v4` (`/perpetualMarkets`, `/candles`) |
| Private ws | same ws endpoint; signed `v4_accounts` subscribe frame |
| Orders | `POST /v4/orders`, cancel `DELETE /v4/orders/{hash}` (signed, DYDX-* headers) |

cfg convention: the **`http clientUrl` stays host-only** (`https://indexer.dydx.trade`) and the request builders carry the full `/v4/...` path — a `/v4` suffix in the http clientUrl doubles the prefix and the venue

answers `404 {"error":"Not Found"}` (first G1 run, fixed 2026-09-06). The ws `clientUrl` is used verbatim and keeps `/v4/ws`.

Startup sequence: marketdata ws + market http connect in parallel -> `/v4/perpetualMarkets` discovery (status ACTIVE only) -> instrument subscribes. Account ws connect -> `buildCustomConnectAccount` yields the signed `v4_accounts` subscribe once per connection (the account http client connect of the same manager is a no-op) -> orders/fills/positions pushes are classified as EMSG_UPDATE_ORDER and fanned out inside `trading.processUpdateOrders`. Credentials are cached at `serverConnected` from the account/trading client entries.

### 13.1.5 Keepalive

The server pushes the plain text `PING` (the connection base consumes text ping/pong frames before the JSON parser); the adapter answers with client-driven text `PONG` frames every ~20 s via `shouldSendPing`/`buildPing` (the bitget v2 pattern) plus the transport-level `framePing` (30 s).

### 13.1.6 Credentials

- `apiKeyRef`: wallet private key (`0x` + 64 hex) signing EIP-712 orders; wallet address derives at connect. Optionally `"0x<pk>,0x<apikey>"` — a second comma-separated value pins the DYDX-API-KEY credential explicitly; without it the adapter sends the derived wallet address as the api key (uncertainty 5/6 above).
- `apiSecretRef`: base64 api secret (HMAC request signing + `apiKeyCredentials.secret`).
- `apiPassphraseRef`: base64 api passphrase (DYDX-PASSPHRASE header + `apiKeyCredentials.passphrase`).
- `i32ClientId`: subaccount number (default 0).
- `"chainId"` (client entry, optional): EIP-712 domain chain id pin.

### 13.1.7 Correlation

The venue order id is a `0x...` 66-char hash — it cannot ride the uint64 exchange id. Correlation rides the (uint32-truncated) `clientId` echoed in every orders/fills push and resolves through a fixed 256-entry ring side map in the root (clientId + hash -> orderInfo_s*, single-writer strand, no locks, no heap). Cancels ride the cached hash (`DELETE /v4/orders/{hash}`). Foreign orders (unknown client id) are logged and dropped.

### 13.1.8 Candle support

- Streaming: `v4_candles`; the live push carries `contents.candles` as an **array** of candle objects (`ticker` + `resolution` on each entry; verified 2026-09-07 — the keyed `market/resolution` object form is also accepted), resolution mapped from the instrument's exchange-stream series (`1MIN`…`1DAY`); provisional / final split by close time (no venue finality flag).
- Historical: `GET /v4/candles/perpetualMarkets/{market}?resolution=…&limit=…` (verified live 2026-09-07; a bare `/v4/candles` answers 404 Not Found; limit clamped to 1000; house request is `_OHLC_ITEM_REQUEST + 2`). Ascending bars; the still-open candle is excluded from the payload (house convention, OKX reference). Requested intervals snap down to the nearest supported resolution.

### 13.1.9 Book + bbo (observed at first live G1 run, 2026-09-07)

- The v4 ws has **no bbo/ticker channel** — the bbo derives from the `v4_orderbook` top (`ce_fBookUpdatesBbo = true`). The book parse stamps receive-time timestamps and a monotonic sequence (the wire carries none) so the derived quotes carry valid metadata.
- Three live level shapes are handled by `_normalizeSideMap`: pair arrays (`"bids":[["79917","0.0027"]]`) on channel_data updates, object rows (`{"price":"80043","size":"0.0002"}`) on the subscribed snapshot, and the documented map form for compatibility.
- The live `/v4/perpetualMarkets` response carries **no** `minOrderSize` field — the size `stepSize` doubles as the minimum increment.

### 13.1.10 Fees / leverage display

The perpetualMarkets payload carries no maker/taker fee or leverage fields — defaults (fees 0, leverage 1–20) are display-only placeholders.

## 13.2 Testnet

| Purpose | URL |
|---|---|
| WS | `wss://indexer.v4testnet.dydx.exchange/v4/ws` |
| REST | `https://indexer.v4testnet.dydx.exchange/v4` |

Swap both `clientUrls` in the cfgs; if the EIP-712 domain chain id differs from mainnet, pin it with `"chainId"` on the trading client. Testnet sign-off should resolve every item in the uncertainties list above — especially the ShortTermOrder deadline binding (2), the POST /v4/orders body shape (3) and the v4_accounts subscribe signature path (6).

# Chapter 14

# Hyperliquid

Marketdata: implemented (trades + l2Book ws streams) Account: implemented (orderUpdates / userFills user streams — keyed by the derived wallet address, no auth token) Trading: implemented (signed POST /exchange L1 actions: order/cancel/modify) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading`, native modify true). Limit-only this pass: market orders and native trigger orders land with testnet hardening. See docs/01-system-protocol/notification-pipeline.md.

## 14.1 Requirements capture

Source: hyperliquid-dex/hyperliquid-python-sdk `utils/signing.py` (read verbatim) + hyperliquid.gitbook.io signing docs. Captured August 2026.

### 14.1.1 Signing (hyperliquidAuth.h)

1. `action_hash = keccak256( msgpack(action) || nonce(8B BE) || 0x00 )` (0x00 = no vault address). **msgpack field ORDER matters** — the server re-packs deterministically per the SDK layout (`type`,`orders`,`grouping`; wire order a,b,p,s,r,t).
2. EIP-712 over domain `{name:"Exchange", version:"1", chainId:1337, verifyingContract:0x0}` with primary type `Agent{source:string, connectionId:bytes32}`, source `"a"` (mainnet).
3. z = standard EIP-712 digest (no personal-message wrap); secp256k1 deterministic-k ECDSA via `CEip712Signing`.
4. Implementation note: `CEip712Signing::sign(domain,msg,key)` digests `0x1901||msg||domain` — pass `(structHash, domainSeparator)` so the digest matches standard argument order (see hyperliquidAuth.h).

### 14.1.2 Endpoints

| Purpose | URL |
|---|---|
| WS (public + user streams) | `wss://api.hyperliquid.xyz/ws` |
| Exchange actions | `POST https://api.hyperliquid.xyz/exchange` |
| Info queries | `POST https://api.hyperliquid.xyz/info` |

### 14.1.3 Streams

- public: `trades`, `l2Book` per coin

- user (subscribe with `user` = derived wallet address):
  - `orderUpdates`: `data[].order` with `oid`, `status` (open, `filled`, `canceled`)
  - `userFills`: fills with px, `sz`, `fee`, `hash`, `oid`
- balances are not pushed on this venue; they seed from `/info` clearinghouseState at reconcile points (positions stay local truth)

### 14.1.4 Order entry

`POST /exchange` envelope `{"action", "nonce", "signature":{r,s,v}}`. Order wire: `{"a":<asset idx>,"b":bool,"p":"px","s":"sz","r":false, "t":{"limit":{"tif":"Gtc"}}}`. Asset indices come from `/info meta` universe position, captured at discovery. Cancel by venue `oid`; modify via the `modify` action. Ack binds the first status entry's `oid`.

Credentials: `apiSecretRef` holds the hex EVM private key; the wallet address derives at connect (`CEip712Signing::infoPriv` and drives the user-stream subscriptions.

## 14.2 Testnet

| Purpose | URL |
|---|---|
| WS (public + user streams) | `wss://api.hyperliquid-testnet.xyz/ws` |
| Exchange/info | `https://api.hyperliquid-testnet.xyz` |

**Signing differs**: the phantom-agent `source` field is `"b"` on testnet vs `"a"` on mainnet (see hyperliquidAuth.h). Flip the constant when configuring the testnet run; asset indices from `/info meta` also differ.

# Chapter 15

# Interactive Brokers (IBKR)

Interactive Brokers via the **TWS API wire protocol** — raw TCP to a running TWS (or IB Gateway) instance. Each sub-manager owns its own API connection with a distinct client id; TWS identifies API sessions by that id. The old Client Portal Gateway (`localhost:5000`) transport is retired.

```
exchanges/exchgInteractiveBrokers/
  exchangeProtocol.h             # root: wire codec, frame dispatch, session
                                 # state machines, shared wire helpers
  exchangeProtocol.marketdata.h  # marketdata channel: subscribe builders,
                                 # tick/depth decoders, bbo overlay + book replay
  exchangeProtocol.trading.h     # trading channel: order builders, execution
                                 # decoders, execution dedupe
  exchangeProtocol.account.h     # account channel: account/position decoders,
                                 # wallet fan-out
  exchange.cfg                 # the venue's exchanges[] entry
algoExchangeTest.cfg         # single certification config: NES / MBT / BTC
                             # instruments, simulation + paper serverSettings;
                             # the algo's instTradeId picks the instrument,
                             # tradingMode / --tradingMode picks the mode
  cert-g1-sim.md               # G1 simulation certification report
  cert-g2-paper.md             # G2 paper certification report
```

Ownership rule (`.github/instructions/20-exchange-protocol-layout.instructions.md`): the root keeps identity, dispatch and shared plumbing; everything marketdata / trading / account lives in its channel file.

## 15.1 Prerequisites

1. **TWS** (or IB Gateway) running and logged in — paper account for certification (ids prefixed `D`, e.g. `DU0512766`).
2. API access enabled: *Configure* -> *Settings* -> *API* -> *Settings* -> "Enable ActiveX and Socket Clients", socket port **7497** (paper), "Read-Only API" **off**.
3. A market data entitlement for the traded instruments (CME futures L1 for the cert profile — trades + bbo; L2 depth needs the BookTrader entitlement).

No API keys or OAuth credentials — the TWS login owns the session. If TWS restarts while the platform runs, the adapters reconnect and re-handshake automatically.

## 15.2 Connection model

One `tcp://` client per sub-manager in `exchange.cfg`:

| client | type | default clientId | session |
|---|---|---|---|
| marketdata | `market` | 4242 | handshake -> START_API -> READY -> `REQ_MKT_DATA` ticks |
| trading | `trading` | 4243 | handshake -> START_API -> READY -> order wire |
| account | `account` | 4244 | handshake -> START_API -> READY -> `REQ_ACCT_DATA` / `REQ_ACCOUNT_SUMMARY` / `REQ_POSITIONS` |

A `clientId` left poisoned in TWS state by an uncleanly killed session is silently starved by TWS until its restart — rotate via config instead of rebuilding.

## 15.3 One config file for sim / paper / live

The `clients` array can be replaced by a `serverSettings` object that maps a `tradingMode` name to its own client set. `tradingMode` in the file — or `--tradingMode` on the command line — selects the set; only the selected set is resolved at load, so unselected secrets never leave the credential store:

```
"tradingMode": "paper",
"serverSettings": {
  "simulation": [ { "type": "market", "clientUrl": "tcp://127.0.0.1:7497", "clientId": 4242 }, ...
  "paper":      [ { "type": "market", "clientUrl": "tcp://127.0.0.1:7497", "clientId": 4242 }, ...
  "live":       [ { "type": "market", "clientUrl": "tcp://127.0.0.1:7496", "clientId": 4242 }, ...
}
```

- `"paper"` is a real `tradingMode` value now: it maps to its own execution mode `EEM_PAPER`, runs the production engine (real order flow — identical to live) and selects the paper set. visible as such: the startup banner says `PAPER TRADING` (never `LIVE TRADING`) and the websocket states snapshot reports `ex: 4` / `mode: "paper"`. IBKR needs no keys; its sets differ by TWS port (paper 7497 / live 7496) and client ids.
- `playback` falls back to the `simulation` set, then to a flat `clients` array.
- A flat `clients` array stays fully valid (legacy configs load unchanged). No `serverSettings` set for the selected mode and no flat array -> config load error. Unknown keys inside `serverSettings` -> warning; a non-array set or invalid client entry -> load error.

## 15.4 Protocol notes

- **Handshake**: client speaks first with `API\0` + big-endian length + `v100..170`; TWS answers one length-framed `[serverVersion][time]`, then the ping tick dispatches START_API once per connection epoch. READY is per-session, marked by `NEXT_VALID_ID` on the receiving socket.
- **Wire version pinned to 170** so the layouts cover the full modern message set (Paxos crypto needs >=170; the official ibapi 9.81 client is the layout reference, verified against the live paper TWS).
- **Orders**: PLACE_ORDER (v45 layout, 99 fields), CANCEL_ORDER; modify is a complete-order resubmission with the same TWS order id. Native plain stops (STP / STP LMT with aux trigger). Trailing stops ride the manager's hybrid ratchet (one-way trigger moves via native modify). GTD/GAD boundaries are manager-enforced (`fNativeGoodTilDate`/`fNativeGoodAfterDate` false).

- **Executions**: EXECUTION_DATA is deduplicated by execId; COMMISSION_REPORT attaches the fee before the fill event is published. Fill accounting is single-owner (the order manager's fill path, driven by EXECUTION_DATA): TWS sends ORDER_STATUS `'Filled'` **before** EXECUTION_DATA, so the status handler must not book the filled size — it would pre-book the fill and the subsequent execution clamps to a zero remaining size (no fill event, no position). Overshoots clamp to the remaining size in both the active and the late-fill path; a fully-clamped fill is dropped.
- **Market data**: the authoritative bbo source is the tick-by-tick **BidAsk** feed (REQ_TICK_BY_TICK_DATA, tick type 3) — one atomic bid/ask/size pair per message; the REQ_MKT_DATA tick overlay (bid 1/0, ask 2/3, last 4/5) is suppressed while BidAsk is live and falls back on rollback. REQ_MKT_DATA carries the mandatory

  delta-neutral flag (omitting it makes the TWS field decoder hit end-of-frame and **silently drop the request**). L2 depth rides REQ_MKT_DEPTH(10) -> UPDATE_MKT_DEPTH(12) / UPDATE_MKT_DEPTH_L2(13) position-indexed rows (op 0/1/2 = insert/update/ delete, side 0/1 = ask/bid), entitlement-gated (BookTrader). TWS depth rows are liquidity buckets, not price levels — CME rows can arrive out of price order and at duplicate prices — so the adapter publishes a normalized price-level ladder: sorted insertion with equal-price rows aggregated into one level, and rows strictly beyond the freshest venue-confirmed price of the opposite side (crossed tops, delete frames in flight) trimmed from the published view only. The wire-aligned row storage is untouched. A locked top (bid == ask) is legal CME state and publishes.
- **Errors**: `ERR_MSG` codes 2104/2106/2107/2158 are farm status (info); 202 restates the cancel; 2109 is an order-event warning (order keeps processing); every other order-scoped code retires the order as failed.
- TWS rejects resting futures orders beyond ~3% from market (error 163, precautionary constraints) — keep `bboOffsetBps` <= ~150.
- **Crypto (Paxos) needs the api level pinned to >=170.** At the old pin 103 TWS refused crypto market data with error 10285; at 163 market data flows but TWS *silently drops* crypto orders; the order gate opens with the size-rule era (verified live at 170). At 164+ the CONTRACT_DATA response drops its per-message version and the mdSizeMultiplier and gains the size rules after the secId list; ORDER_STATUS (api 131+), OPEN_ORDER (145+), EXECUTION_DATA (136+) and PLACE_ORDER (145+) lose their per-message version fields.
- **Paxos time-in-force:** GTC is not supported for crypto — resting orders ride as `DAY` (mapped by the adapter). Outside the Paxos settlement window (orders queue to 03:00 US/Eastern) the venue accepts only short-lived ("Minutes") or IOC orders, and a paper *margin* account rejects crypto entirely with "MARGIN CALCULATION IS NOT SUPPORTED FOR THIS CONTRACT" — a venue/account constraint, not an adapter defect.

## 15.5 Instruments

Instruments are pinned by **conid** in `idAtExchg` (bare numeric id — the adapter resolves it directly into runtime info):

```
"instruments": [
  {
    "id": "instInteractiveBrokersNES",
    "exchgId": "exchgInteractiveBrokers",
    "idAtExchg": "908100745"
  }
]
```

Contract fields for orders (sec type, exchange, currency) come from the adapter's session profile (`FUT` / `CME` / `USD` for the NES cert profile).

## 15.6 Certification status

| Gate | Result |
|---|---|
| G1 simulation | [x] CERTIFIED 2026-09-01 — 26/26, 0 skipped (NES conid `908100745`, including live L2 depth) (`cert-g1-sim.md`) |
| G2 paper | [x] **CERTIFIED 2026-09-01 — 26/26, 0 skipped** (NES conid `908100745`, real fills including Position Close, clean exit, and live L2 depth) (`cert-g2-paper.md`) |

Book (L2 depth): REQ_MKT_DEPTH / UPDATE_MKT_DEPTH wire is implemented and was verified live on the paper account during the 2026-09-01 G2 run. Journal wiring (§1.6) and the 2 h soak (§1.7) remain shared operational gates before [x][x].

# Chapter 16

# Kraken (Futures / derivatives v3)

2-file design: exchangeProtocol.h + CMakeLists.txt

Marketdata: implemented (futures public ws feeds trade/book) Account: implemented (challenge-auth ws openOrders stream) Trading: implemented (signed HTTP sendorder/amendorder/cancelorder; native amend) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading`, native modify true). Plain limit/market only: stops and GTD/GAD are emulated by the order manager. See docs/01-system-protocol/notification-pipeline.md.

## 16.1 Requirements capture (kraken futures derivatives api v3)

Source: docs.kraken.com/api/futures-api (REST trading v3 + websocket v1). Captured August 2026; re-verify wire field names against live docs before testnet sign-off.

### 16.1.1 Endpoints

| Purpose | URL |
|---|---|
| Public ws (trade/book feeds) | `wss://futures.kraken.com/derivatives` |
| Private ws (openOrders feed) | challenge-auth subscribe on the same host |
| REST trading | `https://futures.kraken.com/derivatives/api/v3/sendorde` |
| Instrument discovery | `/derivatives/api/v3/instruments` |

### 16.1.2 Authentication

- REST headers: `APIKey`, `Nonce` (ms), `Authenticator` = hex( HMAC-SHA512( base64decode(apiSecret), nonceMs + postBody ) )
- WS: GET `/derivatives/api/v3/challenge` -> challenge string; `sign = hex(HMAC-SHA512(secret, challenge))`; subscribe frame `{"event":"subscribe","feed":"openOrders","api_key":...,"sign":...}`
- connect sequence: account http client fetches the challenge (`buildCustomConnectAccount`), the response routes through EMSG_AUTHENTICATION -> processLoginAck computes the signature, then the template sends the ws-auth subscription

### 16.1.3 Streams

- `openOrders` feed delivers `orderEvents[]` with types PLACEMENT / EXECUTION / CANCELLATION; each carries the `order` object (orderId, cliOrdId = our client id, status) and execution qty/price/fee for fills.

- correlation needs no local map: cliOrdId carries our decimal client id and the store resolves it directly; foreign orders are logged and dropped.

### 16.1.4 Order entry

POST bodies carry nonce, `cliOrdId`, `symbol`, `side`, `orderType` (`lmt`/`mkt`), `limitPrice`, `size`, `timeInForce` (`gtc`/`ioc`). Amend uses the venue `orderId`. Ack binds `sendStatus.order_id`; authoritative state arrives on the openOrders feed.

## 16.2 exchangeProtocol_s organization

Use the shared protocol block ordering so adapters stay consistent and easy to maintain: `exchanges/exchgLighter/exchang` is the functional reference implementation; `exchanges/exchgBitfinex/exchangeProtocol.h` the layout skeleton.

Naming convention: protocol receive/ack handlers use `processXXXX` (not `handleXXXX`).

## 16.3 Testnet

Kraken Futures provides a **demo** environment with identical APIs:

| Purpose | URL |
|---|---|
| Public ws feeds | `wss://demo.futures.kraken.com/derivatives` |
| Private ws (challenge auth) | same host |
| REST trading/discovery | `https://demo.futures.kraken.com/derivatives/api/v3/...` |

Demo accounts are provisioned through the Kraken Futures UI; api key/secret are demo-scoped.

# Chapter 17

# Lighter (lighter.xyz)

Low-latency L2 exchange interface. Marketdata via WebSocket + REST HTTP for instrument discovery. Authenticated account state via `account_all` WS channel. Trading via signed HTTP POST to `/api/v1/sendTx`.

All signing is performed by lighterCrypto (Poseidon2 + Goldilocks field EC, no third-party crypto libs).

## 17.1 exchangeConfigInfo_s fields

| field | usage |
|---|---|
| `i32KeyIndex` | api key index (int32_t, used directly). |
| `strApiSecret` | 40-byte private key as 80 hex chars, optional `"0x"` prefix. decoded into 5 goldilocks limbs. |
| `i32ClientId` | L1 account index. passed to auth state as `accountIndex`. |
| `u32ChainId` | optional L2 chain id for tx hashing (mainnet 304, testnet 300). the testnet discovery carries no chain id, so a testnet run MUST pin it on the trading client (`"chainId": 300`) or every signed tx fails venue verification with 21120. |
| `hdClient` | host details for the websocket and REST connections. |
| `eType` | `ECT_MARKETDATA`, `ECT_TRADING`, or `ECT_ACCOUNT` — determines which connection this config entry drives. |
| `i32FramePing` | websocket ping interval in seconds. |
| `strLogPattern` | log prefix for this connection. |

the account index (`i32ClientId`) and api key index (`i32KeyIndex`) together identify the on-chain keypair. the private key (`strApiSecret`) is cleared via `SecureZeroMemory` on destructor.

## 17.2 files

| file | role |
|---|---|
| `exchangeProtocol.h` | wire-protocol adapter: message classification, marketdata/account/trading parsers, request builders. |
| `lighterAuth.h/cpp` | api key management, auth-token generation, nonce manager, schnorr sign wrapper. |

| file | role |
|---|---|
| `lighterCrypto.h/cpp` | goldilocks field arithmetic, EC point operations, poseidon2 hash, schnorr signature. |
| `lighterTx.h/cpp` | transaction canonical encoding and hashing (1:1 compatible with lighter-go). |
| `tests/` | native crypto + tx + auth unit tests (no third-party deps). |

## 17.3 capabilities

| feature | status |
|---|---|
| marketdata (websocket order book, trades, bbo, candles) | implemented |
| instrument discovery (REST /api/v1/info) | implemented |
| account state (assets, orders, positions, executions) | implemented |
| order create / cancel / modify (signed HTTP POST) | implemented |
| auth token (schnorr-signed, 7h expiry) | implemented |

declared in `exchangeProtocol_s::ce_capabilities`: `fMarketdata`, `fAccount`, `fTrading` true; native stops + native modify + native good-til-date true (trailing stops run as the hybrid manager ratchet over native modify). wallet updates route through the order-manager hub — see `docs/01-system-protocol/notification-pipeline.md`.

## 17.4 protocol blocks

the adapter follows the shared `exchangeProtocol_s` block order (`exchanges/exchgBitfinex/exchangeProtocol.h` is the layout skeleton; this adapter is the functional reference):

1. identity and capability flags
2. message logging and classification
3. session/control parsers
4. marketdata parsers
5. account/trading parsers
6. request builders
7. private helper methods
8. static protocol state

## 17.5 Testnet

zkLighter operates a public testnet with the same API surface:

| Purpose | URL |
|---|---|
| REST + WS | `https://testnet.zklighter.elliot.ai` (`/stream` suffix for ws) |
| trading UI + faucet | `https://testnet.app.lighter.xyz` |

Chain id, account index and api key indexes differ from mainnet - mint testnet keys and use the testnet account/index values in the account/trading clients. the L2 chain id MUST be pinned on the trading clients

("chainId": 300; mainnet runs pin 304): it feeds the tx hash, the testnet discovery response carries no chain-id field, and without the pin every signed tx is rejected with 21120 invalid signature (the error labels the signature, the cause is the wrong chain id in the hash).

## 17.6 certification

algoExchangeTest.cfg in this folder drives the 26-test certification suite against the BTC/USDC perpetual. the config carries three serverSettings sets:

| set | order flow | connections |
|---|---|---|
| simulation (default) | emulated in-process | market data (mainnet) |
| paper | real, on the testnet | market data + trading + account (testnet) |

launch:

- simulation: ttTrader.exe <config> --tradingMode simulation
- paper/testnet: ttTrader.exe <config> --tradingMode paper

### 17.6.1 cert state (2026-09-03)

the config runs all 26 tests in both modes — the skipTests pin is removed. testnet BTC public trade flow is sparse to zero for long stretches, so the config carries "mdMaxWaitSeconds": 3600: a marketdata test with zero samples at the 30 s checkpoint waits up to 1 h for the first tick instead of failing at the 600 s default (a feed with at least one sample always passes; book/bbo flow continuously and pass early).

- simulation: **CERTIFIED 26/26** (mainnet feed carries public trades; report cert-g1-sim.md)
- paper/testnet: **CERTIFIED 26/26, 0 skipped** — test 01 waited ~3 min for the first testnet tick under the extended window, then the full suite incl. real order flow passed end to end (cert-g2-paper.md)

### 17.6.2 mode visibility

paper runs on its own execution mode (EEM_PAPER) — never confusable with a live run:

- the startup banner says ********* PAPER TRADING *********
- the websocket states snapshot reports ex: 4 (EEM_PAPER) and mode: "paper" alongside ex: 1 (EEM_PRODUCTION) for live; simulation stays ex: 2 / mode: "simulation" — the dashboard maps mode (or ex) straight to its run badge

### 17.6.3 keys and permissions

the paper set uses two separate api keys for one account (same clientId account index, different keyIndex), and the key secrets live only in the windows credential store - never in the config:

| connection | key permission | vault target |
|---|---|---|
| trading | transaction permission (signs orders) | cred:ttTrader/lighterTestnet/trading/ap |
| account | read permission (account_all streams) | cred:ttTrader/lighterTestnet/account/ap |

seed the vault targets once via ttSubSystem::writeCredential (or a small helper) - the 40-byte hex private key per api key, 80 chars, optional 0x.

### 17.6.4 marketdata wait behavior

the three marketdata tests (trades/bbo/book) use a two-stage wait, tuned for thin testnet liquidity: they pass as soon as 10 valid samples arrived, or - if the feed delivered at least one valid sample - at the 30 s checkpoint (`mdMinWaitSeconds`); a feed with zero samples extends once to 600 s (`mdMaxWaitSeconds`) before failing. the no-BBO suite guard uses the same 600 s window. test 01 therefore takes a few minutes on quiet testnets - that is the window doing its job, not a hang.

### 17.6.5 order locations

resting cert orders report their location: `EOL_EXCHANGE` for native-resting limits, `EOL_BROKER` for orders the venue/broker holds and triggers (stops on venues without native stop support), `EOL_MIXED` after partial fills, and `EOL_LOCAL` for locally simulated stops in the simulation set. the position close test opens its own position when the market tests net the account flat, so it verifies the close path in every mode.

# Chapter 18

# OKX (v5)

Marketdata: implemented (public trades/bbo-tbt/books) Account: implemented (login -> orders/account/fills channels on the private ws) Trading: implemented (ws ops order/cancel-order/amend-order; native amend) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading`, native modify true). Plain limit/market only: stops and GTD/GAD are emulated by the order manager. See docs/01-system-protocol/notification-pipeline.md.

## 18.1 Requirements capture (okx v5)

Source: www.okx.com/docs-v5 — websocket login, trade, private channels (orders / account / fills). Captured August 2026; re-verify before testnet.

### 18.1.1 Endpoints

| Purpose | URL |
|---|---|
| Public ws | `wss://ws.okx.com:8443/ws/v5/public` |
| Private ws (login + streams + order entry) | `wss://ws.okx.com:8443/ws/v5/private` |
| REST | `https://www.okx.com` |

### 18.1.2 Authentication

- timestamp = unix seconds; `sign = base64( HMAC-SHA256( secret, timestamp + "GET/users/self/verify" ) )`
- frame: `{"op":"login","args":[apiKey, passphrase, timestamp, sign]}`; ack `{"event":"login","code":"0"`
- passphrase is the third credential — config key `apiPassphraseRef` (`cred:` store), stored zeroized in `exchangeConfigInfo_s`

### 18.1.3 Streams (private socket)

- `orders`: `data[]` with `ordId`, `clOrdId` (= our client id), `state` (`live`, `partially_filled`, `filled`, `canceled`), fills included
- `account`: `data[].details[]` per currency with `ccy`, `cashBal`, `availEq`, `equity`, `frozen` -> routed through the hub
- `fills`: separate fill events with `fillPx`, `fillSz`, signed `fee`, `ts`

### 18.1.4 Order entry (private socket)

`{"id":<ms>,"op":"order"|"cancel-order"|"amend-order","args":[{instId, tdMode:"cross", side, ordType:"limit"/"market", sz, px?, clOrdId?/ordId?, newSz?/newPx?}]}` — acks echo `clOrdId`; `code != "0"` surfaces a reject; authoritative state arrives via the orders channel.

### 18.1.5 Historical data

`buildHistoricalDataRequest` uses `/api/v5/market/history-candles`, which caps `limit` at 100. The shared request default (`_OHLC_ITEM_REQUEST + 2` = 512) exceeds that, so the builder clamps to 100 — historical warm-up on OKX therefore seeds at most 100 candles.

## 18.2 Testnet

OKX runs **simulated trading** on the production hosts rather than separate URLs: send the `x-simulated-trading: 1` header on authenticated REST calls and fund a demo account. WS demo support (query-parameter variant) must be confirmed against current docs during G2 prep — if the private socket does not accept the flag, validate order entry over REST demo + streams on production read-only channels.

Demo credentials/passphrase are separate from live.

# Chapter 19

# Paradex

Marketdata: implemented (perps WebSocket streaming) Account: implemented (JWT-authenticated ws orders/fills channels) Trading: implemented (signed HTTP order create/cancel/modify; native modify) Cert path: simulation cert with live marketdata -> testnet -> production

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`fMarketdata`, `fAccount`, `fTrading` true; no native stops/GTD/GAD). Plain limit/market only. See docs/01-system-protocol/notification-pipeline.md.

## 19.1 Requirements capture

Sources (fetched August 2026): docs.paradex.trade (api-authentication, ws authentication, order endpoints), tradeparadex/paradex-py message builders, starkware-libs/cairo-lang crypto reference. Re-verify wire fields at testnet sign-off.

### 19.1.1 Crypto stack (`paradexAuth.h` + generated `paradexPedersenPoints.h`)

- STARK curve ECDSA (offline-tested by the `ParadexAuth` suite): constants from pedersen_params.json, reference quirks mirrored (r = R.x unreduced, s = (msgHash + r·d)/k mod n), RFC6979 deterministic k, sn_keccak
- SNIP-12 revision 0 typed-data hashing (keccak-based): type hashes via sn_keccak of the encoded type string; struct folding via Pedersen compute_hash_on_elements over the 506-point constant table
- signed_data assembly: [ascii("StarkNet Message"), domain hash, account, message struct hash] with domain {Paradex, chainId hex, "1"}

### 19.1.2 Endpoints & flow

| Purpose | URL |
|---|---|
| Public ws | `wss://ws.api.paradex.trade/v1` |
| Private ws | same socket; jsonrpc `auth` frame with the Bearer JWT |
| JWT mint | `POST /auth/v1/auth` — Request struct signature in PARADEX-* headers |
| Orders | `POST /v1/orders`, cancel `DELETE /v1/orders/{id}`, modify `PUT /v1/orders` |

Startup sequence: account ws defers -> http client mints the JWT (signature over `Request{POST,/v1/auth,"",timestamp,ex` -> response routes through EMSG_AUTHENTICATION -> deferred socket connects -> jsonrpc auth frame -> chained `orders.<account>` / `fills.<account>` subscriptions.

### 19.1.3 Credentials

- `apiKeyRef`: Starknet account address (`0x...`)
- `apiSecretRef`: account private key (64 hex chars)

### 19.1.4 Correlation

`client_id` echoes through order events and fills — resolved through the order store by decimal client id like every other adapter; foreign orders logged and dropped.

## 19.2 Testnet

Full testnet environment with separate Starknet chain and keys:

| Purpose | URL |
|---|---|
| WS | `wss://ws.api.testnet.paradex.trade/v1` |
| REST (auth, orders) | `https://api.testnet.paradex.trade` |

**Domain differs**: the SNIP-12 domain `chainId` is the testnet chain id (`PARADEX_STARKNET_TESTNET`) instead of mainnet - point `domain_s::strChainId` at the testnet value for the run (see hyperliquidAuth-style flip note in the certification plan).

# Chapter 20

# Phemex

2-file design: exchangeProtocol.h + CMakeLists.txt

Shared infrastructure in shared/exchanges/ - exchgPlugin.cpp/h, exchgManagerMainT.h, exchgManager*T.h templates

Protocol: exchangeProtocol_s with constexpr bool flags

Marketdata: implemented (USD(S)-M perpetuals) Account: stubs (tier T3, RSA/HMAC per endpoint class) Trading: stubs (tier T3)

Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (fMarketdata=true only); account/trading sub-manager startup is gated on those bits. See docs/01-system-protocol/notification-pipeline.md.

## 20.1 API Reference

- https://phemex-docs.github.io/

## 20.2 WebSocket Details

- Endpoint: `wss://ws.phemex.com`
- Heartbeat: `{"id":0,"method":"server.ping","params":[]}` -> `{"error":null,"id":0,"result":"pong"}`
- Subscribe uses JSON-RPC style method/params pattern
- Public market data does not require authentication

## 20.3 Instrument Discovery

- REST `GET /public/products`
- Filters `perpProductsV2` (or `perpProductsV1`) array for perpetual contracts
- Type filter: `Perpetual`

## 20.4 exchangeProtocol_s organization

Follows the shared protocol block ordering defined in Bitfinex reference adapter.

1. identity and capability flags
2. message logging and classification
3. session/control parsers (`processSubscribed`, `processUnsubscribed`)

4. marketdata parsers (`processInstrumentDetails`, `processTrade`, `processUpdateBook`, `processBookSnapshot`)
5. account/trading parsers (stubs)
6. request builders (instrument discovery, subscribe/unsubscribe, heartbeat, historical)
7. historical response parser
8. private helper methods
9. static protocol state

## 20.5 Testnet

Phemex provides a public testnet:

| Purpose | URL |
|---|---|
| WS | `wss://testnet-ws.phemex.com/ws` |
| REST | `https://testnet-api.phemex.com` |

Relevant once the RSA/HMAC private stack lands (T3).

# Chapter 21

# Poloniex

## 21.1 Architecture

2-file design using shared template infrastructure:

```
exchanges/Poloniex/
  exchangeProtocol.h     — exchange-specific wire protocol + flags
  CMakeLists.txt         — build rules
```

## 21.2 Protocol flags

Transport: WebSocket (public marketdata) + HTTP (trading, account requests). Capabilities are declared in `exchangeProtocol_s::ce_capabilities` (`exchgCapabilities_s`): `fMarketdata`, `fAccount`, `fTrading` true; all order-type/time-in-force bits false — modify is emulated by the order manager (cancel + replace) until a native venue path lands. Capabilities are exported at plugin load (`getExchgCapabilities`); they are not config keys.

## 21.3 Implementation status

- **marketdata**: trades, tickers/BBO, book, book_lv2 snapshot/update
- **account**: private websocket auth (HMAC-SHA256), wallet/balance parsing real; balance updates route through the order-manager hub (`EEV_2ORD_ASSET_UPDATE` -> `ESRV_ORDER`)
- **trading**: HTTP order create/cancel real; execution parsing real over the private `trade` channel (fills correlated by `clientOrderId`/`orderId`, external fills logged and dropped); order-state parsing over the `orders` channel; modify emulated by the order manager (cancel + replace)

Config: `exchange.cfg` in this folder. Cert path: simulation cert with live marketdata -> testnet -> production.

See `docs/01-system-protocol/notification-pipeline.md` for the notification flow.

## 21.4 exchangeProtocol_s organization

Use the shared protocol block ordering so adapters stay consistent and easy to maintain: `exchanges/exchgLighter/exchang` is the functional reference implementation; `exchanges/exchgBitfinex/exchangeProtocol.h` the layout skeleton. All adapters should keep the same block order and expose empty account/trading stubs when a complete shared-manager skeleton is required. Instrument discovery and historical parsing are intentionally separate from live marketdata parsing.

1. identity and capability flags
2. message logging and classification
3. session/control parsers (`processServerInfo`, `processSubscribed`, `processUnsubscribed`, `processLoginAck` when used)
4. marketdata parsers (`processInstrumentDetails`, `processTrade`, `processBbo`, `processUpdateBook`, `processBookSnapshot`)
5. account/trading parsers
6. request builders (instrument discovery, subscribe/unsubscribe, heartbeat/auth, historical)
7. historical response parser
8. private helper methods
9. static protocol state

Naming convention: protocol receive/ack handlers use `processXXXX` (not `handleXXXX`).

## 21.5 Historical data

`buildHistoricalDataRequest` uses `/v3/market/candles`, which caps `limit` at 500. The shared request default (`_OHLC_ITEM_REQUEST + 2` = 512) exceeds that, so the builder clamps to 500.

## 21.6 Testnet

Poloniex does not offer a public testnet for the trading API. Gate G2 falls back to minimum-size validation on production after internal review of the signed request builders (document deviations in this README as they surface).

# Part IV

# Part IV - Algorithms

# Chapter 22

# Trading Algorithms

## 22.1 Overview

Trading algorithms are compiled as shared-library plugins and loaded dynamically by `storePlugin_c`. Each algo derives from `algoFramework_c`, which provides composable managers for state management and lifecycle callbacks for market data, orders, positions, and timers.

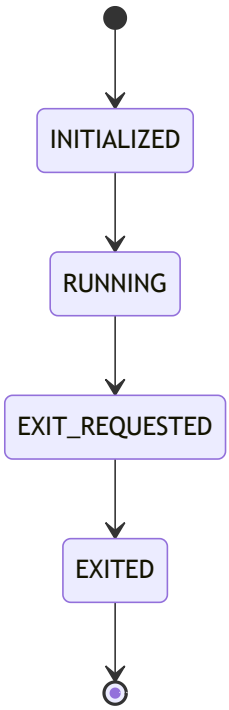
## 22.2 Available Algorithms

| Algorithm | Type | Description | Files |
|---|---|---|---|
| `algoHUNT` | Production | Fades stop-hunts (a structural liquidity pool is swept then reclaimed) with an online neural meta-controller that re-tunes parameters from live outcomes | `algos/algoHUNT/` |
| `algoDELEV` | Production | Positioning-stress liquidity provision: fades a crowded leveraged side after deleveraging exhaustion (crypto perps) | `algos/algoDELEV/` |
| `algoQIS` | Production | Regime-conditional microstructure with online survival models (crypto perps) | `algos/algoQIS/` |
| `algoHERD` | Production | Fades short bursts of uninformed same-direction aggressive flow (IBKR CME micro/nano index, L1 trades + BBO) | `algos/algoHERD/` |
| `algoHUB` | Production | Trades the structural basis between CME E-nano S&P (NES satellite) and E-mini S&P (ES hub) | `algos/algoHUB/` |
| `algoMIRE` | Production | Refill-hazard + trade self-excitation flow resolution; bounded online ML (IBKR CME nano, configurable) | `algos/algoMIRE/` |
| `algoMLD` | Production | Bounded online-ML intraday on completed OHLC bars with BBO protection (perps/futures) | `algos/algoMLD/` |
| `algoMCT` | Production | Microstructural Coherence Trading — quantum-inspired density-matrix purity signals with pyramiding | `algos/algoMCT/` |

| Algorithm | Type | Description | Files |
|---|---|---|---|
| algoHarvest | Production | Mean-reverting reaction harvest (funding/carry, crypto) | algos/algoHarvest/ |
| algoExchangeTest | Certification | Full order-lifecycle exchange certification suite (safety-first defaults) | algos/algoExchangeTest/ |
| algoIndicatorTest | Certification | Indicator V2 runtime integration certification | algos/algoIndicatorTest/ |

## 22.3 Algorithm Framework

algoFramework_c (include/ttTrader/algo/algoFramework.h) is the abstract base class that all algorithms derive from. It follows a state machine:


INITIALIZED
RUNNING
EXIT_REQUESTED
EXITED

### 22.3.1 Composable managers

The framework composes these managers, accessible as protected members:

| Manager | Purpose |
|---|---|
| `algoAssetManager_c` | Multi-asset balance tracking, margin |
| `algoTimerManager_c` | Timers and scheduled callbacks |
| `algoInstrumentManager_c` | Instrument subscription and data access |
| `algoOrderManager_c` | Order creation, modification, cancellation |
| `algoRiskCalculator_c` | Risk metrics and position sizing |
| `algoPositionManager_c` | Position tracking and PnL |
| `algoStopLossCalculator_c` | Stop-loss conditions (stubbed) |
| `algoProfitProtection_c` | Trailing stop / profit protection (stubbed) |
| `algoUtilityFunctions_c` | Shared utility functions |

The `algoStopLossCalculator_c` and `algoProfitProtection_c` managers are currently stubbed: their calculations return `0_dec` (stop loss) or take no action (profit protection) until the OHLC/indicator lookups are implemented. Treat a zero stop-loss result as "not configured", never as "stop at zero".

### 22.3.2 Virtual callbacks

Derived algos override these to implement strategy logic:

**Lifecycle**

```
virtual bool onInitAlgo();
virtual void onExitAlgo();
virtual bool canStart();
virtual bool canExit();
```

**Market data**

```
virtual void onMarketdataTick( instrumentInfo_s*, const marketdataTrade_s& );
virtual void onMarketdataBbo( instrumentInfo_s*, const marketdataBbo_s& );
virtual void onMarketdataBook( instrumentInfo_s*, const marketdataBook_s& );
virtual void onMarketdataOhlcClose( ohlcInfo_s* );
```

**Orders**

```
virtual void onOrderCreated( orderInfo_s* );
virtual void onOrderFill( orderInfo_s*, const executionInfo_s& );
virtual void onOrderCanceled( orderInfo_s* );
virtual void onOrderFailed( orderInfo_s* );
virtual void onOrderUpdate( orderInfo_s* );
```

Every `orderInfo_s` carries its lifecycle in `infoState.eState` / `infoState.eProcessingState`, both typed `orderProcessingState_e`. Callbacks receive the order **after** the manager has already moved it into a terminal state (`executed`, `canceled`, `failed`) or a settled active state — algos should treat `isTerminalState()` as absorbing and never re-submit a terminal order. See `03-architecture`.

**State**

```
virtual void onPositionRegister( positionInfo_s* );
virtual void onAssetUpdate( assetInfo_s* );
```

**Timers**

```
virtual void onTimer( uint16_t timerId );
virtual void onTimerFireOnce( uint16_t timerId );
virtual void onTimerAlert( uint16_t timerId );
```

### 22.3.3 Balance shares and PnL isolation

When several algos run against one account, each algo's buying power is isolated: a losing algo must never be able to size against a profit another algo made on the shared balance. Two mechanisms combine - a **per-asset share** declared by each algo, and a **first-snapshot allocation** that then moves only with that algo's own realized PnL.

#### 22.3.3.1 Declaring the share

The share lives in the algo's `assets` list, next to the asset it applies to, not in `parameter`. It is per asset, so one algo can take 20 % of a USD account and 50 % of a EUR account:

```
"assets": [
    { "id": "assetLighterUSDC", "balanceFraction": 0.20 }
]
```

`storeAlgo_c` resolves every share after all enabled algos are parsed:

| Case | Result |
|---|---|
| `balanceFraction` set | taken verbatim; `0` = listed but read-only (consumes no balance) |
| entry present without `balanceFraction` | splits the remaining budget equally with the other unconfigured entries |
| unconfigured entry with no budget left (`sum(explicit) == 100 %`) | config error: log + `requestApplicationExit()` |
| `sum(explicit) > 100 %` for an asset | config error: log + `requestApplicationExit()` |
| bare asset id (`"assetLighterUSDC"`) | no configured share (defaults, same as an omitted key) |

Three algos on one asset - one at `0.5`, two unconfigured - resolve to `0.5, 0.25, 0.25`. Two unconfigured algos resolve to `0.5, 0.5`. Enabled algos only; an algo that needs an asset but not its balance must list it with `"balanceFraction": 0`.

#### 22.3.3.2 Sizing uses an isolated running balance

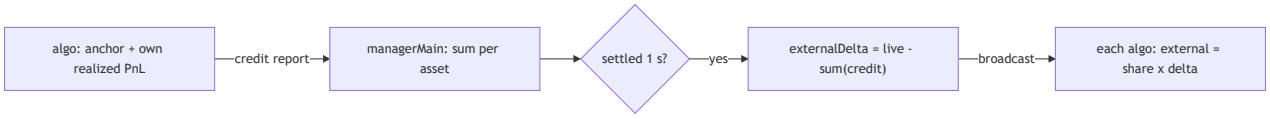
On the **first asset balance snapshot** each algo anchors `allocation = liveAvailable x share` (margin and spot tracked separately). Sizing - the max-loss basis and the notional affordability cap - then uses:

`available = allocation + sum(own realized PnL of closed trades, net of fees)`

The live account balance is never re-read for sizing, so a winner's profit never inflates a loser's headroom. A restart re-anchors to the then live balance. Algos that declare no assets at all keep the legacy single fraction (full balance).

#### 22.3.3.3 External deposits and withdrawals

Mid-run deposits/withdrawals are separated from aggregate PnL and rescaled proportionally, without breaking isolation:




Each framework posts a per-asset credit report (`anchor + realized`) whenever it anchors or closes a trade. `managerMain_c` aggregates them and, on a 1 s settling pass that waits for reports to stop arriving, computes

`externalDelta = liveAvailable - sum(credit)`. That difference is a pure deposit/withdrawal; it is broadcast and each algo applies its own `share x externalDelta`. The settling wait is what stops an in-flight realized PnL from being read as an external flow.

### 22.3.4 Steps

1. Create the directory and files following an existing algo (e.g. `algoMCT/` for a compact example, `algoHUNT/` for the full framework) as a template.
2. Derive from `algoFramework_c` and implement the virtual callbacks needed by the strategy.
3. Define a config struct in `algoConfig.h` that reads algorithm parameters from the JSON config.
4. Add a `CMakeLists.txt` following the existing pattern.
5. Reference the new algo in the top-level `algos/CMakeLists.txt`.

### 22.3.5 Example skeleton

```
#include <algoFramework.h>

class algoCore_c : public algoFramework_c
{
public:
    algoCore_c()  = default;
    ~algoCore_c() = default;

protected:
    bool onInitAlgo() override;
    void onExitAlgo() override;

    void onMarketdataBbo( instrumentInfo_s* p_instr, const marketdataBbo_s& p_bbo ) override;
    void onMarketdataOhlcClose( ohlcInfo_s* p_ohlc ) override;
    void onTimer( uint16_t p_timerId ) override;

    void _evaluateEntry( instrumentInfo_s* p_instr );
    void _evaluateExit( instrumentInfo_s* p_instr );
};
```

### 22.3.6 Configuration

Algo config is JSON-deserialised in `algoConfig.cpp`. Parameters are stored in a config struct and accessible during `onInitAlgo()`.

Example JSON config entry:

```
{
    "algos": [
        {
            "algoId": 10,
            "id": "algoMyAlgo",
            "enabled": true,
            "name": "algoMyAlgo",
            "info": "algoMyAlgo",
            "parameter": {
                "instTradeId": "instLighterBTC",
                "ohlcId5m": "instLighterBTC-5m",
                "risk": { "type": "PERCENT", "value": 0.20 }
            },
```

```
            "assets": [
                { "id": "assetLighterUSDC", "balanceFraction": 0.20 }
            ],
            "instruments": [
                {
                    "instrumentId": "instLighterBTC",
                    "enabled": true,
                    "book": 10,
                    "ohlc": [ "instLighterBTC-5m" ]
                }
            ]
        }
    ]
}
```

The algo plugin (loaded from the `plugins` array, matched by `algoId`) reads `parameter` in `onSetParams()` / `algoConfig.cpp`; `assets` (balances and their shares) and `instruments` are subscribed by the framework before the algo runs.

## 22.4 Algorithm Lifecycle

1. **Plugin loaded** — `storePlugin_c` loads the DLL, calls `createPlugin()`.
2. **Config applied** — `onSetParams()` receives the JSON params object.
3. **Init** — `onInitAlgo()` validates config, subscribes to instruments, sets up timers.
4. **Run** — Market data callbacks fire. The algo evaluates entry/exit signals and places/modifies orders.
5. **Exit requested** — `onExitAlgo()` cancels open orders, unregisters.
6. **Plugin unloaded** — `storePlugin_c` calls `destroyPlugin()`.

## 22.5 algoMCT — Microstructural Coherence Trading

The most sophisticated algorithm in the platform. Key features:

- **Density-matrix construction** from BBO updates within a time window
- **Purity-driven signals** — high purity indicates coherent BBO dynamics
- **Pyramiding** with purity ratchet, tiered PnL cushion, and diminishing sizing
- **Coherence-weighted take profit** with per-layer TP management
- **PnL-aware signal exit** with limit-at-mid and timed market fallback

Full documentation: `algos/algoMCT/README.md`.

# Chapter 23

# algos/

feature root for concrete algo plugins (`algoHUNT/`, `algoMLD/`, `algoHarvest/`, ...). each algo folder is one DLL with its own `config/` subdirectory holding its runtime configs — algo runtime configs belong here, never in an exchange folder.

the framework every algo links against lives in ../shared/algos/ with its contract header ../include/ttTrader/algo/. per-algo details are documented in each folder's own `README.md`.

algos sharing one account declare their buying power per asset in the algo's `assets` list (`{ "id": "asset-LighterUSDC", "balanceFraction": 0.20 }`), not in `parameter`; `storeAlgo_c` resolves the shares and each algo then sizes from its first-snapshot allocation plus only its own realized PnL. see ../docs/05-trading-algorithms/README.md.

# Chapter 24

# algoDELEV — deleveraging-exhaustion fade

Positioning-stress liquidity provision for crypto perpetual futures. The strategy detects a crowded leveraged side (funding rate + mark-index premium + open-interest velocity), waits through the violent phase of the forced unwind, and fades the flush only when the forced flow measurably decelerates (OI acceleration + taker-flow decay) with a bounded retrace of the extreme. The stop sits beyond the flush extreme; the take-profit ladder is ATR/cost gated. An online logistic model (AdaGrad, refreshed from matured first-touch labels) gates entries and scales size.

## 24.1 Components

| File | Responsibility |
|---|---|
| `algoConfig.h/.cpp` | parameter parse with clamp envelopes and cross-field repair |
| `positioningTracker.h/.cpp` | funding Welford, OI minute grid + d5 stats, premium/spread rings, one-second flow/mid buckets, ATR day-EMA, staleness, OI-jump latch |
| `featureBuilder.h/.cpp` | the 13 clipped, scale-free model features |
| `episodeTracker.h/.cpp` | deterministic episode state machine (IDLE / ARMED / STRESS / EXHAUSTION), bar-based |
| `mlModel.h/.cpp` | AdaGrad online logistic model, calibration EMA, pending label ring, first-touch labeling |
| `regimeDetector.h/.cpp` | QUIET / ACTIVE / TURBULENT / DATA_LOST risk regime with hysteresis and vol-extreme freeze |
| `positionSizer.h/.cpp` | sizing stack and pure risk envelopes (caps, stop/TP geometry) |
| `sessionGuard.h/.cpp` | daily flatten, no-entry window, daily limits, funding blackout |
| `orderPath.h` | pure post-only-retry / partial-adoption / protection-restore decisions |
| `algoCore.h/.cpp` | plugin wiring, bar pipeline, entry/exit order flow, state persistence |

## 24.2 Signal pipeline

1. Every venue feed updates the positioning tracker (funding, stats/mark/index/OI, trade tape, BBO). Every closed 1m bar feeds ATR; every closed 5m bar feeds ADX.
2. At bar close: snapshot -> regime -> episode -> 13 features -> mature pending labels (train + calibration) -> capture a new candidate -> evaluate an entry.
3. Entries require an EXHAUSTION window, model readiness (`minTrainSamples`), calibration, session/spread/cooldown/regime gates and the cost gate.

4. Entry = post-only LIMIT at the touch (one plain-limit retry on rejection); fill -> native stop-market beyond the flush extreme + TP1 partial close-limit.
5. TP1 fill -> cancel the full stop, re-arm at breakeven + round-trip cost, place TP2 toward `min(anchorEpisode, entry +/- kTp2 x ATR)`.
6. Exits: signal decay (confirmed), max hold, opposing crowd episode, stale positioning feed flatten, session flatten, daily loss / equity / API kills. Any close that flattens the book cancels every still-working close sibling.

## 24.3 Parameters

All keys live in the algo `parameter` object; missing keys keep the defaults, out-of-range values are clamped. See `config/algoDELEV-lighter-btc-perp.json` for the shipped values.

| Key | Default | Range | Meaning |
|---|---|---|---|
| `instTradeId`, `ohlcId1m`, `ohlcId5m`, `atrIndicatorId`, `adxIndicatorId` | required | config ids | identity binding, fails closed |
| `risk` | `PERCENT 0.25` | framework | consumed before `onSetParams` |
| `sessionMode` | `daily` | daily/continuous | flatten discipline |
| `flattenMinuteUtc` | `1430` | 0-1439 | 23:50 UTC flatten |
| `noEntryMinutes` | `15` | 0-1440 | no late risk |
| `fundingSkirtSeconds` | `10` | 0-600 | entry blackout around next settlement |
| `dailyMaxTrades` | `10` | 1-1000 | session trade cap |
| `dailyMaxLossPct` | `0.02` | 0-0.1 | daily loss kill (fraction) |
| `cooldownAfterLossSec` / `cooldownBackoff` | `900` / `2` | 60-86400 / 1-4 | anti-revenge, doubled per consecutive loss (2 h cap) |
| `entryThreshold` | `0.60` | 0.50-0.90 | model gate baseline |
| `exitThreshold` | `0.42` | 0.30-0.55, < entry | signal decay gate |
| `exitConfirmSeconds` | `15` | 1-120 | decay debounce |
| `minTrainSamples` | `150` | 10-5000 | cold-start / corrupt-state lockout |
| `sampleMinSpacingBars` | `3` | 1-60 | candidate capture rate |
| `labelHorizonBars` | `45` | 5-240 | label horizon |
| `learnRate` / `l2` / `adagradEps` / `weightClip` | `0.10` / `0.0001` / `1e-6` / `4.0` | see config | AdaGrad hyper-parameters |
| `minCalibration` | `0.45` | 0.30-0.70 | reliability freeze |
| `crowdArmMin`, `fundArmZ`, `premArmZ`, `oiArmZ` | `0.5`, `1.5`, `1.5`, `1.0` | see config | episode arming |
| `flushMinAtr` / `flushMaxAtr` | `1.5` / `8.0` | 0.5-5 / 3-30 | minimum overshoot / runaway abandonment |
| `toxMin` / `oiAccelMin` / `flowDecayMin` | `0.15` / `0.5` / `0.25` | see config | exhaustion conditions |
| `qLo` / `qHi` | `0.08` / `0.75` | 0-1, lo<hi | recovery window |

| Key | Default | Range | Meaning |
|---|---|---|---|
| minStressBars, exhaustMaxBars, episodeMaxBars, armTimeoutBars, entryCooldownBars, newExtremeAtr | 3, 30, 240, 120, 15, 0.5 | see config | episode timing |
| regimeHysteresisBars, quietVolPct, turbVolPct, adxTurb | 3, 0.25, 0.90, 35 | see config | regime classifier |
| allowEntryInVolExtreme | false | bool | exhaustion override of the vol freeze |
| stopPadAtr / stopAtr / maxStopBps | 0.6 / 1.5 / 150 | see config | structural + ATR stop |
| tpFrac / kTpCap / maxTpAtr | 0.5 / 2.0 / 4.0 | see config | TP1 geometry |
| kTp2 | 3.0 | 1-8 | TP2 ATR cap |
| partialTpPct | 0.5 | 0-0.9 | TP1 tranche (auto-falls back to full size) |
| costMult | 4 | 1-10 | TP >= costMult x round-trip cost |
| confidenceFloor | 0.35 | 0-1 | pFactor floor |
| maxHoldS | 2700 | 60-14400 | time stop |
| profitProtectionEnabled / profitProtectionLockBps | false / 5.0 | bool / 0-500 | optional trailing lock (never-give-up-profit) |
| killSpreadBps | 35 | >= 0 | spread entry block |
| maxNotionalPerTrade / maxGrossLeverage | 25000 / 1.5 | see config | hard caps (applied last) |
| enableSettlementArbitrage | false | bool | reserved, not implemented in v1 |
| reduceOnlyClose | true | bool | venue reduce-only flag on close orders |
| stateFile | — | path | JSON persistence path; written on trade close, halt and clean exit, plus a periodic checkpoint (first heartbeat, then every 5 min) so warm-up progress, the daily session counters (trade count, loss latch, session index) and the post-loss cooldown/backoff survive a kill |

## 24.4 Run

```
ttTrader.exe algos/algoDELEV/config/algoDELEV-lighter-btc-perp.json --log --logmarketdata
ttTrader.exe algos/algoDELEV/config/algoDELEV-lighter-btc-perp.json --log --tradingMode paper --logmarketdata
```

Build (from the repo root with the VS dev shell):

```
. "C:\Program Files\Microsoft Visual Studio\18\Community\Common7\Tools\Launch-VsDevShell.ps1" -Arch amd64 -HostArch amd64 -SkipAutomaticLocation
cmake --build .build/clang-debug --target algoDELEV
```

Unit tests:

```
cmake --build .build/clang-debug --target ttManagerOrderTests
ttManagerOrderTests.exe Delev
```

## 24.5 Invariants

- All heavy state is fixed-size `std::array`; allocation-free after init; `decimal_t` everywhere except inside the Eigen model boundary.
- No RNG; identical feed sequences produce identical state.
- Indicator/OHLC reads use the V2 exact-sequence API (`findOhlc`, `findIndicator`, `getValueAtSequence`); labels/features never read by offset.
- One instrument per algo instance; positions are keyed by the owning algo's config id inside the order manager, so several algo instances can run the same instrument without position leakage.
- `reduceOnlyClose` defaults to `true`; a combined multi-algo config sets it `false` and relies on the sibling-order hygiene (every close/flat cancels working close orders) to avoid orphan legs.
- A corrupt state file falls back to a fresh model and entries stay locked until `minTrainSamples` matured again.
- Never a naked position: a lost protection leg is re-placed; a failed market close reverts to the in-position state and retries.

## 24.6 Open validation gates

- Lighter funding interval: the adapter's `durInterval` must be verified against live `market_stats` frames before trusting `fundZ`/cost math.
- `.ttpb` v3 playback (funding/stats events) does not exist yet; deterministic end-to-end replay certification requires it.
- Partial reduce-only closes on Binance use the documented full-size TP1 fallback.
- `enableSettlementArchetype` is config-gated off and not implemented in v1.

# Chapter 25

# algoExchangeTest v2.0 - Exchange Certification Suite

## 25.1 Overview

`algoExchangeTest` is a comprehensive exchange certification suite designed to validate the full order lifecycle against any exchange protocol. It is **safety-first by default**, suitable for use on live accounts with real funds – all non-market orders are placed far from the market and kept open for `holdSeconds` (default 20 seconds) so they can be verified manually, then cancelled. Set `holdSeconds: 0` to cancel immediately after creation verification.

The suite runs **27 test cases** sequentially, covering live marketdata validation (trades, BBO, book, venue candles), every order type, modification, error handling, position management, and advanced scenarios.

## 25.2 Safety Design

- **Safe offset**: All non-market orders are placed at `bboOffsetBps` from the market (default: 7500 BPS = 75%). Orders rest open for `holdSeconds` (default 20 seconds) for manual verification, then are cancelled automatically.
- **Verification hold**: When `holdSeconds > 0`, created orders (including modified orders) are held resting for the configured period before the cancel is requested. Manual/exchange cancellation during the hold completes the test with a PASS; a full fill during the hold also completes the test (the lifecycle is verified). Set `holdSeconds: 0` for immediate cancel.
- **Fail-fast**: By default, any test failure immediately halts all remaining tests and marks them as SKIPPED.
- **Market order gate**: Market order and execution tests are **disabled by default**. Set `enableMarketOrders: true` explicitly to opt in.
- **Configurable timeout**: Per-test watchdog timer (default: 60s) with automatic retry on transient timeouts.
- **Order cleanup**: On failure or exit, any remaining open orders are cancelled.

## 25.3 Test Cases

### 25.3.1 Phase 0 — Marketdata Validation (no orders placed)

| # | Test | Description |
|---|---|---|
| 01 | Marketdata Trades | Audit live trade stream: price/size positive, timestamp present and not in the future; pass after 10 clean ticks, any invalid tick fails immediately |
| 02 | Marketdata BBO | Audit live BBO: bid/ask prices and sizes positive, timestamp valid, strict `bid < ask`; pass after 10 clean updates |
| 03 | Marketdata Book | Audit published book top levels: prices/sizes positive (zero sizes never accepted), bids strictly descending, asks strictly ascending, no crossed top; pass after 5 clean snapshots |
| 04 | Marketdata Candles | Audit the venue kline subscription: every finalized exchange candle must be priced and strictly monotonic in sequence. Closes are recorded passively from suite start (tests 01-03 already run while the series builds), so this test passes on the first **fresh** close after it starts — with a reduced single-stage wait of `candleWaitSeconds` (default 90 s, no extension). Only venue-authored candles count (`buildMode: "exchangeOnly"`, or `"auto"` after the venue stream proved alive); locally aggregated fallback candles never satisfy this test. Skipped with a clear reason when the instrument has no exchange-candle series, or when an auto-mode series has seen no venue frame since suite start (the venue does not stream klines); fails when the venue accepts the kline subscription but never closes a candle. |

**Candle test configuration**: the instrument needs an exchange-stream series and the algo must subscribe it:

```
"instruments": [ {
  "id": "instBinanceBTC", "exchgId": "exchgBinance", "idAtExchg": "BTCUSDT",
  "ohlc": [ { "id": "instBinanceBTC-1m-xchg", "source": "exchange", "buildMode": "exchangeOnly", "t
} ],
"algos": [ { "...": "...", "instruments": [ { "instrumentId": "instBinanceBTC", "book": 10, "ohlc":
```

**Venues without adapter kline support**: either omit the exchange ohlc series (the test skips with "no exchange-candle series configured"), or declare the series with `"buildMode": "auto"` — the series then aggregates live windows from trades, and the test self-skips with "venue does not stream klines" while no venue frame

has arrived (Bitstamp uses this; its series is additionally seeded from the REST `/api_v2/ohlc` endpoint). Avoid `"skipTests": "Marketdata Candles"` pins — the suite reports the reason itself.

**Known venue behavior**: a venue that accepts a `@kline` subscription but never pushes frames fails this test honestly after `candleWaitSeconds`. The former Binance-futures zero-frame defect is fixed: Binance futures streams klines and the venue certifies 27/27 (2026-09-06).

### 25.3.2 Phase 1 — Basic Order Types (safe)

| # | Test | Description |
|---|---|---|
| 04 | Limit Buy Order | Place limit buy far below bid, verify created, cancel |
| 05 | Limit Sell Order | Place limit sell far above ask, verify created, cancel |
| 06 | Stop Market Buy Order | Place stop market buy with trigger above market, verify created, cancel |
| 07 | Stop Market Sell Order | Place stop market sell with trigger below market, verify created, cancel |
| 08 | Stop Limit Buy Order | Place stop limit buy with trigger+limit above market, verify created, cancel |
| 09 | Stop Limit Sell Order | Place stop limit sell with trigger+limit below market, verify created, cancel |
| 10 | Trailing Stop Buy Order | Place trailing stop buy with delta, verify created, cancel |
| 11 | Trailing Stop Sell Order | Place trailing stop sell with delta, verify created, cancel |
| 12 | GTD Order | Place Good-til-Date limit order (`gtdExpirySeconds`, simulation default 60s), verify created with correct TIF, system cancel at expiry |
| 13 | GAD Order | Place Good-after-Date limit order (`gadSeconds`), verify created, cancel. **Disabled by default** - set `enableGadTests: true` to exercise the manager's deferred queue (promotion at activation, expiry of a stale GTD out of the queue) |
| 14 | Post-Only Limit Order | Place limit order with post-only flag, verify created, cancel |

### 25.3.3 Phase 2 — Order Modification

| # | Test | Description |
|---|---|---|
| 15 | Modify Limit Price | Place limit order, modify price, verify update ack, cancel |
| 16 | Modify Order Size | Place limit order, modify size, verify update ack, cancel |

| # | Test | Description |
|---|---|---|
| 17 | Modify Price & Size | Place limit order, modify price and size in one request, verify update ack, cancel |

### 25.3.4 Phase 3 — Market Orders (disabled by default)

| # | Test | Description |
|---|---|---|
| 18 | Market Buy Order | Place market buy, verify fill execution |
| 19 | Market Sell Order | Place market sell, verify fill execution |

### 25.3.5 Phase 4 — Execution & Position Management

| # | Test | Description |
|---|---|---|
| 20 | Execution Verification | Verify fill data integrity (price, size, count) from executed orders |
| 21 | Position Close | Close any open position via market close order |

### 25.3.6 Phase 5 — Error Handling

| # | Test | Description |
|---|---|---|
| 22 | Invalid Order - Zero Size | Submit order with zero size, verify rejection error path |
| 23 | Invalid Order - Below Min Size | Submit order one size tick below the min, verify rejection |
| 24 | Invalid Order - Off-Grid Size | Submit order off the size grid, verify rejection (protocol cannot represent the order) |
| 25 | Invalid Order - Off-Grid Price | Submit order off the price grid, verify rejection (protocol cannot represent the order) |
| 26 | Cancel Nonexistent Order | Cancel using bogus user value, verify clean handling |

### 25.3.7 Phase 6 — Advanced Scenarios

| # | Test | Description |
|---|---|---|
| — | OCO Pair *(planned)* | Create synced buy+sell OCO pair, cancel one, verify both cancelled |
| — | Last in Sequence *(planned)* | Create order, modify before cancel, verify full lifecycle with intermediate state changes |

## 25.4 Session Log Output

Each test produces detailed process logging with descriptive names, phase transitions, and clear PASS/FAIL/SKIP status:

```
[CERT 01/27] Marketdata Trades - starting: verify live trade stream
[CERT 01/27] Marketdata Trades - PASSED
[CERT 05/27] Limit Buy Order - order placed: size=0.001 limit=45000.00 offset=45000.00 bps
[CERT 05/27] Limit Buy Order - order created: id=1:0
[CERT 05/27] Limit Buy Order - order held open - resting 20 s for manual verification
...
[CERT 05/27] Limit Buy Order - hold period elapsed - cancel requested
[CERT 05/27] Limit Buy Order - order canceled: id=1:0
[CERT 05/27] Limit Buy Order - PASSED
[CERT 02/27] Marketdata BBO - starting: verify live BBO stream
...
```

Final summary:

```
==========================================
  EXCHANGE CERTIFICATION SUITE SUMMARY
==========================================
  Total:   27
  Passed:  24
  Failed:  1
  Skipped: 2
==========================================
  [PASS] Limit Buy Order
  [PASS] Limit Sell Order
  ...
  [FAIL] Trailing Stop Buy Order
  [SKIP] Marketdata Book
  [SKIP] Market Buy Order
==========================================
  RESULT: NOT CERTIFIED
==========================================
```

## 25.5 Parameters

| Parameter | Type | Default | Description |
|---|---|---|---|
| `instTradeId` | string/id | required | Instrument config ID for trading |
| `orderSize` | decimal | min size | Fixed order size (0 = use instrument minimum) |
| `bboOffsetBps` | uint | 7500 | Limit/trigger price offset in basis points (min 100) |
| `testTimeoutMs` | uint | 60000 | Per-test timeout in milliseconds (min 5000) |
| `holdSeconds` | uint | 20 | Seconds each non-market order rests open before cancel (0 = cancel immediately) |

| Parameter | Type | Default | Description |
|---|---|---|---|
| `gtdExpirySeconds` | int | 360 | GTD expiry in seconds. 0/omitted keeps the venue-safe default (lighter-go minimum 5 min, long cancel-latency watchdog). Simulation cert runs set e.g. 60 - the manager enforces the expiry locally, no venue minimum applies |
| `failFast` | bool | true | Halt all remaining tests on first failure |
| `enableMarketOrders` | bool | false | Enable market order tests (WARNING: executes real trades) |
| `enableExecutionTests` | bool | false | Enable execution verification and position close tests |
| `enableGadTests` | bool | false | Enable GAD order test (currently only exercises the local deferred-queue simulation) |
| `skipTests` | string | "" | Comma-separated test names to skip |
| `onlyTests` | string | "" | Comma-separated test names to run exclusively |

## 25.6 Simulation Certification

The suite doubles as the order-manager simulation test. Set `"tradingMode": "simulation"` in `config/algoExchangeTest.cfg` (Lighter mainnet marketdata stays live; the platform itself becomes the venue). In simulation the adapter-declared venue capabilities (`exchgCapabilities_s`) are ignored: stops, trailing stops and GTD/GAD boundaries are all emulated in-process, so the run exercises exactly the manager-side paths production uses on a non-native exchange. Market orders and execution tests execute no real trades in simulation and can be enabled (`enableMarketOrders`, `enableExecutionTests`) for full coverage.

Notes:

- The GTD test completes via manager-side expiration at the +360s expiry (~6 minutes) instead of the chain-side pull on Lighter (~56 minutes on-chain).
- The GAD test (with `enableGadTests: true`) validates the local deferred queue: promotion at activation and expiry of a stale GTD right out of the queue.
- Watchdogs are one-shot per phase: a timeout means the phase never completed and fails the test.

## 25.7 Config Examples

### 25.7.1 Default (safe certification)

```
{
  "parameter": {
    "instTradeId": "instLighterBTC",
    "orderSize": "0.001",
```

```
    "bboOffsetBps": 7500,
    "testTimeoutMs": 60000,
    "holdSeconds": 20,
    "failFast": true
  }
}
```

### 25.7.2 Full certification with market orders

```
{
  "parameter": {
    "instTradeId": "instLighterBTC",
    "orderSize": "0.001",
    "bboOffsetBps": 7500,
    "testTimeoutMs": 120000,
    "failFast": true,
    "enableMarketOrders": true,
    "enableExecutionTests": true
  }
}
```

### 25.7.3 Run specific tests only

```
{
  "parameter": {
    "instTradeId": "instLighterBTC",
    "onlyTests": "Limit Buy Order,Limit Sell Order,OCO Pair"
  }
}
```

### 25.7.4 Skip problematic tests

```
{
  "parameter": {
    "instTradeId": "instLighterBTC",
    "skipTests": "GTD Order,GAD Order"
  }
}
```

## 25.8 Fail-Fast Mechanism

When `failFast` is enabled (default) and any test fails:

1. The failing test is marked `FAILED` with the reason logged
2. Any remaining pending tests are immediately marked `SKIPPED (fail-fast)`
3. No further exchange communication is attempted
4. The summary is printed with the certification result

This prevents cascading failures from accumulating financial exposure in live-simulated environments.

## 25.9 Automatic Shutdown

When the suite finishes - all tests executed or fail-fast stopped - the runner invokes its completion callback once after printing the summary. `algoCore_c` then calls the algo framework's `requestTraderExit()`, which posts `EEV_GENERAL_EXIT_REQUEST` to the main manager. ttTrader runs the normal shutdown waterfall (algos stop, marketdata/order/exchange/websocket managers unwind, plugin modules unload) and the process exits with code `0` by itself. A certification runner only needs to wait for process exit and read the `RESULT:` marker.

## 25.10 Extensibility

### 25.10.1 Adding New Test Cases

1. Add a `CERTT_*` entry to `certTestType_e` enum in `algoCore.h` (before `CERTT_COUNT`)
2. Add a `certTestPhase_e` entry if new phase transitions are needed
3. Register the test case in `certTestRunner_c::init()` with name, description, and type
4. Add a case in `certTestRunner_c::_executeTest()` to call the implementation
5. Add cases in `onOrderCreated`, `onOrderCanceled`, `onOrderFailed`, `onOrderFill`, `onOrderUpdate` for response handling
6. Add a `_place*()` or `_*Test()` private method with the test logic

### 25.10.2 Attached/Contingent Orders

The framework supports contingent order chains through the OCO pair pattern. To implement triggered child orders:

1. Define a parent test case with `CERTP_WAIT_FILL` phase
2. In `onOrderFill`, when the parent fills, create a child order with a new user value
3. Register the child order's user value in the test descriptor's `ui32UserValue2`
4. Handle the child order's callbacks through the existing `CERTP_WAIT_SECOND_CREATE` / `CERTP_WAIT_SECOND_CANCEL` phases

## 25.11 Additional Edge Cases (Suggested for Production Readiness)

These scenarios are recommended for comprehensive exchange validation but require more complex setup:

1. **FOK/IOC TIF orders**: Verify Fill-or-Kill and Immediate-or-Cancel behavior. Requires orders near market price.
2. **Rapid create-cancel-create**: Stress test the exchange's order sequencing with sub-second order floods.
3. **Modify-then-immediately-cancel race**: Submit modify request followed immediately by cancel; verify no crash or stale state.
4. **Order at exact market price**: Boundary test for price validation.
5. **Maximum size order**: Test exchange limits by submitting maximum allowed order size.
6. **Reconnection during active order**: Drop and re-establish connection while an order is active; verify order state recovery.
7. **Concurrent orders on same instrument**: Place multiple simultaneous orders and verify independent life-cycle tracking.
8. **Leverage boundary orders**: Test orders exceeding available margin; verify margin rejection.
9. **Duplicate client order ID**: Attempt to reuse a client order ID; verify idempotency or proper rejection.
10. **Partial fill handling**: For exchanges supporting partial fills, verify remaining size tracking and fill notification accuracy.

11. **Exchange-side timeout/cancellation**: Let a GTD order expire naturally; verify timeout notification.
12. **Reduce-only order validation**: For exchanges supporting reduce-only flag, verify that it cannot open a new position.

# Chapter 26

# algoHERD — Herding-Exhaustion Reversion with Entropy-Gated Dislocation

`algoHERD` fades short bursts of uninformed, same-direction aggressive flow on CME micro/nano equity-index futures (IBKR L1: trades + BBO only, no L2 depth). The entry condition is the conjunction of three individually insufficient statistics: sign-sequence entropy collapse with small-print dominance (uninformed herding), ATR-normalized dislocation from volume-anchored value, and a Hawkes excitation decay from its trailing peak. An online logistic model learns the residual weight of those components per regime; a deterministic trigger gate plus the regime classifier bound the risk envelope.

Implementation-ready plan: `.kilo/algoPlans/algoHerd-herding-exhaustion-MuseSpark1.3.md`.

## 26.1 How It Works

1. **Tape + quote state** (`herdFeatures.h/.cpp`): the trade tape fills a 256-print ring with tick-rule signs (uptick/downtick/zero carry-forward) and O(1) per-side Hawkes intensities; the BBO stream fills spread / flicker Welford streams and a 512-sample print-size reservoir.
2. **Per closed 1m bar**: 1m indicators (`atr14`, `vwma20`, `linreg20`, `rvol20`) are read at the candle's own sequence; 5m indicators (`linreg20`, `volumeProfile60`, `atr14`) are read at the newest finalized 5m candle's sequence. The 10-feature vector is assembled; invalid reads invalidate the vector (fail closed).
3. **Training**: each feature snapshot is queued with its close, ATR and the toward-VWMA sign. The label matures `horizonBars` bars later: 1 when price touched the `labelGainAtr x ATR` reversion barrier within the horizon, otherwise 0. Exactly one bounded SGD step per closed bar.
4. **Regime**: CALM / NORMAL / HEATED from `0.5 x volPct + 0.3 x spreadZ01 + 0.2 x entropyBase`, with hysteresis bands and a 5-bar minimum dwell. HEATED blocks entries; NORMAL raises the entry threshold and widens stops.
5. **Entry gates** (in order): session -> trained -> regime -> vol-percentile -> deterministic trigger -> value-area position -> 5m trend filter -> model probability (threshold + regime offset) -> cooldown -> quote/spread -> no exposure. Direction always fades the dislocation.
6. **Protection**: market entry; native stop-market at `stopAtrMultiplier x ATR x regimeStop` and a take-profit close-limit at `tpAtrMultiplier x ATR` (partial leg). Once favorable excursion clears `0.8 x ATR`, the stop ratchets to entry +/- `0.2 x ATR` (breakeven-plus, modify-only). The runner exits on signal decay.
7. **Exits**: signal decay (`p < exitThreshold`), opposing herding trigger, max hold, stale-feed time stop (armed after 2 s, flatten after 30 s), and the daily session flatten.

## 26.2 Feature Table

| Idx | Feature | Formula | Lookback |
|---|---|---|---|
| 0 | `dislocAtr` | (close1m - vwma20) / atr14 | indicator |
| 1 | `entropyDeficit` | 1 - normalized Shannon entropy of the sign sequence | 64 ticks |
| 2 | `smallPrintShare` | share of prints <= session median size | 64 ticks / 512 reservoir |
| 3 | `exciteDecay` | (mu + lambdaDom) / (mu + trailing peak) | 256 ticks |
| 4 | `pressureImb` | tick-rule signed volume / total volume | 32 ticks |
| 5 | `microDisloc` | (last print - bbo mid) / atr14 | live |
| 6 | `spreadZ` | spread bps z-score (Welford) — gate only | live |
| 7 | `quoteFlickerZ` | BBO updates/second z-score (Welford) | 1 s buckets |
| 8 | `rvol1m` | native rvol20, clamped [0, 5] | indicator |
| 9 | `valuePos5m` | (close - VAL) / (VAH - VAL); degenerate: 0.5 + (close - POC)/(3 x ATR5m) — gate only | indicator |

Model inputs are slots 0-5, 7, 8 (spread z and value position are gates).

## 26.3 Configuration Parameters

| Parameter | Default | Range | Description |
|---|---|---|---|
| `instTradeId` | (required) | - | Instrument trade id |
| `ohlcId1m` / `ohlcId5m` | (required) | - | 1m signal / 5m filter feeds |
| `atr1mId` ... `atr5mId` | (required) | - | Indicator bindings (see config sample) |
| `risk.type` / `risk.value` | PERCENT / 0.20 | 0.05-1.0 | Per-trade risk budget (% of quote equity) |
| `sessionMode` | daily | daily only | continuous is forced back to daily |
| `flattenMinuteUtc` | 1245 | 0-1439 | Daily flatten minute (UTC) |
| `noEntryMinutes` | 30 | 0-1440 | Pre-flatten entry block |
| `entryThreshold` | 0.62 | 0.50-0.95 | P(revert) entry gate |
| `exitThreshold` | 0.40 | 0.05-0.50 | Signal-decay exit gate |
| `horizonBars` | 5 | 1-50 | Forward label horizon (1m bars) |
| `minTrainBars` | 200 | 1-1000 | Trained-label warm-up before entries |
| `learnRate` | 0.05 | 1e-5-1.0 | Online SGD step |
| `l2` | 0.0001 | 0-1.0 | L2 shrinkage |
| `performanceEmaAlpha` | 0.1 | 0.001-1.0 | Win-rate EMA smoothing |
| `labelGainAtr` | 0.30 | 0.05-5.0 | Reversion barrier in ATR units |
| `entropyMin` | 0.70 | 0.55-0.95 | Minimum entropy deficit |
| `exciteDecayMax` | 0.55 | 0.30-0.80 | Maximum Hawkes decay ratio |
| `dislocAtrMin` | 0.8 | 0.3-3.0 | Minimum ATR dislocation |
| `smallPrintMin` | 0.60 | 0.40-0.90 | Minimum small-print share |

| Parameter | Default | Range | Description |
|---|---|---|---|
| `stopAtrMultiplier` | 1.2 | >= 0.1 | Stop distance (ATR units) |
| `tpAtrMultiplier` | 1.0 | >= 0.1 | Take-profit distance (ATR units) |
| `maxEntrySpreadBps` | 8 | >= 0 | Entry spread filter |
| `bboStaleMsMax` | 2000 | 500-600000 | Entry freeze / time-stop arming |
| `bboStaleFlattenMs` | 30000 | 5000-3600000 | Armed stale-feed flatten delay |
| `maxVolPercentile` | 0.99 | 0-1.0 | ATR-bps percentile entry freeze |
| `cooldownMs` | 300000 | >= 1000 | Post-close re-entry cooldown |
| `stopCooldownMs` | 900000 | >= 1000 | Post-stop-out cooldown |
| `maxHoldMs` | 1200000 | >= 1000 | Hard hold bound (20 min) |
| `dailyMaxTrades` | 8 | 0-100000 | Daily trade cap (0 = unlimited) |
| `dailyMaxLossPct` | 0.02 | >= 0 | Daily loss latch (fraction of equity) |
| `hawkesAlpha` / `hawkesBeta` / `hawkesMu` | 0.8 / 0.05 / 0.05 | > 0 | Excitation rates (per tick / per second) |
| `regimeHeatedEnter` | 0.85 | 0.5-1.0 | HEATED entry score |
| `regimeHeatedLeave` | 0.70 | 0.4-enter | HEATED leave score |
| `regimeCalmEnter` | 0.25 | 0-0.5 | CALM entry score |
| `regimeCalmLeave` | 0.40 | enter-0.8 | CALM leave score |
| `regimeMinDwellBars` | 5 | 1-1000 | Minimum state dwell |
| `reduceOnlyClose` | true | bool | Reduce-only close-order guard |
| `stateFile` | `state/algoHERD-NES-state.json` | - | Framework-managed state path |

## 26.4 State Persistence

Model weights / Welford normalization / vol ring / performance EMA, the regime detector state, daily session counters and the cooldown end persist through the framework `saveStateJson`/`loadStateJson` (atomic `.tmp` + rename). Loads are validate-then-commit; a corrupt model node restarts fresh with entries locked until `minTrainBars` relearns. The tick ring and Hawkes excitation are rebuilt from the live tape after a restart (documented deviation from the plan, which listed them as persisted; the warm-up gates re-arm).

## 26.5 Sample Configuration

- `config/algoHERD-ibkr-nes-front.json` — IBKR CME E-nano S&P 500 front month (conid `908100745.CME.FUT`, NESU6), daily session, flatten 12:45 UTC, L1 only. The conid must be updated on every quarterly roll (see `config/instrumentsIbkr.cfg`, "FINDING CONIDS").

```
ttTrader.exe algos/algoHERD/config/algoHERD-ibkr-nes-front.json --log --logmarketdata
ttTrader.exe algos/algoHERD/config/algoHERD-ibkr-nes-front.json --log --tradingMode paper --logmark
```

## 26.6 Building and Testing

```
cmake --build .build/clang-debug --target algoHERD
.build/clang-debug/ttManagerOrderTests.exe HerdFeatures
.build/clang-debug/ttManagerOrderTests.exe HerdModel
.build/clang-debug/ttManagerOrderTests.exe HerdRegime
.build/clang-debug/ttManagerOrderTests.exe HerdSizer
.build/clang-debug/ttManagerOrderTests.exe HerdSession
.build/clang-debug/ttManagerOrderTests.exe HerdConfig
.build/clang-debug/ttManagerOrderTests.exe HerdOrderPath
```

## 26.7 Important Constraints

1. `decimal_t` for all finance-adjacent values; `double` only for Eigen, the Hawkes intensities and entropy/Z statistics (dimensionless counters/rates, documented HFT override).
2. Entries are MARKET orders by design (the exhaustion point decays with fill latency); no post-only path.
3. One instrument, one position, no stacking; reduce-only close orders.
4. Daily-only sessions; the flatten is enforced on bar close and on the 60 s heartbeat so a stalled feed still flattens.
5. A non-representable stop triggers an immediate flatten — a naked position is never held.

# Chapter 27

# algoHUB — Hub-Anchored Basis Reversion for CME Nano Futures

`algoHUB` trades the structural basis between the CME E-nano S&P 500 (NES, the satellite) and the deep E-mini S&P 500 (ES, the hub). The NES mid drifts away from the hub-implied fair value through stale quotes (hub-led) or uninformed nano flow (satellite-led); both deviations revert when quotes re-center. The strategy enters passively at the touch only in the loosely-coupled regime, conditioned on an online `pSnap` model, a two-class deviation gate, a cost floor and an online coupling regime (TIGHT/LOOSE/DECOUPLED). L1 only: NES trades + BBO, ES BBO.

Implementation-ready plan: `.kilo/algoPlans/algoHub-hub-anchored-basis-reversion-GLM-5.3.md`.

## 27.1 How It Works

1. **Feature tick (1 Hz, driven by the NES BBO)**: `featureBuilder.h/.cpp` maintains the basis EWMA anchor + streaming std, the hub per-second drift and vol EWMAs, the 1 s mid-increment coupling ring, the decayed NES tick-test flow, the trailing rel-max/half-life tracker and the latched 1m indicator context. Output: 12 scale-free features (see README table in `featureBuilder.h`).
2. **Coupling regime**: TIGHT (r >= 0.55, threshold +0.08, size x0.6), LOOSE (0.25 <= r < 0.55, defaults), DECOUPLED (r < 0.25, entries frozen). Bands require a 0.08 hysteresis margin plus `minSojournSamples`; an anchor drift above `anchorDriftFreezeTicks` in 5 minutes arms a 30 min DECOUPLED freeze and a re-anchor; a hub-vol sample above the 99th percentile freezes entries until it falls below the 95th.
3. **Model**: online logistic `pSnap = P(revert past the cost floor within`horizonMs`)`. Samples are captured only when `|relBasisZ| >= 1`; the label resolves when the mid moves `entryGainTicks` toward the anchor (1) or the stop distance away (0), or at the horizon (0). One bounded SGD step per resolution.
4. **Entry gates**: session -> cooldown -> regime freeze -> trained -> quote freshness/spread -> `|relBasisZ| >= minBasisZ` -> class gate (hub-led always; satellite-led needs hub vol below its rolling median and flow pushing away from value) -> cost floor in ticks -> `pSnap >= entryThreshold + offset` -> no exposure. Direction fades the basis (`-sign(rel)`).
5. **Execution**: passive limit at the touch (join bid/ask) with an `entryTimeoutMs` cancel window; stop-market at `max(stopAtrMult x ATR, minStopTicks x tick)`; take-profit close-limit at the hub-implied anchor distance.
6. **Exits**: signal decay (`pSnap` below `exitThreshold` or `relBasisZ` flip beyond 1.0), max hold, daily-loss flatten, session flatten, and a 60 s stale-satellite-feed flatten.

## 27.2 Configuration Parameters

| Parameter | Default | Range | Description |
|---|---|---|---|
| `instTradeId` | (required) | - | Satellite instrument id |
| `instRefId` | (required) | - | Hub reference instrument id |
| `ohlcId` | (required) | - | Satellite 1m signal feed |
| `atrIndicatorId` / `rvolIndicatorId` / `vpIndicatorId` | (required) | - | Indicator bindings |
| `risk.type` / `risk.value` | PERCENT / 0.10 | - | Per-trade risk budget (% of quote equity) |
| `sessionMode` | daily | daily only | Continuous is not supported by the design |
| `flattenMinuteUtc` | 1250 | 0-1439 | Daily flatten minute (UTC) |
| `noEntryMinutes` | 30 | 0-1440 | Pre-flatten entry block |
| `openQuietMinutes` | 2 | 0-240 | Post-22:00-UTC-reopen quiet window |
| `dailyMaxTrades` | 24 | 0-100000 | Daily trade cap (0 = unlimited) |
| `dailyMaxLossPct` | 0.015 | >= 0 | Daily loss latch (fraction of equity) |
| `lossStreakFreezeTrades` | 3 | 0-1000 | Consecutive losses arming the freeze |
| `lossStreakFreezeMs` | 600000 | >= 1000 | Freeze duration after the streak |
| `cooldownMs` | 5000 | >= 100 | Post-close re-entry cooldown |
| `featureTickMs` | 1000 | 100-60000 | Feature evaluation cadence |
| `horizonMs` | 60000 | 1000-3600000 | Label horizon |
| `entryGainTicks` | 3 | 1-1000 | Gain barrier for label 1 |
| `minTrainSamples` | 300 | 1-100000 | Resolved labels before entries |
| `warmupMs` | 600000 | - | Warning threshold for never-warm features |
| `entryThreshold` | 0.62 | 0.50-0.95 | pSnap entry gate |
| `exitThreshold` | 0.40 | 0.05-0.50 | pSnap decay exit |
| `minBasisZ` | 1.25 | 0.1-10 | Relative-basis z entry gate |
| `anchorTauMs` | 120000 | 1000-3600000 | Basis anchor EWMA tau |
| `maxHubAgeMs` | 2000 | 100-60000 | Hub quote freshness bound |
| `maxBboAgeMs` | 5000 | 500-600000 | Satellite entry quote freshness bound |
| `learnRate` / `l2` | 0.05 / 0.0001 | - | Online SGD step / L2 shrinkage |
| `performanceEmaAlpha` | 0.1 | 0.001-1.0 | Win-rate EMA smoothing |
| `regimeMinTrades` | 20 | 1-100000 | Trades before the win-rate multiplier activates |
| `regimeTightMultiplier` | 0.6 | 0-1 | TIGHT regime size multiplier |
| `confMinMultiplier` / `confMaxMultiplier` | 0.5 / 1.25 | - | Confidence envelope |
| `winRateMinMultiplier` / `winRateMaxMultiplier` | 0.5 / 1.25 | - | Win-rate envelope |
| `multiplierProductCap` | 1.5 | 0.1-10 | Cap on the multiplier product |
| `couplingTight` / `couplingLoose` | 0.55 / 0.25 | 0-1 | Coupling band edges |
| `couplingHysteresis` | 0.08 | 0-0.5 | Band crossing margin |
| `minSojournSamples` | 20 | 1-100000 | Samples before a state transition |
| `anchorDriftFreezeTicks` | 20 | >= 1 | 5-minute anchor drift freeze trigger |
| `decoupleFreezeMs` | 1800000 | >= 1000 | Forced DECOUPLED duration |
| `volFreezePercentile` | 0.99 | 0.5-1 | Hub-vol freeze latch |
| `stopAtrMultiplier` / `minStopTicks` | 1.5 / 4 | - | Stop distance geometry |

| Parameter | Default | Range | Description |
|---|---|---|---|
| `maxHoldMs` | 240000 | >= 1000 | Hard hold bound |
| `entryTimeoutMs` | 10000 | 500-600000 | Passive-entry cancel window |
| `feeRoundTurnUsd` | 0.75 | > 0 | Round-turn commission for the cost floor |
| `minEdgeTicks` | 2 | 0-1000 | Minimum expected edge |
| `maxSpreadTicks` | 4 | >= 1 | Satellite spread entry filter |
| `maxPositionContracts` | 12 | >= 1 | Contract cap |
| `maxNotionalUsd` | 200000 | >= 0 | Notional cap |
| `reduceOnlyClose` | true | bool | Reduce-only close-order guard |
| `stateFile` | `state/algoHUB-NES-state.json` | - | Framework-managed state path |

## 27.3 State Persistence

Model weights / Welford normalization / win-rate EMA, the coupling regime (with its vol ring and drift freeze), daily session counters / loss streak, builder level statistics (anchor, basis std, flow decay, half-life) and the cooldown persist through `saveStateJson`/`loadStateJson` (atomic write, validate-then-commit loads). Raw per-second rings rebuild live after a restart; the warm-up gates re-arm.

## 27.4 Sample Configuration

- `config/algoHUB-ibkr-nes-front.json` — NESU6 (`908100745.CME.FUT`) as the satellite and ES Sep 2026 (`496734177.CME.FUT`) as the hub, daily session, flatten 12:50 UTC. Update BOTH conids together on every quarterly roll (see `config/instrumentsIbkr.cfg`, "FINDING CONIDS").

```
ttTrader.exe algos/algoHUB/config/algoHUB-ibkr-nes-front.json --log --logmarketdata
ttTrader.exe algos/algoHUB/config/algoHUB-ibkr-nes-front.json --log --tradingMode paper --logmarket
```

## 27.5 Building and Testing

```
cmake --build .build/clang-debug --target algoHUB
.build/clang-debug/ttManagerOrderTests.exe HubFeatures
.build/clang-debug/ttManagerOrderTests.exe HubModel
.build/clang-debug/ttManagerOrderTests.exe HubRegime
.build/clang-debug/ttManagerOrderTests.exe HubSizer
.build/clang-debug/ttManagerOrderTests.exe HubSession
.build/clang-debug/ttManagerOrderTests.exe HubOrderPath
.build/clang-debug/ttManagerOrderTests.exe HubConfig
```

## 27.6 Important Constraints

1. `decimal_t` for all finance-adjacent values; `double` only for Eigen, EWMAs and Pearson normalization (dimensionless statistics, documented override).
2. L1 only: no order-book depth and no BookTrader entitlement dependency.
3. Entries are passive (maker) by design; the fill-miss distribution is part of the edge (missed entries are adverse-selection winners) and is logged.
4. One position, no stacking, reduce-only close orders.
5. The anchor is never trusted in DECOUPLED: the error mode is not trading, not trading wrong.

# Chapter 28

# algoHUNT — Liquidity-Hunt Reversion

Intraday perp/futures algo for ttTrader. Fades **stop-hunts**: a structural liquidity pool (prior-day high/low, day extreme, equal swing highs/lows) gets swept by forced flow, and price reclaims the pool within a short window — the sweep was liquidity consumption, not information. The algo fades the sweep back toward value with an asymmetric bracket. A **realtime neural meta-controller** (fully in-process, online-trained — no offline training, no external tooling) gates entries and continuously re-tunes the algo's own parameters from live trade outcomes.

Distinct from the other algos in this repo (`algoMLD`, `algoMCT`): the trigger is a discrete structural event (sweep + reclaim + absorption), not a continuous signal; targets are structural (opposing pool / R band), not pure indicator multiples.

Status: **implemented, not yet validated for live sizing**. Builds clean in the clang-debug preset (the release preset adds `-Werror`); all 31 `ttManagerOrderTests` suites pass with the algoHUNT coverage folded into `HuntPool`, `HuntNn`, `HuntCf` and `HuntSession` (feature builder, regime detector, entry-size envelope, TP2 anchor band and quote-staleness checks included). Open gates before live sizing: replay lifecycle test, venue certification re-run (no skips), walk-forward evaluation with the TP2-hit-rate kill metric, then paper. Design + expectancy math: `.kilo/plans/1788752462168-algoHUNT.md` (as-built rev 3).

## 28.1 Signal pipeline (per closed 5m bar)

## 28.2 The realtime meta-controller

One small network (14 -> 24 tanh -> two heads), trained only online inside the dll — no offline training, no external tooling:

| Head | Output | Training signal | Role |
|---|---|---|---|
| gate | pSweep in [0,1] | binary cross-entropy on matured episode labels from the pending ring (reverted >= 1R before extending 0.5R -> 1; extension -> 0; 36-bar timeout settles at half-R retrace). labels accumulate for *all* episodes, including gated-out ones (would-have-been supervision) | entry gate `pSweep >= threshold` (controller adjusts ±12%) |
| controller | 4 raw outputs -> multiplicative deltas | bounded REINFORCE from realized trade R with an EWMA baseline; exploration via a seeded xorshift stream during the learning phase, decaying linearly to a deterministic policy | live-adjusts `tp1R` (±35%), `trailAtr` (±40%), size (±25%), `entryThreshold` (±12%) |

Adaptation guardrails: **seeded random init** (a zero-initialized network has zero hidden activations and could never learn — fixed seed keeps runs and replays deterministic), Welford feature standardization, L2 + per-weight clamp ±8, controller deltas hard-clamped in the core, neutral deltas until `exploreTrades` closed trades, rolling-

20-trade win-rate watchdog suspends the controller below `ctrlSuspendWinRate` — **and gates the memory feed-back**: when the watchdog latches, both adaptive sources revert to config base (the gate head and the memory keep learning throughout). Deterministic end to end: seeded RNG, event-ordered updates -> bit-identical `.ttpb` replays. State persists atomically to `stateFile` (ieee-754 bit-exact weights, validate-then-commit restore) on clean exit and every closed trade, plus a periodic checkpoint (first heartbeat, then every 5 min) so warm-up progress, the pending episode ring, the daily session counters (trade count, loss latch, session index) and the re-entry cooldown survive a kill or crash.

Entry size is a **per-trade stack** — network confidence × controller delta × regime size — applied through one pure envelope helper and clamped to `[0.1, 1.5]`. The previous trade's multiplier is not an input: identical inputs always produce the identical size, so size cannot drift or compound across trades. QUIET scales to 0.6×; STRESSED returns 0× and blocks entries upstream.

## 28.3 Counterfactual parameter memory

On top of the controller head, every closed trade gets a **direct counterfactual answer** instead of a gradient nudge:

1. **Record** the trade's price path (per-bar high/low/close/ATR, up to 40 bars) and its condition context (entry features, pool type, regime).

2. **Re-simulate** the exit ladder over that path for a grid of candidate values — tp1R′ in [0.6, 2.0] step 0.1, trailAtr′ in [0.4, 1.5] step 0.1 (~=180 ladder replays, pessimistic same-bar ordering: an adverse touch resolves before a favorable one). The **argmax per parameter** is the "value that would have maximized this trade".
3. **Store** the optima in the market-condition bucket the trade happened in: a coarse deterministic key over sweep depth, volume z, vol level, utc-phase quadrant, pool type and regime (972 buckets). Each bucket keeps a freshness ring of the last 40 samples with per-parameter optimum histograms.
4. **Feed back when statistically relevant**: a bucket drives a parameter once it holds >= `cfMinBucketSamples` samples and the modal optimum bin holds >= `cfModalShare` of them; the fed-back value is the modal bin center (envelope-clamped, ratchet-consistent). Until then the config-base × controller-delta value is used. Ties in the cutoff search resolve to the more selective threshold.

What the memory optimizes per mechanism:

| Parameter | Counterfactual | Feed-back |
|---|---|---|
| `tp1R` | per-trade ladder re-simulation | modal bin center of the bucket |
| `trailAtr` | per-trade ladder re-simulation | modal bin center of the bucket |
| `entryThreshold` | per-bucket argmax of summed realized R over (pSweep, R) pairs | the winning pSweep cutoff |
| size | degenerate per-trade (linear scaling) — not counterfactual | stays with the sizing stack + watchdog |
| tp2 distance | structural anchor (opposing pool) — deliberately not a free parameter | untouched |

The freshness ring keeps the memory adaptive: stale conditions lose samples and relevance automatically, so the algo keeps fine-tuning as the market changes. The memory persists in the state file (`memory` section, sparse — only touched buckets) and survives restarts; the controller head keeps training regardless, so it stays warm as the fallback wherever a bucket has not (yet) reached relevance.

## 28.4 TP2 anchoring

TP2 is the structural opposing-liquidity target, not a free parameter: at entry the nearest unswept pool on the opposite side (`huntPoolTracker_c::nearestOpposing`) is captured as an absolute price and stored for the trade. The target distance is `|anchor - fill|` clamped to `[minTp2R, maxTp2R] × R`; without a usable anchor (no pool, or a pool on the wrong side of the fill) the distance falls back to 2R and is then clamped to the same band. The captured anchor is reused when a pyramid resizes the leg — later pool updates never move an open trade's target.

## 28.5 Sweep / reclaim semantics

- Sweep: bar extreme penetrates the pool by >= `minSweepBps` × close. The sweep-bar statistics (volume z vs the prior 20 bars, opposite-aggressor share per side, range/ATR) are frozen into the pool at sweep time.
- Reclaim: a bar *closes* back inside the pool within `maxReclaimBars` bars — reclaim lag counts bars **after** the sweep bar (1 = next bar, 0 = reclaimed on the sweep bar itself). The expiry check runs before the reclaim check, so no late reclaims slip through; an expired sweep (continuation, not a hunt) sends the pool to cooldown.
- Absorption evidence is mandatory for any action: sweep-bar volume z >= `minSweepVolZ`, opposite-aggressor share >= `deltaOpposeMin` (sell share for swept highs, buy share for swept lows), penetration <= `maxPenetrationAtr` × ATR.
- A deeper sweep within the reclaim window extends the frozen extreme, so the stop always anchors on the true swing high/low.

## 28.6 Pyramiding (profit direction only)

| # | Condition | Default | Rationale |
|---|---|---|---|
| 1 | tp1 filled (breakeven armed) | — | adds only into a *confirmed* winner |
| 2 | fresh same-direction episode, `pSweep >= pyramidThreshold` | 0.70 | the network re-rates the structure |
| 3 | unrealized pnl bps >= `pyramidMinPnlBps` × (level+1) | 8 × tier | the market must have earned the add-on |
| 4 | interval since last add >= `pyramidIntervalMs` | 300000 | no stacking on one spike |
| 5 | `pyramidCount < pyramidMaxCount` | 2 | max 1.75× base size (scale 0.5) |

Each add is `base × scale^level`; risk capital accrues at the new average entry and the tp2 leg is resized to the new total. The stop is **never widened** by a pyramid.

## 28.7 Parameters (defaults; clamped to these ranges at parse)

| Key | Default | Range | Purpose |
|---|---|---|---|
| `instTradeId` / `ohlcId` / `atrIndicatorId` | — | — | instrument + 5m feed + atr binding |
| `ohlcTimeFrameMin` | 5 | 1–1440 | bar cadence |
| `eqClusterBps` | 3 | 0.1–20 | equal highs/lows cluster tolerance |
| `minSweepBps` | 5 | 1–100 | minimum penetration beyond the pool |
| `maxReclaimBars` | 2 | 1–5 | bars allowed to close back inside |
| `maxPenetrationAtr` | 0.6 | 0.1–3 | deeper sweeps are breakouts, not hunts |
| `minSweepVolZ` | 1.5 | 0–10 | sweep-bar volume z-score floor |
| `deltaOpposeMin` | 0.35 | 0–1 | opposite-aggressor share floor (absorption) |
| `entryThreshold` | 0.60 | 0.55–0.80 | base pSweep gate (controller adjusts ±12%) |
| `minTrainEpisodes` | 40 | 10–2000 | gate dormant below this many labels |
| `learnRate` / `ctrlLearnRate` / `l2` | 0.03 / 0.01 / 0.0001 | 1e-5–1 | network steps |
| `exploreTrades` | 100 | 10–100000 | exploration phase length |
| `tp1R` / `tp1Frac` | 1.0 / 0.5 | 0.5–2 / 0.1–0.8 | first target + size fraction |
| `tp2Frac` | 0.3 | 0–0.8 | opposing-pool target fraction |
| `minTp2R` / `maxTp2R` | 1.5 / 3.0 | 1–8 | tp2 distance band (fallback 2R) |
| `trailAtr` | 0.8 | 0.1–3 | runner trail distance |

| Key | Default | Range | Purpose |
|---|---|---|---|
| `stopBufferAtr` | 0.3 | 0–2 | stop sits beyond the sweep extreme + buffer |
| `breakevenBufferTicks` | 1 | 0–100 | fee-aware breakeven offset |
| `pyramidMaxCount` | 2 | 0–5 | add-ons per position |
| `pyramidScale` | 0.5 | 0.1–1 | diminishing add size (base × scale^level) |
| `pyramidMinPnlBps` | 8 | >=0 | tiered profit cushion (× level) |
| `pyramidIntervalMs` | 300000 | >=1000 | add-on throttle |
| `pyramidThreshold` | 0.70 | 0.5–0.95 | pSweep floor for add-ons |
| `maxEntrySpreadBps` | 8 | >=0 | spread ceiling |
| `maxVolPercentile` / `quietVolPercentile` | 0.97 / 0.30 | 0–1 | stress tail / quiet regime bounds |
| `cooldownMs` / `maxHoldMs` | 600000 / 10800000 | >=1000 | re-entry + time stop (3 h) |
| `ctrlSuspendWinRate` | 0.45 | 0.1–0.9 | controller watchdog trigger |
| `cfMemoryEnabled` | true | — | counterfactual parameter memory on/off |
| `cfMinBucketSamples` | 30 | 10–40 | samples before a bucket feeds back (bounded by the 40-sample freshness ring) |
| `cfModalShare` | 0.40 | 0.3–0.9 | modal-bin share required for relevance |
| `sessionMode` | daily | daily/continuous | per venue (continuous for 24/7 perps) |
| `flattenMinuteUtc` / `noEntryMinutes` | 1320 / 30 | 0–1439 / 0–1440 | daily flatten window (ibkr: no overnight) |
| `dailyMaxTrades` / `dailyMaxLossPct` | 12 / 0.02 | >=0 | daily limits; the loss value is a fraction of quote-currency equity (0.02 = 2%) |
| `reduceOnlyClose` | true | bool | venue reduce-only on every close order; clear only for multi-algo netting on one account |
| `fundingMinutesUtc[]` / `fundingSkirtMinutes` | [] / 5 | <=8 entries / 0–60 | funding no-entry skirts |
| `stateFile` | state/algoHUNT-state.json | — | atomic network/memory persistence (framework-managed; a relative path resolves against the process working directory) |

`risk.type` / `risk.value` are consumed by the shared framework risk calculator (position sizing derives the loss budget from entry-to-stop distance).

## 28.8 Exits (priority order)

1. structural stop (ratchet-only: breakeven after tp1, then atr trail; a lost stop is re-placed at the ratcheted trigger, never the original)
2. opposite-direction episode (thesis flipped)
3. max hold (3 h default)
4. session flatten (timer-enforced even without bars; daily mode = no overnight on IBKR)
5. tp1 / tp2 limit legs fill independently; a 1 s feed watch compares the local wall clock against the latest venue bbo timestamp — > 30 s with an open position force-flats, > 30 s flat blocks new entries (an empty bbo ring is "no market yet" and changes nothing)
6. whenever the position closes (stop, tp, time exit, flatten) every still-working sibling close order is canceled **before** the machine returns to IDLE, and a lost protection leg is re-placed from the current position size — stop at the ratcheted trigger, both TP legs, a filled tp1 is skipped. This matters when `reduceOnlyClose` is off (multi-algo netting): an orphan close leg could otherwise open a reverse position.

## 28.9 Implementation

| Component | File | Role |
|---|---|---|
| pool registry + sweep/reclaim machine | `poolTracker.h/.cpp` | fixed levels (prior-day, session) before swing pools; equal H/L merge by touch count; 48-bar per-pool dormancy |
| features + absorption gates | `featureBuilder.h/.cpp` | 14 scale-free features; precomputed 15-min utc phase table |
| realtime meta-controller | `nnGate.h/.cpp` | trunk + gate head + controller head; online training; atomic state io |
| counterfactual parameter memory | `counterfactual.h/.cpp` | ladder re-simulation grid, condition-bucket memory, statistically-relevant feed-back |
| regime | `regimeDetector.h/.cpp` | vol-percentile ring, spread/chop EWMA, hysteresis 3 in / 6 out, STRESSED immediate |
| session | `sessionGuard.h/.cpp` | daily/continuous, daily limits, funding skirts (circular across midnight) |
| orchestration | `algoCore.h/.cpp` | state machine, ladder + ratchet, pyramiding, per-trade size envelope, entry-anchored TP2, feed-watch kill switch, orphan-close cleanup, order-loss recovery, persistence |
| decision math | `algoMath.h` | pure helpers: entry size envelope `[0.1, 1.5]` (no compounding), TP2 anchor band + 2R fallback, quote staleness |
| run config | `config/algoHUNT-lighter-btc-perp.json` | Lighter BTC/USDC perp, continuous session, funding skirts, reduce-only closes |
| unit tests | `tests/managerOrderTests/algos/testPoolAlgoMm.cpp` | `HuntCf`, `HuntSession` — deterministic, no exchange |

Deployed dlls: `W:\ttTraderDebug\algos\algoHUNT.dll` and `W:\ttTrader\algos\algoHUNT.dll` (post-build copy).

## 28.10 Validation status

All 31 `ttManagerOrderTests` suites pass (0 failures). algoHUNT coverage lives in `tests/managerOrderTests/algos/testHu` registered as `HuntPool`, `HuntNn`, `HuntCf` and `HuntSession`; the feature-builder, regime-detector, size-envelope, TP2-anchor and quote-staleness checks run inside those registered suites (the suite table in `tests/managerOrderTests/main.cpp` was out of scope, so no new suite names were added). `algoConfig.cpp` is not linked into the test target, so the parameter parse/clamp path is not unit-tested here — config validation stays a run-level check against `config/algoHUNT-lighter-btc-perp.json` and the clamped ranges in the parameter table above. Remaining before live sizing: static replay lifecycle check, venue certification re-run (no skips), walk-forward evaluation with the TP2-hit-rate kill metric (>= 20%), then >= 4 weeks paper with >= 150 closed trades and net WR >= 52% / PF >= 1.3. Full gates: `.kilo/plans/1788752462168-algoHUNT.md` §8.

# Chapter 29

# algoHarvest — level-reaction harvester

Implements the plan at `.kilo/algoPlans/sascha/algoHarvest.md` (v1.0) as a certifiable algo plugin. algoHarvest trades mean-reverting reactions at confirmed 5m fractal levels on closed 1m candles, gated by a measured session window or a direction-locked momentum slide. Multiple instruments (BTC + ETH) run as independent state machines inside one algo instance; each may hold one position at a time.

## 29.1 Files

| File | Content |
|---|---|
| `algoCore.h/.cpp` | State machine, gates, order plumbing, kill switches |
| `harvestFeatureEngine.h/.cpp` | Deterministic §3 features (fractals, atr5, med_range15, net, vol_ratio, pulse, session) |
| `harvestLevelTracker.h` | level_set bookkeeping (merge, burn, age-out, touch) |
| `algoConfig.h/.cpp` | Parameter set with documented clamping |
| `config/algoHarvest-lighter-btc-eth.json` | Run config: Lighter BTC + ETH, 1m/5m trade-built candles, funding feed |

Run: `ttTrader.exe algos/algoHarvest/config/algoHarvest-lighter-btc-eth.json --log --logmarketdata`

## 29.2 Features (plan §3)

All series run on closed candles (1m and 5m, built from exchange trades with `gapFillEmpty` so the 1m ring stays contiguous).

- **frac_high / frac_low** — strict 4-neighbor 5m fractal on the bar 2 back (plan bar [3]); each bar is evaluated exactly once, 2 bars after its close.
- **level_set** — merged (within `levelMergeBps`, same direction -> mean) fractal levels of the last 288 5m bars (24h). Levels age out, burn when price closes beyond them by `levelBurnBps`, and burn after triggering one entry per direction.
- **atr5** — Wilder ATR(14) on 5m, seeded with the mean of the first 14 true ranges; gaps re-seed to stay deterministic.
- **ema_drift** — incremental EMA(close, 60) on 1m with k = 2/61.
- **med_range15** — median over the last 32 non-overlapping 15-candle blocks of (max close - min close) / close[1] × 10000 bps.

- **net15 / net60** — (close[1]/close[16] - 1) × 10000 and the 60-bar equivalent.
- **vol_ratio** — vol[1] / median(vol[2..61]).
- **pulse** — median(range_bps[1..15]) / median(range_bps[1..480]); requires the full 480-bar lookback (ring capacity 480 per instrument).
- **session_ok** — UTC Mon–Fri, `sessionStartMinuteUtc` … `sessionEndMinuteUtc` (default 13:00–21:00, `[start, end)`).
- **slide_dir** — +1 when net60 >= `slideMin` × med_range15, -1 mirrored, else 0 (dead regime).

## 29.3 Entry (plan §4)

GATE = (session_ok OR slide_dir != 0) AND pulse >= `pulseMin` AND vol_ratio >= `volMin`.

LONG: a frac_low level touched on 1m candles [1] or [2], close[1] > level, close[1] > close[2], and (close[1] >= ema_drift OR slide_dir = +1), with slide_dir != -1. SHORT mirrored. Simultaneous LONG + SHORT signals on one candle take no trade.

Funding filter: the paying side is blocked when the next funding rate (8h normalized) exceeds `fundingMax`, and no paying-side entries within `fundingSkirtSeconds` of the settlement. Fail-open when the venue has not published funding data yet. Requires `"funding": true` on the instrument.

Sizing (plan §7): risk = `riskPct` × quote-asset equity; stop distance = max(`atrStopX` × atr5, `breathX` × med_range15) in bps; size = risk / stop distance, capped by `levCap` × equity / planned positions and `maxNotional`, floored to the size grid, skipped below `minNotional`.

The entry is a post-only LIMIT at the touch-side best quote with a fresh framework order id per (level, direction). On a post-only rejection it is resent once as a plain LIMIT at the same price; a second loss returns the machine to FLAT (the level stays burned). Unfilled remainders cancel after `entryTimeoutS`; partial fills adopt the position.

## 29.4 Exits (plan §4/§8, precedence STOP > BREATH > RATCHET > TIME)

- **STOP** — logical stop at the plan stop distance from the average entry, never widened by adds; detected intra-candle (1m low/high) and on every BBO tick crossing; executed as a reduce-only MARKET.
- **BREATH-CUT** — pnl_bps <= -`breathX` × med_range15 (pnl_bps is unrealized, in bps of entry notional, after entry-side maker fee + `slippageBudgetBps`).
- **RATCHET** — once the peak reaches tier 1/2/3 (`ratchT1Bps..ratchT3Bps`), exit when pnl falls back below the tier's locked fraction of the peak (`ratchKeep1..ratchKeep3`, defaults 0.5/0.6/0.75).
- **TIME-EXIT** — age >= `maxHoldS` while the peak never reached tier 1 and no add was taken.
- **KILL** (global) — daily loss <= -`dailyKillPct` of day-start equity, equity <= (1 - `ddKillPct`) × equity high-water mark, no market update for `gapHaltS`, or 5 consecutive order API errors: close everything, latch `HALTED`. `HALTED` never auto-resets (00:00 UTC re-anchors only the daily equity); clearing it requires an algo restart.

## 29.5 Pyramiding (plan §4/§7)

At most `pyrMax` adds, each doubling the current position (clipped to the total notional cap; a clipped remainder below 10% of the current size skips). Conditions per 1m close: pnl >= 0.5 × med_range15, drawdown since the previous add >= -`pyrTolBps`, position age >= `pyrMinAgeS` before the first add, pulse gate still open. Adds are MARKET orders; a rejected add is skipped without retry. The stop never widens.

## 29.6 Precedence and plumbing

Evaluation order per closed 1m candle: KILL > STOP > BREATH-CUT > RATCHET > TIME-EXIT > Pyramid > Entry. Exit closes retry every `exitRetryIntervalS` up to `exitRetryCount` times, then `API_HALT`. Restart reconcile: a venue position found on the first candle is adopted (stop re-derived from the plan formula at the adopted average price).

## 29.7 Parameters

| Key | Default | Range | Meaning |
|---|---|---|---|
| `levelMergeBps` | 3.0 | 1..10 | merge distance for same-direction levels |
| `levelBurnBps` | 10.0 | 5..50 | close-beyond distance that burns a level |
| `levelMaxCount` | 64 | 16..256 | level set capacity |
| `pulseMin` | 1.2 | 1.0..3.0 | min 15m/8h range-energy ratio |
| `volMin` | 1.0 | 0.5..3.0 | min last-bar volume vs 1h median |
| `slideMin` | 5.0 | 3..10 | net60 threshold in med_range15 multiples |
| `sessionStartMinuteUtc` / `sessionEndMinuteUtc` | 780 / 1260 | 0..1440 | session window (13:00–21:00 UTC) |
| `sessionEnabled` | true | — | false = always-on (testing only) |
| `fundingFilterEnabled` | true | — | master switch for the funding entry filter |
| `fundingMax` | 3.0 (0.03%/8h) | 0..10 | max tolerated rate for the paying side |
| `fundingSkirtSeconds` | 60 | 0..600 | no paying-side entries this close to settlement |
| `atrStopX` | 1.0 | 0.5..2 | stop distance in atr5 multiples |
| `breathX` | 1.0 | 0.5..2 | breath-cut in med_range15 multiples |
| `ratchT1Bps` / `ratchT2Bps` / `ratchT3Bps` | 5 / 10 / 20 | 3..60 | ratchet tier triggers (bps) |
| `ratchKeep1` / `ratchKeep2` / `ratchKeep3` | 0.5 / 0.6 / 0.75 | 0.3..0.95 | locked fraction of peak per tier |
| `maxHoldS` | 3600 | 600..14400 | time exit for non-ratcheted, non-added positions |
| `pyrMax` | 5 | 0..7 | max adds per position |
| `pyrTolBps` | 5.0 | 0..20 | tolerated drawdown since the previous add |
| `pyrMinAgeS` | 60 | 15..600 | min position age before the first add |
| `riskPct` | 0.01 | 0.001..0.02 | risk fraction of equity per entry |
| `levCap` | 3.0 | 1..5 | max total notional in × equity |
| `maxNotional` / `minNotional` | 30000 / 100 | — | absolute per-position notional bounds |
| `slippageBudgetBps` | 2.0 | 0..10 | per-side slippage budget (plan §2: 0.02%) |

| Key | Default | Range | Meaning |
|---|---|---|---|
| entryTimeoutS | 20 | 5..120 | cancel unfilled entry remainder |
| exitRetryCount / exitRetryIntervalS | 5 / 1 | — | close-order retry ladder |
| dailyKillPct | 0.03 | 0.01..0.05 | daily loss kill (fraction of day-start equity) |
| ddKillPct | 0.10 | 0.05..0.2 | drawdown kill from equity HWM |
| gapHaltS | 120 | 30..600 | market-update silence that trips DATA_GAP |

All numeric keys clamp into these ranges at parse time: a malformed config degrades to the nearest safe value instead of disabling a risk control.

## 29.8 Tests

tests/managerOrderTests/testHarvestAlgo.cpp (suites HarvestRing, HarvestFractals, HarvestAtr, HarvestFeatures, HarvestSession, HarvestLevels, HarvestConfig, HarvestGateMath) covers the ring continuity/wraparound semantics, strict fractal geometry, the Wilder seed and recursion, hand-computed medians/nets/ratios, UTC session boundaries, level merge/burn/age/touch rules and the config clamping.

Certification: the Marketdata Funding test in algoExchangeTest verifies the funding subscription binding per venue without waiting for a settlement frame (skips on venues that declare no funding capability).

# Chapter 30

# Indicator Certification Algo

`algoIndicatorTest` is the runtime integration certification layer for Indicator V2. It complements the deterministic `IndicatorsV2` unit suite; it does not replace formula-vector tests.

For one configured indicator, the algo validates:

- instrument and OHLC subscription binding
- indicator ID and expected type
- descriptor and output schema
- historical warm-up: every finalized candle in the warmed ring is checked against the indicator's published samples (newest-to-oldest, stopping at the indicator's sample/warm-up boundary)
- monotonic finalized-candle delivery
- warm-up followed by at least one valid output
- exact-sequence lookup
- watchdog timeout behavior

Phases and runtime: the historical warm-up validates within seconds of the `HISTORICAL_OK` event (progress logged every 32 candles); the live phase then requires `minimumCandles` live candle closes, logged per close. The default and all runtime configs require 3 new live candles; on 1-minute candles the runtime configs target roughly 3-4 minutes total (`timeoutMs: 300000`).

The final log contains one machine-readable result:

```
INDICATOR CERTIFICATION SUITE SUMMARY
RESULT: CERTIFIED
```

or:

```
RESULT: NOT CERTIFIED - <reason>
```

Debug builds additionally report `steady_clock` read/callback timing:

```
DEBUG INDICATOR READ TIMING ns: min=... avg=... max=... samples=...
```

Timing is diagnostic and does not determine functional certification. Performance gates should use repeated release-build benchmarks on controlled hardware.

The runtime algo is a secondary end-to-end smoke test. The deterministic `IndicatorsV2` suite is the certification authority for exact formulas and DLL lifecycle because it uses static candle vectors and returns a conventional process failure code.

Each indicator directory contains an `algoIndicatorTest.cfg` configured for that plugin. These runtime smoke configs use Binance public market data and simulation mode, so they require network access but no trading credentials. Deterministic formula and DLL ABI certification remains available locally through:

```
.build/clang-debug/ttManagerOrderTests.exe IndicatorsV2
```

The preferred end-to-end certification uses `exchgStaticIndicator` and one-second candles. Every indicator directory contains its own `algoIndicatorTest-static.cfg` alongside `algoIndicatorTest.cfg`:

```
W:\ttTraderDebug\ttTrader.exe `
  W:\development\ttTrader\indicators\indIchimoku\algoIndicatorTest-static.cfg `
  --tradingMode simulation `
  --log
```

The static exchange provides fixed historical and live prices, requires no network or credentials, and normally completes three live-candle checks in approximately four seconds.

Build:

```
cmake --build .build/clang-debug --target algoIndicatorTest
```

Run an indicator certification from the deployment directory, for example:

```
W:\ttTraderDebug\ttTrader.exe `
  W:\development\ttTrader\indicators\indATR\algoIndicatorTest.cfg `
  --tradingMode simulation `
  --log
```

Do not pass `--logmarketdata` for these runtime certifications: the debug build's per-frame marketdata logging caps the event pipeline below the live BTCUSDT frame rate, the log falls minutes behind exchange time, candles arrive late, and the suite times out (`NOT CERTIFIED - certification timeout before valid output`). Without the flag, candle latency is ~1-6 s.

After the `RESULT:` marker, the algo calls the algo framework's `requestTraderExit()`, which posts `EEV_GENERAL_EXIT_REQUEST` to the main manager. The process then runs the complete exit waterfall - algos stop, marketdata/order/exchange/websocket managers unwind, indicator instances are destroyed through their owning DLLs, modules unload with zero live instances, and the asio context joins - and exits with code 0 on its own. A certification runner only needs to wait for process exit; `Stop-Process` is a fallback, not part of the flow. `CTRL_BREAK/CTRL_C` remains the manual abort for pre-completion interruption.

Debug builds route CRT assert reports to stderr instead of modal dialogs (`_CrtSetReportMode` in `main`), so a failing invariant can never stall an asio worker or block a headless certification run; the assert text lands in the captured stderr with module/file/line context.

# Chapter 31

# algoMCT — Microstructural Coherence Trading

Quantum-inspired density matrix approach to order flow analysis. BBO-driven, purity-signaled entries with pyramiding and coherence-weighted take profit.

---

## 31.1 Overview

**algoMCT** treats the two-state microstructural regime (bullish / bearish pressure) as a quantum system. A 2×2 density matrix $\rho$ is constructed from BBO updates over a sliding time window. The purity $\gamma = \text{Tr}(\rho^2)$ measures how *coherent* (informed, directional) the order flow is. When $\gamma$ crosses above an entry threshold, the market has "collapsed into a decision" — we trade in the direction of the dominant eigenvector.

| Property | Value |
|---|---|
| Primary data source | BBO (`onMarketdataBbo`) — universal across exchanges |
| Trade data | v2 enrichment only; synthetic/generated trades filtered out |
| Instrument type | Perpetual futures (long/short) |
| Holding period | Seconds to minutes (microstructural regime) |
| Computation | Closed-form 2×2 algebra, pure `decimal_t` (no Eigen needed) |

### 31.1.1 Implementation Status

| Feature | Status | Notes |
|---|---|---|
| BBO density-matrix signal ($r_z$, $r_x$, purity, entropy) | Implemented | Time-window scan over instrument BBO ring buffer |
| Entry/exit state machine | Implemented | IDLE / entering / in-position / exiting states |
| Pyramiding guards | Implemented | Purity ratchet, alignment, `rxTrend`, PnL bps gate, max size cap, interval |
| Coherence-weighted TP layers | Implemented | Per-fill TP limit placement with layer tracking |

| Feature | Status | Notes |
|---|---|---|
| TP cancellation on signal exit | Implemented | Cancels tracked TP layers before signal-driven close |
| `tpGraceMs` limit-then-market fallback | Implemented | Profitable purity-collapse exit: limit at mid then timed market fallback |

---

## 31.2 Why BBO-Primary?

| Exchange type | Trade frequency | BBO frequency | Examples |
|---|---|---|---|
| High-activity CEX | 10–500/sec | 100–5000/sec | Binance, Bybit, Okx |
| Mid-activity CEX | 0.1–5/sec | 5–100/sec | Bitfinex, Kraken |
| Low-activity / DEX-relay | 0.01–0.1/sec | 1–20/sec | Poloniex, Aster, Lighter, Paradex, Hyperliquid |

On low-activity exchanges you could wait 30+ seconds for a single trade. A trade-only algo would freeze. BBO updates are continuous — market makers constantly re-quote. The physics analogy: trades are strong (projective) measurements; BBO updates are weak (continuous) measurements. Continuous weak measurement provides more information.

### 31.2.1 Trade Filtering (v2)

When trade data is eventually used for enrichment, only genuine aggressor trades pass the filter:

| Price | Classification | Action |
|---|---|---|
| $P = B_{\text{bid}}$ | Real — someone hit the bid | Bearish confirmation |
| $P = A_{\text{ask}}$ | Real — someone lifted the ask | Bullish confirmation |
| $P = \frac{B+A}{2}$ | Synthetic / chart-generated | **Discard** |
| Otherwise | Synthetic | **Discard** |

---

## 31.3 1. Density Matrix Construction

### 31.3.1 Bloch Sphere Parameterization

Every 2×2 density matrix can be written in terms of the Bloch vector $\vec{r} = (r_x, r_y, r_z)$ and the Pauli matrices $\vec{\sigma}$:

$$\rho = \frac{1}{2}\left(I + \vec{r} \cdot \vec{\sigma}\right)$$

For real-valued financial data we set $r_y = 0$ (no imaginary phase), giving:

$$\boxed{\rho = \frac{1}{2}\begin{pmatrix} 1 + r_z & r_x \\ r_x & 1 - r_z \end{pmatrix}}$$

Constraint: $r_z^2 + r_x^2 \leq 1$ (positive semi-definite).

### 31.3.2 Computing $r_z$ — Directional Bias

From a sliding time window of BBO samples, each sample $i$ has: - **Direction:** $d_i = \text{sign}(\text{mid}_i - \text{mid}_{i-1}) \in \{-1, 0, +1\}$ - **Weight:** $w_i = \min(\text{bidSize}_i, \text{askSize}_i)$ — two-sided liquidity conviction

$$r_z = \frac{\sum_i d_i \cdot w_i}{\sum_i w_i}$$

$r_z \in [-1, +1]$. Positive = bullish BBO drift. Negative = bearish BBO drift. Zero = no net direction.

### 31.3.3 Computing $r_x$ — Serial Coherence

For consecutive sample pairs $(i-1, i)$ within the window:

$$r_x = \frac{\sum_i d_{i-1} \cdot d_i \cdot \sqrt{w_{i-1} \cdot w_i}}{\sum_i \sqrt{w_{i-1} \cdot w_i}}$$

| $r_x$ | Interpretation |
|---|---|
| $\to +1$ | Consecutive BBO moves same-direction — coherent trending |
| $\to 0$ | Random walk BBO — no serial structure |
| $\to -1$ | BBO moves alternate — oscillating/choppy (market maker battle) |

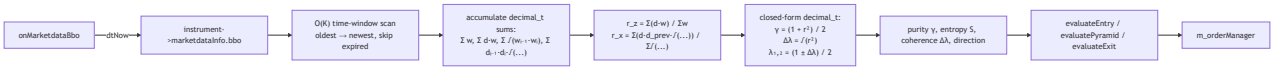
### 31.3.4 Computation

The calculator does **not** maintain its own BBO buffer. Instead, it scans the instrument's existing `marketdataBboInfo_s` ring buffer directly on each `compute()` call. The scan walks oldest->newest, skipping entries with `dtTimestamp < now - windowMs` (time-based expiry). The scan is $O(K)$ where $K$ = samples in window (typically 10–500 at 50ms throttle).

Edge cases: - If fewer than `minSamplesInWindow` (default 3) samples are within the window -> return maximally mixed default ($\gamma = 0.5$) - If $r_z^2 + r_x^2 > 1$ -> rescale both to lie on the Bloch sphere - Zero memory overhead — no duplicate buffer, no heap allocations

---

## 31.4 2. Signal Computation (Closed-Form, No Eigen)



For a 2×2 density matrix, the eigendecomposition has a trivial closed form — no numerical solver needed. All computation uses `decimal_t` only; no floating-point anywhere in the signal path.

| Quantity | Formula | Range | Meaning |
|---|---|---|---|
| $r_z$ | $\frac{\sum d_i \cdot w_i}{\sum w_i}$ | $[-1, +1]$ | Size-weighted directional bias |
| $r_x$ | $\frac{\sum d_{i-1} \cdot d_i \cdot \sqrt{w_{i-1} \cdot w_i}}{\sum \sqrt{w_{i-1} \cdot w_i}}$ | $[-1, +1]$ | Serial coherence (+ trending, - alternating) |
| Purity $\gamma$ | $\frac{1+r_z^2+r_x^2}{2}$ | $[0.5, 1.0]$ | 0.5 = maximally mixed (noise), 1.0 = pure state |
| Coherence $\Delta\lambda$ | $\sqrt{r_z^2 + r_x^2}$ | $[0, 1]$ | How strongly one direction dominates |
| Entropy $S$ | $-\sum \lambda_i \log_2 \lambda_i$ | $[0, 1]$ | 0 = pure state, 1 = maximally mixed |
| Direction | sign$(r_z)$ if $r_z \neq 0$, else sign$(r_x)$ | $\{-1, +1\}$ | $+1$ = long bias, $-1$ = short bias |

We primarily trade on **purity thresholds**.

---

## 31.5 3. Entry Rules

### 31.5.1 Entry Conditions (ALL must be true)

| # | Condition | Default | Rationale |
|---|---|---|---|
| 1 | $\gamma > \gamma_{\text{entry}}$ | 0.75 | Market has decided |
| 2 | $\|r_z\| > r_z^{\text{min}}$ | 0.15 | Non-trivial directional bias |
| 3 | $r_x > r_x^{trend}$ | 0.10 | Consecutive BBO moves must exceed trend threshold |
| 4 | $\Delta t_{\text{since last exit}} > t_{\text{cooldown}}$ | 5000ms | Prevent immediate re-entry |

The $r_x > r_x^{trend}$ condition separates trending from oscillating markets:

| $r_z$ | $r_x$ | Interpretation | Action |
|---|---|---|---|
| + | **+** | Bullish bias + trending BBO (consecutive upward ticks) | **Enter LONG** |
| - | **+** | Bearish bias + trending BBO (consecutive downward ticks) | **Enter SHORT** |
| any | **-** | BBO alternating — market maker oscillation, no directional signal | No trade |
| any | **~= 0** | Random walk — no serial structure | No trade |

The sign of $r_x$ encodes **serial consistency** (+ same-direction, - alternating), NOT directional alignment with $r_z$. A bearish trend with consecutive downward ticks produces $r_x > r_x^{trend}, r_z < 0$ — both conditions pass and the trade direction is given by $\text{sign}(r_z)$.

On entry: place market order, record $\gamma_{\text{entry}}$, $r_z^{\text{entry}}$, timestamp. After fill, place coherence-weighted take-profit limit order.

---

## 31.6 4. Pyramid Rules

### 31.6.1 Pyramid Conditions (ALL must be true)

| # | Condition | Default | Rationale |
|---|---|---|---|
| 1 | $\gamma > \gamma_{\text{pyramid}}$ | 0.85 | Higher bar than entry |
| 2 | $\gamma > \gamma_{\text{lastEntry}} \times (1+\delta)$ | $\delta = 0.03$ | Conviction is *still rising* |
| 3 | $\text{sign}(r_z) =$ position direction | — | Still aligned |
| 4 | $r_x > r_x^{trend}$ | 0.10 | Still trending above configured threshold |
| 5 | unrealizedPnL (bps) $>$ $b_{\text{min}} \times (n+1)$ | $b_{\text{min}} = 5.0$ | Profit cushion, tiered per pyramid |
| 6 | pyramidCount $<$ maxPyramids | 2 | Max 3× base size |
| 7 | $\Delta t_{\text{since last pyramid}} > t_{\text{pyramidInterval}}$ | 10000ms | Don't stack pyramids on a spike |
| 8 | totalSize+pyramidSize $\leq$ maxPositionSize | — | Hard cap |

### 31.6.2 Pyramid Sizing (Diminishing)

$$\text{size}_k = \text{baseSize} \times s^k$$

where $s$ is the pyramid scale factor (default 0.5):

| Level | Size multiplier | Cumulative |
|---|---|---|
| Entry | 1.0× | 1.0× |
| Pyramid 1 | 0.5× | 1.5× |
| Pyramid 2 | 0.25× | 1.75× |

### 31.6.3 Tiered PnL Cushion

The minimum unrealized PnL required grows with each pyramid level:

| Pyramid # | Min unrealized PnL (bps) | Rationale |
|---|---|---|
| 1st add | $> 5$ | Covers spread + taker fee on the add |
| 2nd add | $> 10$ | Must be deeper in profit — conviction confirmed by price |

A raw $> 0$ PnL check would allow pyramiding at +0.3 bps, which the spread alone can flip negative. The tiered cushion ensures the market has **earned** the pyramid.

---

## 31.7 5. Exit Rules

### 31.7.1 Exit Triggers (ANY triggers exit)

| # | Condition | Default | Exit type |
|---|---|---|---|
| 1 | $\gamma < \gamma_{\text{exit}}$ | 0.55 | Purity collapsed |
| 2 | $r_z$ crosses zero against position | $\|r_z\| > r_z^{\text{flip}}$ (0.10) | Direction reversed |
| 3 | $r_x \leq r_x^{trend}$ while in position | 0.10 | Trend broken (alternation / weak structure) |
| 4 | $\Delta t_{\text{in position}} > t_{\text{maxHold}}$ | 600000ms (10min) | Max hold reached |
| 5 | Stop loss triggered | via `algoStopLossCalculator` | Risk exit |

### 31.7.2 PnL-Aware Signal Exit

When $\gamma < \gamma_{\text{exit}}$ (signal decay):

| PnL status | Action | Rationale |
|---|---|---|
| In profit | Cancel all TPs -> place limit close at mid -> if not filled, market fallback after `tpGraceMs` | Save spread first, guarantee eventual flat |
| In loss | Cancel all TPs -> market exit immediately | Minimize further loss |

---

## 31.8 6. Coherence-Weighted Take Profit

### 31.8.1 Dynamic TP Distance

The TP distance scales with the purity at entry time — higher conviction = wider target:

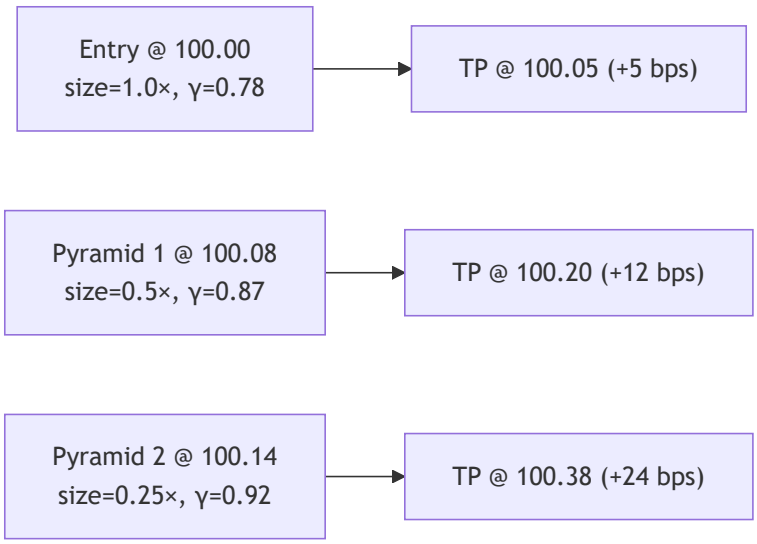
$$\text{tpBps} = \text{tpMin} + (\text{tpMax} - \text{tpMin}) \times \frac{\gamma - \gamma_{\text{entry}}}{\gamma_{\text{pyramid}} - \gamma_{\text{entry}}}$$

Clamped to $[\text{tpMin}, \text{tpMax}]$.

| $\gamma$ at entry | TP distance | Interpretation |
|---|---|---|
| 0.76 (near entry threshold) | ~5 bps | Marginal conviction, take profit quickly |
| 0.85 (pyramid threshold) | ~15 bps | Strong conviction, let it run |
| 0.92 (exceptional) | ~25 bps | Exceptional conviction, wide target |

### 31.8.2 Per-Pyramid TP Management

Each entry layer gets its own TP:



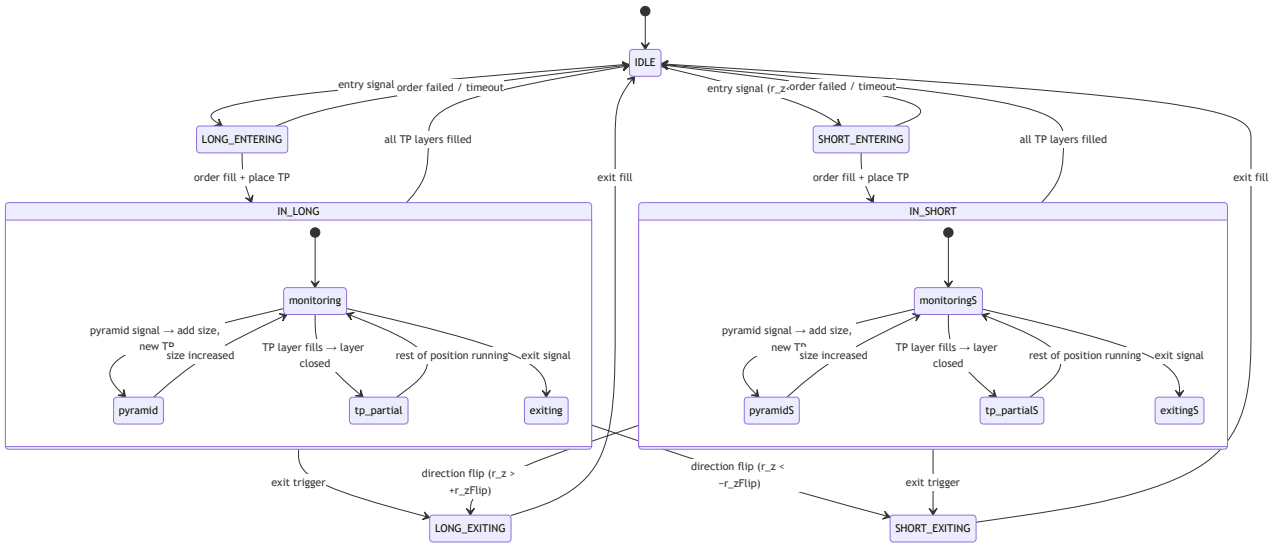

- When a TP fills, only that layer closes — remaining position continues
- When signal exits, all unfilled TPs are canceled

- Later pyramids naturally get wider TPs (entered at higher $\gamma$)

### 31.8.3 Exit Priority Order

| Priority | Mechanism | Trigger | Order type |
|---|---|---|---|
| 1 | TP fill | Price reaches coherence-weighted target | Limit (resting) |
| 2 | Signal exit, green | $\gamma < \gamma_{exit}$, PnL > 0 | Limit at mid -> market after grace |
| 3 | Signal exit, red | $\gamma < \gamma_{exit}$, PnL < 0 | Market (immediate) |
| 4 | Direction flip / coherence break | see exit rules | Market |
| 5 | Stop loss | Price hits stop | Market |

## 31.9 7. Full State Machine



# Chapter 32

# algoMIRE — Refill-Hazard-Conditioned Flow Resolution (CME Nano Futures)

`algoMIRE` is a bounded online-ML intraday trading algorithm for the ttTrader framework. It trades one configured CME nano future (primary: IBKR NES front month) on the thesis that a top-of-book sweep's *resolution direction* (revert vs. continue) is predicted by two latent quantities: **refill hazard** (how quickly resting liquidity rebuilds the drained level) and **trade-arrival self-excitation** (whether the sweep ignited a follow-on cluster). Fast refill + calm arrivals = liquidity-provision absorption (reversion edge); slow refill + self-excited arrivals = informed flow (continuation). The same imbalance feature flips sign with the hazard state — the explicit `sweep x refill` / `sweep x calm` interaction terms feed that conditioning directly to the online model.

Plan of record: `.kilo/algoPlans/algoMIRE-refill-hazard-flow-reversion-Qwen3.8-MAX.md`.

## 32.1 How It Works

### 32.1.1 Core Logic

1. **Tape accumulation** (per tick / per quote, allocation-free): same-direction trade runs (sweeps), Cont-style L1 order-flow imbalance, micro-price deviation, post-sweep depth-rebuild observation (refill), post-sweep cluster counting (excitation), spread channel.

2. **Per closed 1m bar**: the tape accumulators fold into a 15-dim model input (13 features + 2 interactions), the regime detector classifies, the signal model trains on the sample whose `horizonBars` label just closed, and the decision layer runs. No per-tick training, no look-ahead (sequence-pinned indicator reads).

3. **Signal**: an Eigen online logistic regression (Welford-normalized inputs, bounded SGD, L2, weight clipping) outputs `pUp`; `pUp >= entryThreshold` wants long, `pUp <= 1-entryThreshold` wants short. A drift monitor freezes predictions at 0.5 when the rolling loss EMA stays above 1.5x its long baseline for 200 bars (training continues; recovery below 1.2x releases).

4. **Regime detector** (4 states, 5-bar hysteresis):
   - `STRESS` — vol z > 1.5 or spread z > 2.5: entries frozen, stop x1.5, size x0.5; sustained STRESS (`stressHaltBars`) escalates to the kill-switch halt.
   - `QUIET` — vol z < -0.25 and squeeze z < 0: size x0.75, stop x1.25, daily trade budget halved.
   - `TREND` — trend gate + excite > 1.2: only the trend direction is tradeable, stop x0.9.
   - `NEUTRAL` — defaults.
   - A realized-vol percentile ring (99th) freezes entries independently of the state.

5. **Entries**: passive LIMIT at the touch (buy at bid / sell at ask) with a timeout cancel — no chase. Gates: session window, regime-scaled trade budget, regime freeze, min training, spread, bbo staleness, signal, direction permission, cooldown, no stacking.

6. **Position management**:

   - Risk-based sizing via `CalculatePositionSize` with the regime-scaled ATR stop distance, then `regimeMultiplier x confidenceMultiplier x winRateMultiplier` (product capped, caps only shrink), hard contract/notional caps, grid snap.
   - Protection placed immediately on the entry fill: native stop-market plus the take-profit ladder — partial close-limit at `partialTpAtr x ATR` taking `partialTpFraction`, remaining close-limit at `tpAtr x ATR`. When the partial leg fills, the stop ratchets to breakeven.
   - Exits: signal decay through `exitThreshold`, max hold, session flatten, daily-loss latch. No direct reversals.

7. **Kill-switches** (flatten + HALTED until the next UTC day): spread-z anomaly sustained `spreadKillMs`, bbo staleness/data gap beyond `staleBboMs`, high-water drawdown breach `ddKillPct`, sustained STRESS, session-end flatten.

## 32.2 Feature Table (15 model inputs)

| Index | Feature | Source | Description |
|---|---|---|---|
| 0 | `mSweepDepth` | tape | largest same-direction trade run's signed bps move / bar ATR1m, clamped ±20 |
| 1 | `mRefillHz` | bbo | EMA of post-sweep depth-rebuild rate / its 60-bar rolling median, clamped [0,10] |
| 2 | `mExciteRatio` | tape | EMA of cluster-trades-within-500ms / sweep-trades, clamped [0,3] |
| 3 | `mOfiZ` | bbo+tape | (Cont L1 OFI + signed volume) z-scored over 60 bars, clamped ±10 |
| 4 | `mMicroDev` | bbo | EMA of (microprice - mid) / half-spread, clamped ±5 |
| 5 | `mSpreadDyn` | bbo | (spread_bps - mean60) / (std60 + eps), clamped ±10 |
| 6 | `mValuePos` | indicator | volume-profile value-area position + POC distance in ATR units |
| 7 | `mRvPressure` | indicator | RVOL(1m) x candle close-location value |
| 8 | `mRangeCompress` | indicator | bollinger width z-score over 240 bars, clamped ±10 |
| 9 | `mTrendCtx` | 5m indicator | linreg slope sign x clamp(ADX/25, 0, 2) |
| 10 | `mVolRegime` | both | ATR1m/ATR5m ratio z-score, clamped ±10 |
| 11 | `mHourSin` | clock | sin of utc minute-of-day phase |
| 12 | `mHourCos` | clock | cos of the same phase |
| 13 | `intSweepRefill` | 0 x 1 | hazard interaction: sweep x refill |
| 14 | `intSweepCalm` | 0 x 2 | hazard interaction: sweep x (1 - excite/3) |

Tape-derived rings warm from live bars only (the vector turns valid after ~60 live minutes); candle-derived rings

warm from historical replay. Training is anchored on live tape via the `minTrainBars` entry lock after a restart.

## 32.3 Configuration Parameters

| Parameter | Default | Valid Range | Description |
|---|---|---|---|
| `instTradeId` | (required) | - | Traded instrument id |
| `ohlcId1m` | (required) | - | 1m signal OHLC feed id |
| `ohlcId5m` | (required) | - | 5m context OHLC feed id |
| `atrIndicatorId` | (required) | - | 1m ATR indicator id |
| `bollingerIndicatorId` | (required) | - | 1m bollinger indicator id (width output) |
| `vpIndicatorId` | (required) | - | 1m volume profile indicator id |
| `rvolIndicatorId` | (required) | - | 1m RVOL indicator id |
| `atr5mIndicatorId` | (required) | - | 5m ATR indicator id |
| `linregIndicatorId` | (required) | - | 5m linreg indicator id (slope output) |
| `adxIndicatorId` | (required) | - | 5m ADX indicator id |
| `ohlc1mTimeFrameMin` | 1 | >= 1 | 1m series timeframe |
| `ohlc5mTimeFrameMin` | 5 | >= 1m timeframe | 5m series timeframe |
| `risk.type` | PERCENT | PERCENT\|BPS\|MIN\|FIXED | Risk mode for position sizing |
| `risk.value` | 0.20 | 0.05 - 0.5 % | Risk fraction per trade |
| `sessionMode` | daily | continuous\|daily | `daily` flattens at `flattenMinuteUtc`; `continuous` for 24/7 venues only |
| `flattenMinuteUtc` | 1290 | 0-1439 | Minutes since midnight UTC to flatten (default 21:30) |
| `noEntryMinutes` | 45 | 0-1440 | Minutes before flatten where no new entries are accepted |
| `entryThreshold` | 0.62 | 0.50 - 0.95 | Probability gate: >= this -> long, <= 1-this -> short |
| `exitThreshold` | 0.42 | 0.05 - 0.50 | Signal decay exit band |
| `minTrainBars` | 300 | 1-1000 | Minimum trained bars before entries are allowed |
| `horizonBars` | 5 | 2-15 | Bars ahead for the training label |
| `learnRate` | 0.03 | 1e-5 - 1.0 | SGD learning rate |
| `l2` | 0.0001 | 0 - 1.0 | L2 regularization strength |
| `performanceEmaAlpha` | 0.1 | 0.001 - 1.0 | EMA smoothing for the win-rate tracker |
| `refillWindowMs` | 2000 | 100-60000 | Post-sweep depth-rebuild observation window |
| `exciteWindowMs` | 500 | 50-10000 | Post-sweep cluster trade window |
| `stopAtrMultiplier` | 1.6 | >= 0.1 | Stop distance in ATR units (regime-scaled) |
| `tpAtrMultiplier` | 2.2 | >= 0.1 | Final take-profit distance in ATR units |
| `partialTpAtr` | 1.0 | 0 - tpAtr | Partial take-profit distance in ATR units |
| `partialTpFraction` | 0.5 | 0 - 0.9 | Fraction closed by the partial leg |

| Parameter | Default | Valid Range | Description |
|---|---|---|---|
| `entryTimeoutMs` | 15000 | 1000-600000 | Passive entry timeout (cancel, no chase) |
| `maxHoldMs` | 1500000 | >= 1000 | Max hold time (default 25 min) |
| `maxEntrySpreadBps` | 4 | >= 0 | Max spread in bps for entry |
| `staleBboMs` | 5000 | 1000-3600000 | BBO staleness kill / entry gate |
| `volFreezePercentile` | 0.99 | 0.5 - 1.0 | Realized-vol percentile freeze guard |
| `trendGateMin` | 0.5 | 0 - 2 | Min \|trendCtx\| for the TREND state |
| `stressHaltBars` | 3 | 1-100 | Consecutive STRESS-classified bars before the halt |
| `spreadKillZ` | 3.0 | 1 - 20 | Spread-z anomaly kill level |
| `spreadKillMs` | 10000 | 1000-3600000 | Sustained spread anomaly duration before the kill |
| `ddKillPct` | 0.08 | 0 - 0.9 | High-water drawdown breach fraction |
| `cooldownMs` | 10000 | >= 1000 | Cooldown between entries after trade close |
| `regimeMinTrades` | 20 | 1-500 | Closed trades before the win-rate multiplier activates |
| `regimeMinMultiplier` | 0.5 | 0.1 - 1.0 | Min size multiplier when active |
| `regimeMaxMultiplier` | 1.25 | min - 3.0 | Max size multiplier when active |
| `maxTradesPerSession` | 25 | 0-100000 (0 = unlimited) | Daily trade cap (regime-scaled) |
| `maxDailyLossPct` | 0.02 | >= 0 | Daily loss limit as equity fraction |
| `lossStreakTrades` | 3 | 0-100 (0 = off) | Consecutive losses arming the streak freeze |
| `lossStreakCooldownMs` | 300000 | 1000-86400000 | Loss-streak entry freeze duration |
| `maxPositionContracts` | 10 | >= 1 | Hard per-position contract cap |
| `maxNotionalUsd` | 50000 | >= 0 | Hard per-position notional cap |
| `reduceOnlyClose` | true | bool | Reduce-only close-order guard; clear only for multi-algo netting on one account/instrument |
| `stateFile` | "state/algoMIRE-state.json" | - | State persistence path (framework-managed) |

Balance share is not a `parameter` key: declare it per asset in the algo's `assets` list (`{ "id": "assetInterActiveBrokersUSD", "balanceFraction": 1.0 }`). `storeAlgo_c` resolves the share across all algos, and sizing then uses a first-snapshot allocation plus only this instance's own realized PnL, so a losing algo never sizes against another algo's profit. See docs/05-trading-algorithms.

The confidence scaler's [0.4, 1.0] envelope and its 0.7 cold-start fallback are fixed plan values (not configurable). The combined size-multiplier product is capped at 1.5.

## 32.4 Indicator Subscriptions (V2)

- 1m series (`source: "close"`, `historyDepth: 512`): `atr14`, `bollinger(20,2)`, `volumeProfile(period 240, bins 48, valueAreaPercent 70)`, `rvol(60)`.
- 5m series (`historyDepth: 512`): `atr14`, `linreg(period 24)`, `adx(14)`.

## 32.5 State Persistence

- Model (weights bit-exact, Welford stats, drift monitor, performance), regime (classifier + vol ring), session (daily counters), sizer (confidence weights), cooldown, halt flag, high-water equity — all via validate-then-commit JSON with atomic rename.
- The pending-label ring is NOT persisted: labels require live bars; after a restart the entry lock (`minTrainBars` on the restored train count) governs.
- Saved on every trade close, every 5 minutes (checkpoint) and on exit.

## 32.6 Sample Configuration

Ready-to-run ttTrader configs live in `config/`:

- `config/algoMIRE-ibkr-nes-front.json` — IBKR CME E-nano S&P 500 (NES) front month, daily session (flatten 21:30 UTC). The front-month conid must be set on every quarterly roll (see the header of the file and `config/instrumentsIbkr.cfg`).

The certification-run analog also exists at `exchanges/exchgInteractiveBrokers/algoMIRE-nes.json` (same parameter set; both share the `state/algoMIRE-NES-state.json` state file).

```
ttTrader.exe algos/algoMIRE/config/algoMIRE-ibkr-nes-front.json --log --logmarketdata
ttTrader.exe algos/algoMIRE/config/algoMIRE-ibkr-nes-front.json --log --tradingMode paper --logmark
```

## 32.7 Important Constraints

1. **decimal_t everywhere**: all financial values use `decimal_t`; `double` only inside the Eigen/confidence boundaries with explicit justification comments.
2. **L1-only by design**: works without L2 depth (tape + bbo tick-by-tick).
3. **No per-tick training**: strictly once per finalized 1m bar.
4. **Allocation-free hot path**: fixed rings/members in the per-tick and per-bar paths.
5. **One instrument per instance**: multiple instances can run for portfolio trading.
6. **Session mode**: `daily` only on venues with defined trading hours; `continuous` only for 24/7 venues.
7. **Halt semantics**: kill-switches flatten and latch HALTED until the next UTC day boundary (calendar-safe; an overnight data-gap halt does not eat the following session).

## 32.8 Building and Testing

```
# Build the plugin
cmake --build .build/clang-debug --target algoMIRE

# Unit tests (deterministic, hermetic)
.build/clang-debug/ttManagerOrderTests.exe Mire
```

Suites: `MireFeatures` (tape features, warm-up, degenerates), `MireModel` (convergence, drift freeze, state round trips), `MireRegime` (classification, hysteresis, vol freeze), `MireSizer` (confidence scaler, caps), `MireSessionGuard` (boundaries, limits, streak), `MireConfig` (clamps).

## 32.9 Validation Protocol (plan §6)

Walk-forward on recorded `.ttpb` quote replay: expanding train (>= 40 sessions), purged 5-fold validation with 10-bar embargo, fixed unseen test (>= 20 sessions); deflated Sharpe, bootstrap CIs, White's reality check; crash/zero-vol/data-gap/fee+50%/reconnect stress tests; >= 15 live paper sessions before go-live. No parameter touch after test lock.

## 32.10 License

Part of the ttTrader HFT framework. See project root for full license.

# Chapter 33

# algoMLD — Bounded Online-ML Intraday Futures/Perpetual Strategy

`algoMLD` is a bounded online machine-learning intraday trading algorithm for the ttTrader framework. It trades one configured perpetual or futures instrument using completed OHLC bars and current BBO protection.

## 33.1 How It Works

### 33.1.1 Core Logic

1. **Bar Processing**: On each closed OHLC bar, the algo computes 10 features (momentum, volatility, candle shape, volume z-score, order-flow pressure, position within 20-bar range) and updates the online ML model.
2. **Model Training**: A forward-return label (1 if price rose next `horizonBars` bars, 0 otherwise) is generated after the required lookback period. The Eigen-based logistic regression model trains on every closed bar using stochastic gradient descent with L2 regularization.
3. **Entry Signal**:
   - Probability >= `entryThreshold` -> long entry
   - Probability <= 1 - `entryThreshold` -> short entry
   - Multiple gates: session open, spread check, vol percentile, minimum training bars, cooldown
4. **Position Management**:
   - **Risk-based sizing**: Uses `algoPositionManager_c::CalculatePositionSize` with ATR stop distance
   - **Regime multiplier**: Win-rate EMA scaled size multiplier (0.5x-1.25x) activated after `regimeMinTrades` (default 20)
   - **Stop loss**: `stopAtrMultiplier × ATR` from entry price
   - **Take profit**: `tpAtrMultiplier × ATR` from entry price
   - **Maximum hold**: Configurable `maxHoldMs` (default 30 min) forces market exit
5. **Exits** (symmetric around the 0.5 baseline):
   - Long: probability decays below `exitThreshold` -> market exit
   - Short: probability rises above 1 - `exitThreshold` -> market exit

- Max hold reached -> market exit
- Session flatten time -> market close (daily mode)
- Session guard limits (max trades/day, max loss %)

6. **Protection**: Stop-limit and take-profit orders placed immediately after entry fill.

### 33.1.2 Key Design Decisions

- **No overnight positions**: Configurable session mode - `daily` (flatten at configurable UTC time) or `continuous` (24/7 venues, no flatten)
- **Eigen isolated**: `double` used only inside Eigen boundary; all trading logic uses `decimal_t`
- **Bounded adaptation**: Regime multiplier prevents over-scaling; volatility filter blocks entries in high-vol regimes
- **State persistence**: Model weights, normalization stats, and performance counters saved/loaded from JSON
- **One instrument per instance**: Multiple instances can run for portfolio trading

## 33.2 Configuration Parameters

| Parameter | Default | Valid Range | Description |
|---|---|---|---|
| `instTradeId` | (required) | - | Instrument trade ID, must match algo config in the instrument subscription |
| `ohlcId` | (required) | - | OHLC feed ID, must be subscribed in the algo's instruments section |
| `ohlcTimeFrameMin` | 5 | >= 1 | OHLC bar timeframe in minutes |
| `risk.type` | PERCENT | PERCENT\|BPS\|MIN\|FIXED | Risk mode for position sizing |
| `risk.value` | 0.25 | 0.01 - 1.0 | Risk amount (percent, bps, min size, or fixed) |
| `sessionMode` | daily | continuous\|daily | `daily` flattens at `flattenMinuteUtc`; `continuous` for 24/7 venues only |
| `flattenMinuteUtc` | 1320 | 0-1439 | Minutes since midnight UTC to flatten (default 22:00 = 1320) |
| `noEntryMinutes` | 30 | 0-1440 | Minutes before flatten where no new entries accepted |
| `entryThreshold` | 0.60 | 0.50 - 0.95 | Probability gate: >= this -> long, <= 1-this -> short |
| `exitThreshold` | 0.40 | 0.05 - 0.50 | Signal decay exit: long exits below this, short exits above 1-this |
| `minTrainBars` | 200 | 1-1000 | Minimum bars trained before entries allowed |
| `stopAtrMultiplier` | 1.5 | >= 0.1 | ATR multiplier for stop loss distance |
| `tpAtrMultiplier` | 2.5 | >= 0.1 | ATR multiplier for take profit distance |

| Parameter | Default | Valid Range | Description |
|---|---|---|---|
| `maxEntrySpreadBps` | 12 | >= 0 | Maximum spread in bps for entry to be accepted |
| `maxVolPercentile` | 0.95 | 0.0 - 1.0 | Volatility percentile ring buffer; entries blocked when > this |
| `cooldownMs` | 10000 | >= 1000 | Cooldown between entries after trade close (ms) |
| `maxHoldMs` | 1800000 | >= 1000 | Maximum hold time; forces exit after this many ms (default 30 min) |
| `horizonBars` | 5 | 1-50 | Bars ahead for label generation in model training |
| `learnRate` | 0.05 | 1e-5 - 1.0 | SGD learning rate for online logistic regression |
| `l2` | 0.0001 | 0 - 1.0 | L2 regularization strength |
| `performanceEmaAlpha` | 0.1 | 0.001 - 1.0 | EMA smoothing alpha for win-rate tracking |
| `regimeMinTrades` | 20 | 1-500 | Minimum closed trades before regime multiplier activates |
| `regimeMinMultiplier` | 0.5 | > 0 | Minimum size multiplier when regime active |
| `regimeMaxMultiplier` | 1.25 | > regimeMinMultiplier | Maximum size multiplier when regime active |
| `stateFile` | "state/algoMLD-state.json" | - | JSON file path for model state persistence (framework-managed; relative paths resolve against the deploy directory) |

### 33.2.1 Feature Details (10 features from `featureBuilder.h`)

| Index | Feature | Description |
|---|---|---|
| 0 | 1m return | (close - close_1) / close_1 |
| 1 | 5m return | (close - close_5) / close_5 |
| 2 | 20m return | (close - close_20) / close_20 |
| 3 | 20-bar std | Standard deviation of 20 one-bar returns |
| 4 | Range % | (high - low) / close |
| 5 | Close-open % | (close - open) / close |
| 6 | Wick bias | (lowerWick - upperWick) / range (positive = buyers defend lows) |
| 7 | Volume z-score | (currentVol - meanVol20) / stdVol20 |
| 8 | CVD delta | (CVD_now - CVD_5barago) / volumeSum_5bar |
| 9 | Close in 20b range | (close - low20) / range20 (0=at low, 1=at high) |

### 33.2.2 State Persistence

- Model state (weights, normalization stats, performance counters, vol buffer) saved to JSON file
- Automatically saved on `onExitAlgo()` and after each closed trade
- On restart: model warm-start from saved state; if no state file, starts fresh

### 33.2.3 Risk and Money Management Notes

- Position size calculated from stop distance via `CalculatePositionSize`:
  - `size = floor_to_grid(min(risk_budget / (stop_distance × contract_size), notional_cap / entry_price, margin_cap))`
- Regime multiplier applied BEFORE caps/grid checks (never overrides hard limits)
- Handles missing/empty balances, zero stop distance, inverse contract semantics
- Size-grid rounding and minimum notional enforced
- Loss streak/cooldown and max trades per session respected

## 33.3 Sample Configuration Files

Ready-to-run ttTrader configs (plugins, assets, exchange, instrument with the 5m OHLC feed, and the algo parameter block) live in `config/`:

- `config/algoMLD-ibkr-nq-front.json` — IBKR CME E-mini Nasdaq-100 front month, daily session (flatten 22:00 UTC). The front-month conid must be set on every quarterly roll (see the header of the file and `config/instrumentsIbkr.cfg`).
- `config/algoMLD-lighter-btcusd-perp.json` — Lighter BTC/USDC perpetual, continuous session (24/7 venue).

Run with:

```
ttTrader.exe <config file> --log --logmarketdata
ttTrader.exe <config file> --log --tradingMode paper --logmarketdata
```

The `parameter` block inside both files carries the full algoMLD parameter set shown below (keys mirror the parameter table above).

## 33.4 Parameter Impact Guide

### 33.4.1 Entry/Exit Thresholds

- **entryThreshold** (0.5-0.95): Higher = fewer entries, higher quality signals
- **exitThreshold** (0.05-0.5): Lower = fewer exits, trades run longer; higher = more frequent exits
- Typical: entry 0.60, exit 0.40 gives ~80% of signal probability gap used

### 33.4.2 Risk Parameters

- **stopAtrMultiplier** (0.1+): Larger = wider stops, smaller position size; smaller = tighter stops, larger size (but more stop-outs)
- **tpAtrMultiplier** (0.1+): Larger = farther targets, better risk:reward ratio
- Standard: 1.5 / 2.5 gives ~1:1.67 risk:reward

### 33.4.3 Regime Adaptation

- **regimeMinTrades** (1-500): Fewer trades = regime activates sooner (but less reliable); more trades = stabler signal
- **regimeMinMultiplier** (0.1-1.0): Below 1.0 shrinks size when win-rate < 50%; above 1.0 expands when win-rate > 50%
- **regimeMaxMultiplier** (1.0-3.0): Caps maximum expansion; prevents over-scaling

### 33.4.4 Volatility Filter

- **maxVolPercentile** (0.0-1.0): Higher = allows more entries in volatile markets; lower = conservative, avoids choppy periods
- Default 0.95 blocks entries when vol in top 5% of historical

### 33.4.5 Time Controls

- **maxHoldMs** (1000+): Protects against long-held losing positions; default 30 min for intraday
- **cooldownMs** (1000+): Prevents revenge trading; default 10 sec
- **flattenMinuteUtc** (0-1439): Daily flatten time; 1320 = 22:00 UTC
- **noEntryMinutes** (0-1440): Minutes before flatten where no new entries

### 33.4.6 Model Learning

- **learnRate** (1e-5-1.0): Higher = faster adaptation; too high = unstable weights
- **l2** (0-1.0): Higher = more shrinkage; default 1e-4 prevents weight explosion
- **performanceEmaAlpha** (0.001-1.0): Higher = win-rate EMA reacts faster; default 0.1 balances responsiveness

## 33.5 Important Constraints

1. **decimal_t everywhere**: All financial values use `decimal_t`; `double` only inside Eigen model boundary with explicit justification
2. **No float/double in trading logic**: Financial calculations (size, price, PnL) strictly decimal_t
3. **Eigen isolated**: Normalization and weight math use `double`; predictions cross back as `decimal_t::fromDouble()`
4. **Session mode**: `daily` only on venues with defined trading hours; `continuous` only for 24/7 venues
5. **Model dormant**: Regime multiplier = 1.0 until `regimeMinTrades` (default 20) closed trades executed
6. **Persistence safety**: State file written via atomic rename (.tmp -> real path); failure leaves previous state intact
7. **One instrument per algo instance**: Multiple algos can run for portfolio diversification

## 33.6 Building and Running

```
# Build the plugin
cmake --build .build/clang-debug --target algoMLD

# Run unit tests (model + session guard suites)
.build/clang-debug/ttManagerOrderTests.exe MldModel
.build/clang-debug/ttManagerOrderTests.exe MldSessionGuard

# Full test suite
.build/clang-debug/ttManagerOrderTests.exe
```

## 33.7 License

Part of the ttTrader HFT framework. See project root for full license.

# Chapter 34

# algoQIS — order-book survival-probability asymmetry

Regime-conditional microstructure strategy for crypto perpetual futures. Two online logistic models estimate the survival probability of the resting bid and ask liquidity (which side the next trade hits). The differential `S_ask - S_bid` is the raw directional signal; it is weighted by the micro-price entropy (low entropy = deterministic, predictable flow) and the volatility regime, then gated by a 4-state microstructural regime detector. Entries are MARKET orders because the signal assumes immediate execution; protection is a native stop-market plus a take-profit close-limit placed on the fill. Signal-decay, max-hold, stale-feed and session exits run on a 1 s timer. No LLM, no background thread, deterministic in-process only.

## 34.1 Components

| File | Responsibility |
|---|---|
| `algoConfig.h/.cpp` | parameter parse with clamp envelopes and cross-field repair |
| `featureBuilder.h/.cpp` | per-bar aggregate ring and the 16 clipped features |
| `mlModel.h/.cpp` | bid/ask online logistic pair, Welford normalization, entropy calculator, vol-percentile ring, next-trade labels |
| `regimeDetector.h/.cpp` | LOW_VOL_TREND / LOW_VOL_RANGE / HIGH_VOL_BREAKOUT / HIGH_VOL_CHOP with hysteresis |
| `positionSizer.h/.cpp` | sizing stack and pure risk envelopes (confidence, caps, ATR geometry) |
| `sessionGuard.h/.cpp` | continuous/daily flatten, no-entry window, daily trade/loss limits |
| `orderPath.h` | pure market-entry / protection-restore / decay-exit / direction-gate decisions |
| `algoCore.h/.cpp` | plugin wiring, bar pipeline, order flow, state persistence |

## 34.2 Signal pipeline

1. Every venue feed updates the bar tracker: trades aggregate per-bar cvd/size and feed the next-trade label, BBO updates drive the entropy series and the micro-price/spread/staleness state, book updates drive the top-5 depth sums.
2. At every closed 5m bar: push the bar aggregate -> build the 16 features -> predict `S_ask`, `S_bid` -> capture the 8-input vector for the next-trade label -> classify the regime -> compute `signalScore`.
3. Entry gates: state IDLE, no position/orders, session/cooldown, fresh BBO within `bboStaleSeconds`, spread <= `maxEntrySpreadBps`, both models at `minTrainBars`, no anomaly freeze, vol percentile <= `maxVolPercentile`, `|signalScore| >= max(signalThreshold, regime gate)` and the regime direction gate. Size = framework risk sizing x regime multiplier x `clamp(|signalScore| x 2, 0, 1)`, then notional/gross caps and `makeSize`.

4. Entry = MARKET; fill -> stop-market at `stopAtrMultiplier x ATR` + take-profit close-limit at `tpAtr-Multiplier x ATR`.
5. Exits: signal decay confirmed over `exitConfirmSeconds`, max hold, stale BBO flatten, session flatten, daily loss / equity / order-API kills. Any close that flattens the book cancels every still-working close sibling.

## 34.3 Features and plan adaptations

The plan §2 table is aspirational in two places; the implemented formulas are the closest equivalents the framework carries:

| # | Feature | Implementation | Adaptation |
|---|---|---|---|
| 0 | spread bps | `(ask-bid)/mid x 10000` from the newest BBO | — |
| 1 | depth asymmetry | top-5 `marketdataInfo.book` size sums | — |
| 2/3 | cvd5 / cvd20 | `sum(decSizeCvd)/sum(decSize)` per-bar buckets | — |
| 4/5 | ret1 / ret5 | BBO mid recorded at each bar close (trade close fallback) | "micro-price" is the BBO mid at the bar close; no synthetic micro-price formula exists in the framework |
| 6/7 | vol5 / vol20 | newest ATR bps / 20-bar ATR-mean bps | the plan's vol5 and vol20 are the same ATR-bps expression; vol20 is made the 20-bar ATR mean so the ratio is meaningful |
| 8 | volRatio | vol5 / vol20 | — |
| 9 | wick bias | `(lowWick - highWick) / (high - low)` | — |
| 10 | vol z-score | per-bar trade size, population moments over 20 bars | "volume" is the trade-tape size sum, not the candle volume (identical for trade-built candles) |
| 11 | entropy | 20-change sliding window, up/down/flat class counts, normalized Shannon by log2(3) | the plan's exponential decay is dropped for a plain sliding window (deterministic, auditable); the log-return sign equals the mid-delta sign, so no log is evaluated |
| 12 | survival diff | `S_ask - S_bid` from the pair | — |
| 13 | depth change | `dasym0 - dasym1` | — |
| 14 | aggressive flow | `sum(cvd_i x sign(ret_i)) / sum(size_i)` over 10 bars | `ret_i` is bar i's own close-to-close return |
| 15 | time in bar | elapsed fraction of the live working bar at capture | measured on the working bar so it exposes bar-close processing lag; 1.0 when no working bar exists |

Model input (plan §3): `[spr, dasym, cvd5, cvd20, vol5, vol20, volZ, timeInBar]`. Entropy reports the maximum (1.0, i.e. zero weight) until the 20-change window is full, so an unwarmed entropy can never open the gate.

## 34.4 Model

- Two bounded online logistic regressions (bid survival, ask survival). Label = the taker side of the **first trade of the next bar**: taker sell -> bid label 1, taker buy -> ask label 1. A capture whose bar saw no trade is dropped and counted.
- Bounded SGD: `w += learnRate x (err x x - l2 x w)`, weights clipped to `+/-weightClip` (default 20), bias clipped to +/-4.
- Streaming Welford mean/M2 normalization per input; the 1e-8 variance floor keeps untouched features at near-zero contribution.
- State (both models + entropy window + counters) persists as JSON; the restore is validate-then-commit. A corrupt document restarts fresh and the entry lock (`minTrainBars`) keeps trading disabled until it relearns. The file is written on trade close, halt and clean exit, plus a periodic checkpoint (first heartbeat, then every 5 min), so warm-up progress, the daily session counters (trade count, loss latch, session index) and the post-trade cooldown survive a kill or crash.

## 34.5 Regimes (plan §3 table, exact)

| State | Condition | Size mult | Entry gate | Max trades/day | Directions |
|---|---|---|---|---|---|
| LOW_VOL_TREND | volRatio < 0.8, entropy < 0.6 | 1.00 | 0.62 | 20 | both |
| LOW_VOL_RANGE | volRatio < 0.8, entropy >= 0.6 | 0.75 | 0.60 | 15 | both |
| HIGH_VOL_BREAKOUT | volRatio >= 1.5, entropy < 0.6 | 0.50 | 0.65 | 10 | trend-only |
| HIGH_VOL_CHOP | volRatio >= 1.5, entropy >= 0.6 | 0.25 | 0.75 | 5 | none |

- State changes need `regimeHysteresisBars` consecutive qualifying bars; the band between `regimeVolLow` and `regimeVolHigh` holds the current state.
- Trend direction (trend-only gate) comes from cvd20 when `|cvd20| >= regimeCvdTrendMin`, else from depth asymmetry when `|dasym| >= regimeDasymTrendMin`, else none -> no breakout entry.
- The detector never consumes the survival models (no circularity).
- The regime's max-trades/day combines with `dailyMaxTrades` as the stricter of the two.

## 34.6 Exits and risk controls

- Hard stop: native stop-market at `stopAtrMultiplier x ATR` (placed on fill).
- Take-profit: close-limit at `tpAtrMultiplier x ATR` (distinct order user-id family, so stop/tp never collide in the manager's find-by-user helpers).
- Signal decay: long exits when `score < -exitThreshold`, short when `score > +exitThreshold`, confirmed for `exitConfirmSeconds`.
- Max hold: `maxHoldMs`; session flatten in daily mode; stale-BBO flatten.
- Hard caps: `maxNotionalPerTrade`, `maxGrossLeverage`, `dailyMaxLossPct` (fraction of equity), `dailyMaxTrades`, one position per instrument (structural), post-trade cooldown `cooldownMs`, spread kill switch `maxEntrySpreadBps`, vol percentile entry filter `maxVolPercentile`, anomaly freeze above `anomalyVolPct` (99th percentile default), equity drawdown halt.
- Never a naked position: a lost protection leg is re-placed; a failed market close reverts to the in-position state and retries; a flat book cancels every working close order (multi-algo runs clear the venue reduce-only flag, so an orphan close leg could open a reverse position).

## 34.7 Parameters

All keys live in the algo `parameter` object; missing keys keep the defaults, out-of-range values are clamped. See `config/algoQIS-lighter-btc-perp.json` for the shipped values.

| Key | Default | Range | Meaning |
|---|---|---|---|
| `instTradeId`, `ohlcId`, `atrIndicatorId` | required | config ids | identity binding, fails closed |
| `ohlcTimeFrameMin` | 5 | 1-1440 | signal bar; must match the bound OHLC feed |
| `risk` | `PERCENT 0.20` | framework | consumed before `onSetParams` |
| `sessionMode` | `continuous` | continuous/daily | flatten discipline |
| `flattenMinuteUtc` | 1320 | 0-1439 | daily flatten |
| `noEntryMinutes` | 15 | 0-1440 | no late risk |
| `dailyMaxTrades` | 10 | 1-1000 | session hard cap (regime cap may be stricter) |
| `dailyMaxLossPct` | 0.02 | 0-0.1 | daily loss kill, FRACTION of equity |
| `signalThreshold` | 0.35 | 0.10-0.90 | minimum `\|signalScore\|` floor |
| `exitThreshold` | 0.40 | 0.10-0.90 | signal-decay gate |
| `exitConfirmSeconds` | 20 | 1-300 | decay confirmation |
| `minTrainBars` | 150 | 10-5000 | model readiness / corrupt-state lock |
| `learnRate` / `l2` / `weightClip` | 0.05 / 0.0001 / 20 | see config | SGD envelope |
| `stopAtrMultiplier` / `tpAtrMultiplier` | 1.5 / 2.5 | 0.3-5 / 0.5-8 | ATR protection geometry |
| `maxHoldMs` | 1200000 | 60 s-6 h | time stop |
| `cooldownMs` | 10000 | 1-600 s | post-trade cooldown |
| `maxEntrySpreadBps` | 15 | 1-200 | spread entry block |
| `maxVolPercentile` | 0.90 | 0.5-0.995 | vol-percentile entry filter |
| `anomalyVolPct` | 0.99 | 0.9-1.0, > filter | anomaly freeze threshold |
| `anomalyFreezeBars` | 5 | 1-60 | freeze length after an anomaly |
| `bboStaleSeconds` / `bboFlattenSeconds` | 5 / 30 | see config | entry block / position flatten |
| `maxNotionalPerTrade` / `maxGrossLeverage` | 500 / 1.0 | see config | hard caps (applied last) |
| `regimeHysteresisBars` | 3 | 1-20 | consecutive bars to change regime |
| `regimeVolLow` / `regimeVolHigh` | 0.8 / 1.5 | see config | vol band boundaries |
| `regimeEntropyTrend` | 0.6 | 0.1-0.95 | trend/range entropy split |
| `regimeCvdTrendMin` / `regimeDasymTrendMin` | 0.3 / 0.5 | see config | trend-direction confirmation |
| `reduceOnlyClose` | true | bool | venue reduce-only flag on close orders |

| Key | Default | Range | Meaning |
|---|---|---|---|
| `stateFile` | — | path | JSON persistence path |

Plan-to-config mapping: the plan's `entryThreshold`/`exitThreshold` pair collapse into `signalThreshold` (entry floor) and `exitThreshold` (decay), and `risk.value 0.20` maps to the framework `PERCENT 0.20` risk block. The plan's %-of-equity notional caps are inconsistent with its own 0.20 %-risk / 1.5xATR sizing; the implementation uses the algoDELEV-style absolute notional plus gross-leverage caps instead.

## 34.8 Run

```
ttTrader.exe algos/algoQIS/config/algoQIS-lighter-btc-perp.json --log --logmarketdata
ttTrader.exe algos/algoQIS/config/algoQIS-lighter-btc-perp.json --log --tradingMode paper --logmarketdata
```

Build (from the repo root with the VS dev shell):

```
. "C:\Program Files\Microsoft Visual Studio\18\Community\Common7\Tools\Launch-VsDevShell.ps1" -Arch amd64 -HostArch amd64 -SkipAutomaticLocation
cmake --build .build/clang-debug --target algoQIS
```

Unit tests:

```
cmake --build .build/clang-debug --target ttManagerOrderTests
ttManagerOrderTests.exe Qis
```

## 34.9 LLM layer rejected

The plan's optional local LLM advisor is deliberately **not implemented**:

- there is no text feed in this framework; running a quantized LLM on a feature vector is an unvalidated, expensive non-sequitur;
- token sampling is nondeterministic, which breaks the strategy's auditability invariant (identical feed sequences must produce identical state and orders);
- a background inference thread contradicts the no-locks / no-background- thread HFT design of the algo host.

The deterministic logistic pair + entropy + regime stack is the whole decision layer; nothing in the order path can block on inference.

## 34.10 Invariants

- All heavy state is fixed-size `std::array`; allocation-free after init; `decimal_t` everywhere except inside the Eigen / entropy double boundary.
- No RNG; identical feed sequences produce identical state and orders.
- Indicator/OHLC reads use the V2 exact-sequence API (`findOhlc`, `findIndicator`, `getValueAtSequence`); labels/features never read by offset. `getIndicatorValue()` is not used.
- One instrument per algo instance; positions are keyed by the owning algo's config id inside the order manager, so several algo instances can run the same instrument without position leakage. `ui64PositionUser` is never set.
- `reduceOnlyClose` defaults to `true`; a combined multi-algo config sets it `false` and relies on sibling-order hygiene (every close/flat cancels the working close orders) to avoid orphan legs.
- A corrupt state file falls back to a fresh pair; entries stay locked until `minTrainBars` labels mature again.
- Entries are market-only (no post-only path); protection is placed on the entry fill and restored after a lost leg.

## 34.11 Open validation gates

- **Unvalidated design targets**: no walk-forward, paper or cert run has validated this strategy. The plan §1 return/Sharpe figures are internally inconsistent and are NOT design claims; conservative, unvalidated targets are a 0-15 % annualized net edge with 1.0-1.5 profit factor, to be confirmed (or falsified) only by the gates below.
- Walk-forward protocol (plan §6): purged k-fold, embargo, deflated Sharpe, White's reality check — not yet executed.
- Paper trading on Lighter testnet (`tradingMode paper`) for the plan's 4-week minimum — not yet executed.
- Deterministic end-to-end replay certification requires `.ttpb` playback for BBO/book/trade events; not available yet.
- The entropy window and the label horizon (next trade) are design choices that need sensitivity checks; no parameter has been optimized.
- Lighter book feed depth must be verified to carry 5 usable levels before trusting `dasym` live.

# Chapter 35

# algoRANGE

adaptive band-fade mean reversion with online parameter feedback. algoRANGE fades the outer zones of a validated rolling close-band back toward the middle: entries are post-only limits in the outer zone gated by an online logistic reversion model, exits are a limit target inside the band plus a reduce-only stop-market beyond the wick extreme. the learned components (counterfactual parameter memory, nn controller, win-rate tracker) feed learned values back into the algo's own parameters within hard envelopes — fully online, no offline training, no external tooling.

plan: `.kilo/plans/1789187480641-algoRANGE-band-fade-plan.md`

## 35.1 strategy

- band: min/max of closed 5m bar closes over `bandLookbackBars` (wick extremes tracked separately as the stop anchor). a band is tradeable when the width sits in its bps corridor, both edges were touched often enough, the drift stays small and it held stable for `bandStableBars` closes.
- entry: post-only limit at `entryDepthFrac` of the band width from the near edge, armed only while price approaches within `armFrac` of that level, gated by the reversion probability `p >= pThreshold` (boosted in COMPRESSED, nudged by the nn controller).
- exit: target at `targetFrac` of the distance toward the opposite edge, stop beyond the wick extreme by `max(stopAtrMult x atr1m, stopMinBps)`, max-hold budget with a single extension while the mark beats entry + cost.
- regime protection: two-strike breakout protocol (soft break pulls entries, confirmed break flattens and invalidates the band for `reestablishMinBars`), COMPRESSED (squeeze) shrinks size and lifts the threshold, STRESS (vol/spread shock) blocks and flattens, kill switches on stale feed, spread anomaly, daily loss and drawdown.

## 35.2 learned components (all online)

| component | learns | feeds back |
|---|---|---|
| `signalModel` | logistic P(fade recedes to target before stop), first-touch labels from every armed decision (would-have-been supervision, persisted pending ring) | entry gate + confidence multiplier |

| component | learns | feeds back |
|---|---|---|
| paramMemory | per-condition counterfactual optima of targetFrac / stopAtr / holdBars via grid re-simulation of the recorded path (hunt precedent) | overrides config base x nn delta when a bucket is statistically relevant |
| nnController | 17->24 tanh policy head trained by bounded reinforce on realized R (hunt nnGate precedent) | multiplicative deltas for targetFrac / stopAtr / size / threshold |
| watchdog | rolling 20-trade win rate | suspends nn deltas + memory adoption below ctrlSuspendWinRate |
| win-rate ema | closed-trade win rate | size multiplier [winRateMinMultiplier, winRateMaxMultiplier] |

precedence: counterfactual memory > nn controller > config base; the sizing caps always apply after every multiplier (caps only shrink).

## 35.3 parameter table (config key, default, range)

| key | default | range |
|---|---|---|
| bandLookbackBars | 48 (5m) | 12–192 |
| bandStableBars | 6 | 3–48 |
| minEdgeTouches | 2 | 1–10 |
| edgeZoneFrac | 0.20 | 0.05–0.5 |
| minBandWidthBps / maxBandWidthBps | 8 / 300 | 1–1000 / 10–5000 |
| maxDriftZ | 0.8 | 0.1–3.0 |
| entryDepthFrac | 0.15 | 0.05–0.4 |
| armFrac / disarmFrac | 0.20 / 0.30 | arm <= disarm, <= 1.0 |
| targetFrac | 0.40 | 0.2–1.0 |
| stopAtrMult / stopMinBps | 1.0 / 6 | 0.1–3.0 / 1–50 |
| maxHoldBars | 24 (1m) | 5–120 |
| entryTimeoutMs / armCooldownMs / cooldownMs | 120000 / 60000 / 180000 | ms clamps |
| pThreshold | 0.60 | 0.55–0.9 |
| minTrainLabels | 60 | 10–2000 |
| learnRate / l2 / performanceEmaAlpha | 0.03 / 0.0001 / 0.1 | clamped |
| labelTimeoutBars / labelProgressFrac | 60 / 0.25 | 10–240 / 0.05–0.9 |
| nnEnabled / nnExploreTrades | true / 40 | – / 10–100000 |
| nnLearnRate / nnL2 | 0.01 / 0.0001 | clamped |
| ctrlSuspendWinRate / ctrlResumeWinRate | 0.40 / 0.48 | 0.2–0.6 / >= suspend |

| key | default | range |
|---|---|---|
| cfMemoryEnabled / cfMinBucketSamples / cfModalShare | true / 30 / 0.40 | – / 10–200 / 0.3–0.9 |
| regimeHysteresisBars | 5 (1m) | 1–60 |
| breakConfirmAtr / breakSoftVolPct / breakWatchBars | 0.5 / 0.90 / 12 | clamped |
| reestablishMinBars | 12 (5m) | 0–288 |
| compressedWidthPct / compressedSizeMult / compressedThresh-oldBoost | 0.20 / 0.6 / 0.03 | clamped |
| stressVolPct / stressSpreadZ | 0.97 / 2.5 | clamped |
| maxEntrySpreadBps / staleBboMs | 2.5 / 5000 | clamped |
| spreadKillZ / spreadKillMs / ddKillPct | 3.0 / 10000 / 0.015 | clamped |
| roundTurnCostBps / minEdgeBufferBps | 2 / 2 | 0–100 |
| productCap | 1.5 | 0.1–3.0 |
| winRateMinMultiplier / winRateMaxMulti-plier | 0.5 / 1.25 | clamped |
| regimeMinTrades | 20 | 1–500 |
| maxPositionContracts / maxNotionalUsd | 10 / 50000 | caps only shrink |
| sessionMode | continuous | continuous/daily |
| dailyMaxTrades / dailyMaxLossPct | 30 / 0.02 | clamped |
| lossStreakTrades / lossStreak-CooldownMs | 3 / 900000 | clamped |
| fundingMinutesUtc / fundingSkirtMinutes | hourly / 3 | <= 32 entries / 0–30 |
| reduceOnlyClose | true | – |

## 35.4 state file

state/algoRANGE-BTCUSD-PERP-state.json (atomic tmp+rename, validate- then-commit): model (weights as ieee-754 bit patterns, welford stats, drift monitor, win-rate ema, persisted pending label ring), memory (sparse counterfactual buckets), nn (bit-exact weights, welford, rng, baseline), regime, band (ring so a restart does not re-wait a full lookback), session, watchdog, cooldowns, halt latch, high-water equity. written on exit, every closed trade, and a 300 s heartbeat checkpoint.

## 35.5 files

| file | contents |
|---|---|
| algoConfig.h/.cpp | parameter set + clamped json parse |

| file | contents |
|---|---|
| algoCore.h/.cpp | plugin identity, state machine, arming, bracket, exits, kill switches, persistence |
| bandTracker.h/.cpp | rolling close band, wick extremes, quality gates, establishment |
| featureBuilder.h/.cpp | 15 raw features from candles/tape/bbo + the side orientation to the 18-dim model input |
| signalModel.h/.cpp | eigen online logistic + first-touch pending ring + drift monitor |
| paramMemory.h/.cpp | counterfactual grid re-simulation + condition buckets (hunt precedent) |
| nnController.h/.cpp | reinforce policy fine-tuner (hunt nnGate precedent) |
| regimeDetector.h/.cpp | 4-state regime + two-strike breakout protocol |
| positionSizer.h/.cpp | sizing stack, fee-aware edge gate (pure helpers) |
| sessionGuard.h/.cpp | continuous/daily session guard (mld/mire precedent) |
| watchdog.h/.cpp | rolling win-rate suspension latch |
| rangeMath.h/.cpp | linreg slope + circular minute distance |
| config/algoRANGE-lighter-btc-perp.json | run config (simulation/paper) |

## 35.6 run

```
cmake --preset msvc-debug && cmake --build .build/msvc-debug --target algoRANGE
w:\ttTraderDebug\ttTrader.exe w:\development\ttTrader\algos\algoRANGE\config\algoRANGE-lighter-btc-perp.json --tradingMode simulation
```

## 35.7 kpi gates (before any size-up)

- = 50 matured labels before the gate model trades; calibration tracked.
- counterfactual buckets act only at >= 30 samples and >= 0.40 modal share.
- rolling window over closed fades: net expectancy > 0 after costs, pf > 1.15, max dd <= 4%, win rate >= the breakeven implied by the realized r:r (`L / (W + L)` + cost drag). failing gates suspend learned parameters; they never raise size.

# Part V

# Part V - Indicators

# Chapter 36

# Technical Indicators

The active indicator architecture is Native Indicator V2. Each indicator is built as an independent DLL, loaded only when referenced by an enabled candle-series configuration, and accessed through cached enum/function-table bindings.

The authoritative implementation, configuration, formula, HFT lifecycle, and extension guide is:

- `../../indicators/README.md`

Key properties:

- Finalized event-time candles are the input contract.
- One V2 DLL is produced per indicator.
- Unused indicators are not loaded and allocate no runtime state.
- Configuration strings are parsed during startup only.
- Runtime output selection uses `indicatorOutput_e`, not string comparisons.
- Indicator instances are opaque to `ttTrader` and destroyed by their owning DLL.
- Rolling state and output history are bounded and preallocated.
- Financial calculations use `decimal_t`.

Available plugins:

| Indicator | DLL | Configuration type |
|---|---|---|
| EMA | `indEMA` | `ema` |
| SMA | `indSMA` | `sma` |
| WMA | `indWMA` | `wma` |
| HMA | `indHMA` | `hma` |
| Bollinger Bands | `indBollingerBands` | `bollinger` |
| ATR | `indATR` | `atr` |
| RSI | `indRSI` | `rsi` |
| MACD | `indMACD` | `macd` |
| Rate of Change | `indROC` | `roc` |
| Stochastic | `indStochastic` | `stochastic` |
| On-Balance Volume | `indOBV` | `obv` |
| Parabolic SAR | `indPSAR` | `psar` |
| Ichimoku Cloud | `indIchimoku` | `ichimoku` |
| Commodity Channel Index | `indCCI` | `cci` |
| Money Flow Index | `indMFI` | `mfi` |
| Keltner Channels | `indKeltnerChannels` | `keltner` |
| Donchian Channels | `indDonchianChannels` | `donchian` |
| Standard Deviation | `indStdDev` | `stddev` |
| VWAP | `indVWAP` | `vwap` |

| Indicator | DLL | Configuration type |
|---|---|---|
| Chaikin Money Flow | indCMF | cmf |
| Volume Profile | indVolumeProfile | volumeProfile |
| Accumulation/Distribution Line | indAD | ad |
| Linear Regression Channel | indLinearRegressionChannel | linreg |
| Z-Score | indZScore | zscore |
| Relative Volume | indRVOL | rvol |
| Choppiness Index | indChoppiness | chop |
| Elder Ray | indElderRay | elderRay |
| ADX / DMI | indADX | adx |
| Supertrend | indSupertrend | supertrend |
| Volume Weighted Moving Average | indVWMA | vwma |
| Fibonacci Retracement | indFibonacci | fibonacci |
| Pivot Points | indPivotPoints | pivotPoints |
| Williams %R | indWilliamsR | williamsR |
| Stochastic RSI | indStochRsi | stochRsi |
| Williams Alligator | indAlligator | alligator |
| TRIX | indTRIX | trix |
| KST (Know Sure Thing) | indKST | kst |
| Chaikin Oscillator | indChaikinOscillator | chaikinOscillator |
| Ultimate Oscillator | indUltimateOscillator | ultimateOscillator |
| Vortex Indicator | indVortex | vortex |
| Ease of Movement | indEOM | eom |
| Elder Force Index | indForceIndex | forceIndex |
| Percentage Price Oscillator | indPPO | ppo |
| Momentum | indMomentum | momentum |
| ATR Channels | indAtrChannels | atrChannels |
| TRIN / Arms Index | indTRIN | trin |
| NVI / PVI | indVolumeIndex | volumeIndex |

Multi-output indicators publish dedicated output enums: Bollinger Bands (`EIO_BOLLINGER_*`), MACD (`EIO_MACD*`), Stochastic (`EIO_STOCHASTIC_*`), Ichimoku (`EIO_ICHIMOKU_*`), Keltner (`EIO_KELTNER_*`), Donchian (`EIO_DONCHIAN_*`), Volume Profile (`EIO_VOLUMEPROFILE_POC/VAH/VAL`), Linear Regression Channel (`EIO_LINEARREG_*`), Elder Ray (`EIO_ELDER_*`), ADX/DMI (`EIO_ADX`, `EIO_DMI_*`), Supertrend (`EIO_SUPERTREND`, `EIO_SUPERTREND_DIRECTION`), Fibonacci Retracement (`EIO_FIB_*`), Pivot Points (`EIO_PIVOT`, `EIO_PIVOT_R1..R3`, `EIO_PIVOT_S1..S3`), Stochastic RSI (`EIO_STOCHRSI_*`), Alligator (`EIO_ALLIGATOR_*`), KST (`EIO_KST`, `EIO_KST_SIGNAL`), Vortex (`EIO_VORTEX_PLUS/MINUS`), PPO (`EIO_PPO*`), ATR Channels (`EIO_ATRCHANNEL_*`), and NVI/PVI (`EIO_NVI`, `EIO_PVI`). All remaining indicators publish `EIO_VALUE`.

Example:

```
{
  "id": "instBTC-5m",
  "source": "close",
  "timeFrame": 5,
  "historyDepth": 2048,
  "indicators": [
    {"id":"ema20","type":"ema","params":{"period":20,"input":"close"}},
    {"id":"atr14","type":"atr","params":{"period":14}},
    {"id":"macd","type":"macd","params":{"fastPeriod":12,"slowPeriod":26,"signalPeriod":9}},
    {"id":"keltner","type":"keltner","params":{"period":20,"atrPeriod":10,"multiplier":2}},
    {"id":"vprofile","type":"volumeProfile","params":{"period":100,"bins":24}}
  ]
}
```

Algo access uses enums:

```
void algoCore_c::onFinalizedCandle(
    instrumentInfo_s* p_instrumentPtr,
    ohlcInfo_s* p_ohlcPtr,
    const candle_s& p_candle)
{
    const indicatorValue_s l_atr =
        p_ohlcPtr->getIndicatorValue(m_atrId, EIO_VALUE, 0);

    const indicatorValue_s l_histogram =
        p_ohlcPtr->getIndicatorValue(m_macdId, EIO_MACD_HISTOGRAM, 0);

    const indicatorValue_s l_keltnerUpper =
        p_ohlcPtr->getIndicatorValue(m_keltnerId, EIO_KELTNER_UPPER, 0);

    if (!l_atr.isValid() || !l_histogram.isValid() || !l_keltnerUpper.isValid())
    {
        return;
    }
}
```

Per-indicator configuration, formulas, warm-up behavior, and certification status are documented in the authoritative README above; runtime certification runs offline via the per-folder `algoIndicatorTest-static.cfg` against `exchgStaticIndicator`.

The removed V1 `indicatorCore_c`, positional data buffers, `indAvgEMA`, `indAvgHMA`, and old `createPlugin` indicator interface are not supported.

# Chapter 37

# Native Indicators V2

This directory contains ttTrader's candle-driven native indicator runtime and built-in technical indicators.

The active implementation is a greenfield replacement for the removed `indicatorCore_c`, `indAvgEMA`, and `indAvgHMA` interface. Each indicator is built as an independent V2 DLL. `ttTrader` loads only indicator types referenced by enabled candle-series configuration, caches their function tables, instantiates them per series, initializes them from historical finalized candles, and updates them exactly once for each newly finalized candle.

Configuration strings are parsed only during startup. Runtime selection uses `indicatorType_e`, `indicatorOutput_e`, cached function pointers, and opaque instance handles. Finalized-candle processing does not compare indicator or output strings.

## 37.1 Architecture

The data flow is:


Important source files:

- `include/ttTrader/indicator/indicatorPluginV2.h`: public V2 types and contracts.

- `indicators/nativeIndicatorBase.h`: bounded output history, runtime-capacity circular windows, allocation lifecycle, input extraction, and shared EMA accumulator.
- `indicators/indicatorDllApi.h`: common opaque V2 DLL function-table adapter.
- `indicators/<indicator>/indicatorDll.cpp`: one exported `ttGetIndicatorApiV2` entry point per DLL.
- `src/storeIndicator.cpp`: parses indicator declarations and owns configuration rollback.
- `include/ttTrader/general/indicatorInfo.h`: one configured indicator identity and instance.
- `include/ttTrader/general/ohlcInfo.h`: per-series dispatch and value lookup.
- `indicators/<indicator>/indicatorDefinition.h`: descriptor, output schema, and class declaration.
- `indicators/<indicator>/indicator.cpp`: formula and state implementation.

## 37.2 HFT Properties

The native runtime follows the ttTrader HFT constraints:

- All financial calculations use `decimal_t`.
- Indicator instances and metadata use `st_malloc` and `st_free`.
- Output history and rolling windows are allocated once during initialization.
- Finalized-candle update paths perform no dynamic allocation.
- Runtime-sized windows use circular overwrite semantics.
- Indicator output history is bounded by the owning candle series `historyDepth`.
- Indicator values and candle windows can be resolved by finalized sequence for queued catch-up events.
- Duplicate and decreasing finalized sequences cannot advance indicator state.
- Invalid or unavailable input produces an explicit non-valid result instead of an implicit zero signal.

The project `circularBuffer_s<T, N>` remains appropriate where capacity is known at compile time. Indicator periods are runtime configuration values, so native indicators use `preallocatedWindow_c<T>` with equivalent chronological circular behavior.

## 37.3 Lifecycle

1. `storeOhlc_c` parses an OHLC/candle series.
2. `storeIndicator_c` parses the series' `indicators` array.
3. `createNativeIndicatorInstance()` allocates the concrete indicator.
4. The indicator validates and allocates its bounded state during `configure()`.
5. Historical finalized candles initialize the indicator in chronological order.
6. Every new finalized candle invokes `onCandleClose()` once.
7. Algos read outputs by configured indicator ID and `indicatorOutput_e`.
8. `indicatorInstance_c::release()` runs the concrete destructor and releases project-managed storage.

Indicators are calculated before the finalized-candle event is published to algos. Consequently, output offset `0` corresponds to the candle delivered by the current `onFinalizedCandle()` callback.

## 37.4 Configuration

Indicators are declared inside an instrument's OHLC entry:

```
{
  "id": "instBTC",
  "exchgId": "exchgLighter",
  "idAtExchg": "BTC",
  "ohlc": [
    {
      "id": "instBTC-5m",
      "source": "close",
```

```
      "timeFrame": 5,
      "historyDepth": 1024,
      "indicators": [
        {
          "id": "instBTC-5m-ema20",
          "type": "ema",
          "params": {
            "period": 20,
            "input": "close"
          }
        },
        {
          "id": "instBTC-5m-rsi14",
          "type": "rsi",
          "params": {
            "period": 14,
            "input": "hlc3"
          }
        },
        {
          "id": "instBTC-5m-macd",
          "type": "macd",
          "params": {
            "fastPeriod": 12,
            "slowPeriod": 26,
            "signalPeriod": 9,
            "input": "close"
          }
        }
      ]
    }
  ]
}
```

### 37.4.1 Common Fields

| Field | Required | Description |
|---|---|---|
| `id` | Yes | Unique configured indicator identity used by algos. |
| `type` | Yes | Native implementation ID such as `ema`, `atr`, or `macd`. |
| `enabled` | No | Standard ttTrader enable flag. Disabled declarations are skipped. |
| `params` | No | Indicator parameters. `parameter` is also accepted. |

Periods must be unsigned integers in the range `1..65536`. Bollinger Bands requires a period of at least 2, Standard Deviation a period of at least 2, Linear Regression Channel a period of at least 2, and Volume Profile bounds `bins` to `1..1024`. MACD and PPO require `slowPeriod > fastPeriod`, the Chaikin Oscillator requires `slowPeriod > fastPeriod`, and the Ultimate Oscillator requires `period1 < period2 < period3`. Invalid types, negative values, unknown inputs, invalid MACD period ordering, non-positive Parabolic SAR/Keltner/Supertrend/ATR

Channels/Ease of Movement parameters, an unknown `style` for Pivot Points (`classic`, `camarilla`, or `woodie`), and out-of-range Volume Profile percentages reject the configuration.

## 37.5 Candle Inputs

Scalar indicators accept an optional `input` parameter:

| Input | Value |
|---|---|
| `open` | Candle open |
| `high` | Candle high |
| `low` | Candle low |
| `close` | Candle close; default |
| `hl2` | `(high + low) / 2` |
| `hlc3` | `(high + low + close) / 3` |
| `ohlc4` | `(open + high + low + close) / 4` |
| `vwap` | Candle VWAP when its validity bit is present |

Supported by EMA, SMA, WMA, HMA, Bollinger Bands, RSI, MACD, ROC, Standard Deviation, OBV's comparison price, Linear Regression Channel, Z-Score, TRIX, KST, PPO, Momentum, VWMA's price leg, Stochastic RSI's price leg, Alligator, ATR Channels, and NVI/PVI's price leg.

ATR, Stochastic, Parabolic SAR, Ichimoku, Donchian Channels, Keltner Channels, CCI, MFI, VWAP, CMF, Volume Profile, A/D, Relative Volume, Choppiness Index, Elder Ray, ADX/DMI, Supertrend, Fibonacci Retracement, Pivot Points, Williams %R, Chaikin Oscillator, Ultimate Oscillator, Vortex, Ease of Movement, Elder Force Index, and TRIN are full-candle calculations and do not accept a scalar input. The volume-based indicators (OBV, MFI, VWAP, CMF, Volume Profile, A/D, RVOL, VWMA, Chaikin Oscillator, Ease of Movement, Elder Force Index, TRIN, and NVI/PVI) always use candle trade volume.

A `vwap` input remains unavailable when the candle does not have a valid VWAP metric. Configure the candle series to compute VWAP when an indicator requires it.

## 37.6 Value Status

Every lookup returns `indicatorValue_s`:

```
struct indicatorValue_s
{
    indicatorValueType_e eType;
    indicatorValueStatus_e eStatus;
    decimal_t decValue;
    int64_t i64Value;
    bool fValue;
};
```

Statuses:

| Status | Meaning |
|---|---|
| `EIVS_VALID` | The value is ready for trading logic. |
| `EIVS_WARMUP` | The indicator has not received enough valid candles. |
| `EIVS_NOT_FOUND` | Indicator, output, or requested history offset is unavailable. |
| `EIVS_INVALID` | No meaningful result was produced. |

Always check `isValid()` before using `decValue`:

```
const indicatorValue_s l_rsi = p_ohlcPtr->getIndicatorValue(m_rsiId, EIO_VALUE);
if (!l_rsi.isValid())
{
    return;
}

if (l_rsi.decValue < 30_dec)
{
    // strategy-specific oversold handling
}
```

## 37.7 Runtime Output Enums

| Indicator | Output enum |
|---|---|
| EMA, SMA, WMA, HMA, ATR, RSI, ROC, OBV, Parabolic SAR, CCI, MFI, Standard Deviation, VWAP, CMF, A/D | `EIO_VALUE` |
| Bollinger middle | `EIO_BOLLINGER_MIDDLE` |
| Bollinger upper | `EIO_BOLLINGER_UPPER` |
| Bollinger lower | `EIO_BOLLINGER_LOWER` |
| Bollinger width | `EIO_BOLLINGER_WIDTH` |
| MACD line | `EIO_MACD` |
| MACD signal | `EIO_MACD_SIGNAL` |
| MACD histogram | `EIO_MACD_HISTOGRAM` |
| Stochastic %K | `EIO_STOCHASTIC_K` |
| Stochastic %D | `EIO_STOCHASTIC_D` |
| Ichimoku tenkan | `EIO_ICHIMOKU_TENKAN` |
| Ichimoku kijun | `EIO_ICHIMOKU_KIJUN` |
| Ichimoku senkou A | `EIO_ICHIMOKU_SENKOU_A` |
| Ichimoku senkou B | `EIO_ICHIMOKU_SENKOU_B` |
| Keltner middle | `EIO_KELTNER_MIDDLE` |
| Keltner upper | `EIO_KELTNER_UPPER` |
| Keltner lower | `EIO_KELTNER_LOWER` |
| Donchian upper | `EIO_DONCHIAN_UPPER` |
| Donchian middle | `EIO_DONCHIAN_MIDDLE` |
| Donchian lower | `EIO_DONCHIAN_LOWER` |
| Volume Profile POC | `EIO_VOLUMEPROFILE_POC` |
| Volume Profile VAH | `EIO_VOLUMEPROFILE_VAH` |
| Volume Profile VAL | `EIO_VOLUMEPROFILE_VAL` |
| Linear Regression line | `EIO_VALUE` |
| Linear Regression upper | `EIO_LINEARREG_UPPER` |
| Linear Regression lower | `EIO_LINEARREG_LOWER` |
| Linear Regression slope | `EIO_LINEARREG_SLOPE` |
| Elder Ray bull power | `EIO_ELDER_BULL` |
| Elder Ray bear power | `EIO_ELDER_BEAR` |
| ADX | `EIO_ADX` |
| DMI +DI | `EIO_DMI_PLUS` |
| DMI -DI | `EIO_DMI_MINUS` |
| Supertrend level | `EIO_SUPERTREND` |
| Supertrend direction | `EIO_SUPERTREND_DIRECTION` |
| Fibonacci 38.2% | `EIO_FIB_382` |
| Fibonacci 50% | `EIO_FIB_500` |

| Indicator | Output enum |
|---|---|
| Fibonacci 61.8% | `EIO_FIB_618` |
| Fibonacci 78.6% | `EIO_FIB_786` |
| Pivot point | `EIO_PIVOT` |
| Pivot resistance 1..3 | `EIO_PIVOT_R1`, `EIO_PIVOT_R2`, `EIO_PIVOT_R3` |
| Pivot support 1..3 | `EIO_PIVOT_S1`, `EIO_PIVOT_S2`, `EIO_PIVOT_S3` |
| Stochastic RSI %K | `EIO_STOCHRSI_K` |
| Stochastic RSI %D | `EIO_STOCHRSI_D` |
| Alligator jaw | `EIO_ALLIGATOR_JAW` |
| Alligator teeth | `EIO_ALLIGATOR_TEETH` |
| Alligator lips | `EIO_ALLIGATOR_LIPS` |
| KST | `EIO_KST` |
| KST signal | `EIO_KST_SIGNAL` |
| Vortex VI+ | `EIO_VORTEX_PLUS` |
| Vortex VI- | `EIO_VORTEX_MINUS` |
| PPO line | `EIO_PPO` |
| PPO signal | `EIO_PPO_SIGNAL` |
| PPO histogram | `EIO_PPO_HISTOGRAM` |
| ATR Channels middle | `EIO_ATRCHANNEL_MIDDLE` |
| ATR Channels upper | `EIO_ATRCHANNEL_UPPER` |
| ATR Channels lower | `EIO_ATRCHANNEL_LOWER` |
| Negative Volume Index | `EIO_NVI` |
| Positive Volume Index | `EIO_PVI` |

Descriptor strings such as `"histogram"` are metadata for configuration tools and diagnostics. They are not runtime selectors.

## 37.8 Algo Access

### 37.8.1 From A Candle Callback

```
void algoCore_c::onFinalizedCandle(
    instrumentInfo_s* p_instrumentPtr,
    ohlcInfo_s* p_ohlcPtr,
    const candle_s& p_candle)
{
    if (p_ohlcPtr->ohlcConfigInfo.configInfo != m_ohlcId)
    {
        return;
    }

    const indicatorValue_s l_fast =
        p_ohlcPtr->getIndicatorValue(m_fastEmaId, EIO_VALUE, 0);
    const indicatorValue_s l_slow =
        p_ohlcPtr->getIndicatorValue(m_slowEmaId, EIO_VALUE, 0);

    if (!l_fast.isValid() || !l_slow.isValid())
    {
        return;
    }

    if (l_fast.decValue > l_slow.decValue)
    {
```

```
        // strategy-specific signal handling
    }
}
```

### 37.8.2 Through The Framework Helper

```
const indicatorValue_s l_histogram = getIndicatorValue(
    m_instrumentId,
    m_ohlcId,
    m_macdId,
    EIO_MACD_HISTOGRAM,
    0);

if (!l_histogram.isValid())
{
    return;
}
```

The `offset` is a publication-order offset:

- `0`: newest finalized indicator output
- 1: previous finalized output
- 2: two finalized outputs ago

Offsets do not assume consecutive wall-clock candle sequences, so sparse series retain correct history semantics.

For queued callbacks that may run after newer candles have finalized, use sequence-relative access. `algoMLD` binds each callback to `p_candle.base.time.ui64Sequence`, obtains the exact candle window ending at that sequence, and reads ATR from `getValueAtSequence()`. This prevents catch-up batches from processing the newest candle repeatedly.

## 37.9 ABI And Module Lifetime

Every DLL publishes `indicatorApiV2_s` through `ttGetIndicatorApiV2(2)`. The host validates:

- API version
- Function-table size
- Indicator type enum
- Required function pointers
- Descriptor type and output schema
- ABI fingerprint derived from the shared financial/candle/config value layouts

The fingerprint intentionally enforces the repository's controlled compiler, CRT, packing, ttSubSystem, and shared-header build contract. A DLL from a different build is rejected during startup.

`storeIndicator_c` tracks live instances for each loaded module. Modules are not unloaded while instances remain. Normal shutdown destroys OHLC-owned instances first and unloads their DLLs afterward.

## 37.10 Built-In Indicators

### 37.10.1 EMA

Directory: `indEMA`

Type: `ema`

Parameters:

- `period`: smoothing period, default 14
- `input`: scalar candle input, default `close`

Output:

- `value`: exponential moving average

Initialization uses the simple average of the first `period` values. Subsequent values use:

```
EMA = previousEMA + 2 / (period + 1) * (input - previousEMA)
```

Example:

```
{"id":"ema20","type":"ema","params":{"period":20,"input":"close"}}
```

### 37.10.2 SMA

Directory: `indSMA`

Type: `sma`

Parameters:

- `period`: rolling window, default 14
- `input`: scalar candle input

Output:

- `value`: arithmetic mean of the latest `period` values

Uses a circular window and rolling sum.

```
{"id":"sma50","type":"sma","params":{"period":50,"input":"hlc3"}}
```

### 37.10.3 WMA

Directory: `indWMA`

Type: `wma`

Parameters:

- `period`: rolling window, default 14
- `input`: scalar candle input

Output:

- `value`: linearly weighted moving average

The oldest sample has weight 1 and the newest has weight `period`.

```
{"id":"wma20","type":"wma","params":{"period":20,"input":"close"}}
```

### 37.10.4 HMA

Directory: `indHMA`

Type: `hma`

Parameters:

- `period`: Hull period, default 14
- `input`: scalar candle input

Output:

- `value`: Hull moving average

Formula:

```
HMA(n) = WMA(2 * WMA(input, n / 2) - WMA(input, n), round(sqrt(n)))
```

The square-root length uses nearest-integer rounding.

```
{"id":"hma21","type":"hma","params":{"period":21,"input":"close"}}
```

### 37.10.5 Bollinger Bands

Directory: `indBollingerBands`

Types: `bollinger` or `bollingerBands`

Parameters:

- `period`: rolling window, default 20, minimum 2
- `deviation`: positive standard-deviation multiplier, default 2
- `input`: scalar candle input

Outputs:

- `middle`: rolling SMA
- `upper`: `middle + deviation * populationStdDev`
- `lower`: `middle - deviation * populationStdDev`
- `width`: `(upper - lower) / middle`

```
{
  "id": "bands20",
  "type": "bollinger",
  "params": {
    "period": 20,
    "deviation": 2,
    "input": "close"
  }
}
```

### 37.10.6 ATR

Directory: `indATR`

Type: `atr`

Parameters:

- `period`: Wilder smoothing period, default 14

Output:

- `value`: average true range

True range is the maximum of:

- `high - low`
- `abs(high - previousClose)`
- `abs(low - previousClose)`

The first ATR is the mean of the initial true ranges. Later values use Wilder smoothing.

```
{"id":"atr14","type":"atr","params":{"period":14}}
```

algoMLD configurations use this native output for volatility normalization, position sizing, stops, and take-profit distances.

### 37.10.7 RSI

Directory: `indRSI`

Type: `rsi`

Parameters:

- `period`: Wilder period, default 14
- `input`: scalar candle input

Output:

- `value`: RSI in the range `0..100`

Average gains and losses use Wilder smoothing. A fully rising window produces 100 when average loss is zero.

```
{"id":"rsi14","type":"rsi","params":{"period":14,"input":"close"}}
```

### 37.10.8 MACD

Directory: `indMACD`

Type: `macd`

Parameters:

- `fastPeriod`: fast EMA period, default 12
- `slowPeriod`: slow EMA period, default 26
- `signalPeriod`: signal EMA period, default 9
- `input`: scalar candle input

The slow period must be greater than the fast period.

Outputs:

- `macd`: fast EMA minus slow EMA
- `signal`: EMA of the MACD line
- `histogram`: MACD minus signal

```
{
  "id": "macd-12-26-9",
  "type": "macd",
  "params": {
    "fastPeriod": 12,
    "slowPeriod": 26,
    "signalPeriod": 9,
    "input": "close"
  }
}
```

### 37.10.9 Rate Of Change

Directory: `indROC`

Type: `roc`

Parameters:

- `period`: comparison distance, default 14
- `input`: scalar candle input

Output:

- `value`: percentage rate of change

Formula:

```
ROC = (current - value[period]) / value[period] * 100
```

```
{"id":"roc10","type":"roc","params":{"period":10,"input":"close"}}
```

### 37.10.10 Stochastic Oscillator

Directory: `indStochastic`

Type: `stochastic`

Parameters:

- `period`: high/low lookback, default 14
- `smoothPeriod`: %D smoothing period, default 3

Outputs:

- k: current close position within the rolling high/low range
- d: SMA of %K

Formulas:

```
%K = (close - lowestLow) / (highestHigh - lowestLow) * 100
```

```
%D = SMA(%K, smoothPeriod)
```

```
{
  "id": "stoch14",
  "type": "stochastic",
  "params": {
    "period": 14,
    "smoothPeriod": 3
  }
}
```

### 37.10.11 OBV

Directory: `indOBV`

Type: `obv`

Parameters:

- `input`: comparison price, default `close`

Output:

- `value`: cumulative on-balance volume

Trade volume is added when the comparison price rises and subtracted when it falls. Equal prices leave OBV unchanged.

```
{"id":"obv","type":"obv","params":{"input":"close"}}
```

### 37.10.12 Parabolic SAR

Directory: `indPSAR`

Types: `psar` or `parabolicSar`

Parameters:

- `step`: acceleration factor increment, default `0.02`, must be positive

- `max`: acceleration factor cap, default `0.2`, must be at least `step`

Output:

- `value`: trailing stop-and-reverse level; sits below rising prices and above falling prices

Implements Wilder's algorithm including the prior-two-extreme clamp and trend reversal. The first candle seeds the state and publishes `EIVS_WARMUP`.

```
{"id":"psar","type":"psar","params":{"step":0.02,"max":0.2}}
```

### 37.10.13 Ichimoku Cloud

Directory: `indIchimoku`

Type: `ichimoku`

Parameters:

- `tenkanPeriod`: conversion line lookback, default 9
- `kijunPeriod`: base line lookback, default 26
- `senkouBPeriod`: leading span B lookback, default 52

Outputs:

- `tenkan`: midpoint of the tenkan period's high/low range
- `kijun`: midpoint of the kijun period's high/low range
- `senkouA`: `(tenkan + kijun) / 2`
- `senkouB`: midpoint of the senkouB period's high/low range

All outputs publish valid values simultaneously once the senkouB window is full. Values are undisplaced; the classic 26-bar span shift is a charting concern left to consumers.

```
{"id":"ichimoku","type":"ichimoku","params":{"tenkanPeriod":9,"kijunPeriod":26,"senkouBPeriod":52}}
```

### 37.10.14 CCI

Directory: `indCCI`

Type: `cci`

Parameters:

- `period`: rolling window, default 20

Output:

- `value`: commodity channel index of the typical price

`CCI = (tp - SMA(tp)) / (0.015 * meanAbsoluteDeviation)`. A perfectly flat window publishes `0`.

```
{"id":"cci20","type":"cci","params":{"period":20}}
```

### 37.10.15 MFI

Directory: `indMFI`

Type: `mfi`

Parameters:

- `period`: rolling window of money flows, default 14

Output:

- `value`: money flow index in the range `0..100`

Positive flow accumulates `tp * volume` when the typical price rises; negative flow when it falls. Degenerate windows publish `100` (only positive flow) or `50` (no flow at all).

```
{"id":"mfi14","type":"mfi","params":{"period":14}}
```

### 37.10.16 Keltner Channels

Directory: `indKeltnerChannels`

Types: `keltner` or `keltnerChannels`

Parameters:

- `period`: EMA period of the typical price, default 20
- `atrPeriod`: Wilder ATR period, default 10
- `multiplier`: band distance in ATR units, default 2, must be positive

Outputs:

- `middle`: EMA of the typical price
- `upper`: `middle + multiplier * ATR`
- `lower`: `middle - multiplier * ATR`

```
{"id":"keltner","type":"keltner","params":{"period":20,"atrPeriod":10,"multiplier":2}}
```

### 37.10.17 Donchian Channels

Directory: `indDonchianChannels`

Types: `donchian` or `donchianChannels`

Parameters:

- `period`: rolling window, default 20

Outputs:

- `upper`: highest high of the window
- `middle`: channel midpoint
- `lower`: lowest low of the window

```
{"id":"donchian20","type":"donchian","params":{"period":20}}
```

### 37.10.18 Standard Deviation

Directory: `indStdDev`

Types: `stddev` or `standardDeviation`

Parameters:

- `period`: rolling window, default 20, minimum 2
- `input`: scalar candle input

Output:

- `value`: population standard deviation of the input over the window

```
{"id":"stddev20","type":"stddev","params":{"period":20,"input":"close"}}
```

### 37.10.19 VWAP

Directory: `indVWAP`

Type: `vwap`

Parameters:

- `period`: rolling window of candles, default 20

Output:

- `value`: `sum(tp * volume) / sum(volume)` over the window

A rolling approximation of session VWAP; set `period` to the session candle count for session-anchored behavior. Zero volume in the window publishes `EIVS_WARMUP`.

```
{"id":"vwap20","type":"vwap","params":{"period":20}}
```

### 37.10.20 CMF

Directory: `indCMF`

Types: `cmf` or `chaikinMoneyFlow`

Parameters:

- `period`: rolling window, default 20

Output:

- `value`: Chaikin money flow, roughly in `(-1..1)`

`CMF = sum(closeLocationValue * volume) / sum(volume)`. A zero volume sum publishes `EIVS_WARMUP`.

```
{"id":"cmf20","type":"cmf","params":{"period":20}}
```

### 37.10.21 Volume Profile

Directory: `indVolumeProfile`

Type: `volumeProfile`

Parameters:

- `period`: rolling window of candles, default 100
- `bins`: price buckets across the window range, default 24, bounded `1..1024`
- `valueAreaPercent`: target volume share of the value area, default 70, bounded `(0..100]`

Outputs:

- `poc`: price level of the point of control
- `vah`: upper edge of the value area
- `val`: lower edge of the value area

Candle volume is binned by typical price; the value area grows from the POC by absorbing the larger adjacent bucket until the target share is reached. The profile is rebuilt from the raw window on every candle, so reset/replay determinism holds by construction.

```
{"id":"vprofile","type":"volumeProfile","params":{"period":100,"bins":24,"valueAreaPercent":70}}
```

### 37.10.22 A/D

Directory: `indAD`

Types: `ad` or `accumulationDistribution`

Parameters: none

Output:

- `value`: cumulative accumulation/distribution line

Volume enters the line weighted by where the close sits inside the candle range. Valid from the first priced candle.

```
{"id":"ad","type":"ad"}
```

### 37.10.23 Linear Regression Channel

Directory: `indLinearRegressionChannel`

Types: `linreg` or `linearRegressionChannel`

Parameters:

- `period`: regression window, default 20, minimum 2
- `deviation`: channel distance in residual standard deviations, default 2, must be positive
- `input`: scalar candle input

Outputs:

- `value`: regression line endpoint at the newest candle
- `upper`: `value + deviation * residualStdDev`
- `lower`: `value - deviation * residualStdDev`
- `slope`: least-squares slope

The line is fitted over the window with the oldest sample at x=0; channels derive from the population standard deviation of the fit residuals.

```
{"id":"linreg20","type":"linreg","params":{"period":20,"deviation":2,"input":"close"}}
```

### 37.10.24 Z-Score

Directory: `indZScore`

Type: `zscore`

Parameters:

- `period`: rolling window, default 20, minimum 2
- `input`: scalar candle input

Output:

- `value`: `(input - rollingMean) / rollingPopulationStdDev`

A perfectly flat window publishes `0`.

```
{"id":"zscore20","type":"zscore","params":{"period":20,"input":"close"}}
```

### 37.10.25 Relative Volume

Directory: `indRVOL`

Types: `rvol` or `relativeVolume`

Parameters:

- `period`: prior-candle window, default 20

Output:

- `value`: current candle volume divided by the mean volume of the previous `period` candles

Values above 1 mark above-average activity. A zero prior-volume window stays in `EIVS_WARMUP`.

```
{"id":"rvol20","type":"rvol","params":{"period":20}}
```

### 37.10.26 Choppiness Index

Directory: `indChoppiness`

Types: `chop` or `choppiness`

Parameters:

- `period`: rolling window, default 14, minimum 2

Output:

- `value`: `100 * log10(sumTrueRange / highestHighLowestLowRange) / log10(period)` in `0..100`

High values near 100 mark ranging markets; low values mark trending markets. A flat window publishes `100`.

```
{"id":"chop14","type":"chop","params":{"period":14}}
```

### 37.10.27 Elder Ray

Directory: `indElderRay`

Type: `elderRay`

Parameters:

- `period`: EMA period of close, default 13

Outputs:

- `bull`: `high - EMA(close)` (buying pressure)
- `bear`: `low - EMA(close)` (selling pressure)

```
{"id":"elderRay13","type":"elderRay","params":{"period":13}}
```

### 37.10.28 ADX / DMI

Directory: `indADX`

Type: `adx`

Parameters:

- `period`: Wilder DI smoothing period, default 14
- `smoothPeriod`: Wilder ADX smoothing period, default `period`

Outputs:

- `adx`: average directional index
- `plusDi`: `+DI`
- `minusDi`: `-DI`

Implements Wilder's directional movement including the seed means and the smoothed +DI/-DI/DX recursion. All three outputs publish valid values simultaneously once ADX seeding completes.

```
{"id":"adx14","type":"adx","params":{"period":14}}
```

### 37.10.29 Supertrend

Directory: `indSupertrend`

Type: `supertrend`

Parameters:

- `period`: Wilder ATR period, default 10
- `multiplier`: band distance in ATR units, default 3, must be positive

Outputs:

- `value`: trailing supertrend level
- `direction`: `1` in an uptrend (level below price), `-1` in a downtrend (level above price)

Basic bands sit around the HL2 midpoint and ratchet with the classic final-band rules; the trend flips when close crosses the active band.

```
{"id":"supertrend","type":"supertrend","params":{"period":10,"multiplier":3}}
```

### 37.10.30 VWMA

Directory: `indVWMA`

Type: `vwma`

Parameters:

- `period`: rolling window, default 20
- `input`: scalar candle price input, default `close`

Output:

- `value`: `sum(price * volume) / sum(volume)` over the window

Unlike session-anchored VWAP, VWMA is a plain rolling volume-weighted average. A zero volume sum publishes `EIVS_WARMUP`.

```
{"id":"vwma20","type":"vwma","params":{"period":20,"input":"close"}}
```

### 37.10.31 Fibonacci Retracement

Directory: `indFibonacci`

Types: `fibonacci` or `fibRetracement`

Parameters:

- `period`: rolling window, default 20

Outputs (measured upward from the window's lowest low toward its highest high):

- `level382`: 38.2% level
- `level500`: 50% level
- `level618`: 61.8% level
- `level786`: 78.6% level

```
{"id":"fib20","type":"fibonacci","params":{"period":20}}
```

### 37.10.32 Pivot Points

Directory: `indPivotPoints`

Types: `pivotPoints` or `pivot`

Parameters:

- `period`: window of candles providing high/low/close, default 1 (classic daily pivots when the series is daily)
- `style`: `classic` (default), `camarilla`, or `woodie`

Outputs:

- `pivot`: the pivot level
- `r1`, `r2`, `r3`: resistance levels
- `s1`, `s2`, `s3`: support levels

Classic and Woodie share the resistance/support formulas and differ in the pivot (`(H+L+C)/3` versus `(H+L+2C)/4`). Camarilla computes pivot `(H+L+C)/3` with close-anchored `1.1`-ratio bands (R4/S4 are not published).

```
{"id":"pivots","type":"pivotPoints","params":{"style":"classic"}}
```

### 37.10.33 Williams %R

Directory: `indWilliamsR`

Types: `williamsR` or `wpr`

Parameters:

- `period`: high/low lookback, default 14

Output:

- `value`: `(highestHigh - close) / (highestHigh - lowestLow) * -100` in `(-100..0]`

A flat window publishes `0`.

```
{"id":"wpr14","type":"williamsR","params":{"period":14}}
```

### 37.10.34 Stochastic RSI

Directory: `indStochRsi`

Types: `stochRsi` or `stochasticRsi`

Parameters:

- `rsiPeriod`: Wilder RSI period, default 14
- `stochPeriod`: RSI range lookback, default 14
- `smoothPeriod`: %D smoothing period, default 3
- `input`: scalar candle input

Outputs:

- k: position of RSI inside its own `stochPeriod` range, `0..100`
- d: SMA of %K

A flat RSI range publishes %K `0`.

```
{"id":"stochrsi","type":"stochRsi","params":{"rsiPeriod":14,"stochPeriod":14,"smoothPeriod":3}}
```

### 37.10.35 Williams Alligator

Directory: `indAlligator`

Type: `alligator`

Parameters:

- `jawPeriod` / `jawShift`: default 13 / 8
- `teethPeriod` / `teethShift`: default 8 / 5
- `lipsPeriod` / `lipsShift`: default 5 / 3
- `input`: scalar candle input

Outputs:

- `jaw`: SMMA of the jaw period, published displaced by `jawShift` bars
- `teeth`: SMMA of the teeth period, displaced by `teethShift`
- `lips`: SMMA of the lips period, displaced by `lipsShift`

The smoothed moving average seeds with the simple average of its first `period` values and then applies Wilder smoothing. Displacement is applied internally, so each output equals the SMMA value from `shift` bars ago.

```
{"id":"alligator","type":"alligator","params":{"jawPeriod":13,"jawShift":8,"teethPeriod":8,"teethSh
```

### 37.10.36 TRIX

Directory: `indTRIX`

Type: `trix`

Parameters:

- `period`: EMA period applied three times, default 14
- `input`: scalar candle input

Output:

- `value`: `100 * (tripleEma - previousTripleEma) / previousTripleEma`

```
{"id":"trix14","type":"trix","params":{"period":14,"input":"close"}}
```

### 37.10.37 KST

Directory: `indKST`

Type: `kst`

Parameters:

- `period1`..`period4`: ROC distances, default 10 / 15 / 20 / 30
- `smooth1`..`smooth4`: SMA smoothing of each ROC, default 10 / 10 / 10 / 15
- `signalPeriod`: SMA smoothing of KST, default 9
- `input`: scalar candle input

Outputs:

- `value`: `SMA(ROC1) + 2 * SMA(ROC2) + 3 * SMA(ROC3) + 4 * SMA(ROC4)`
- `signal`: SMA of KST

```
{"id":"kst","type":"kst","params":{"signalPeriod":9}}
```

### 37.10.38 Chaikin Oscillator

Directory: `indChaikinOscillator`

Types: `chaikinOscillator` or `chaikinOsc`

Parameters:

- `fastPeriod`: fast EMA of ADL, default 3
- `slowPeriod`: slow EMA of ADL, default 10, must be greater than `fastPeriod`

Output:

- `value`: fast EMA minus slow EMA of the accumulation/distribution line

```
{"id":"chaikinOsc","type":"chaikinOscillator","params":{"fastPeriod":3,"slowPeriod":10}}
```

### 37.10.39 Ultimate Oscillator

Directory: `indUltimateOscillator`

Types: `ultimateOscillator` or `ultimate`

Parameters:

- `period1`: short buying-pressure window, default 7
- `period2`: middle window, default 14, must be greater than `period1`
- `period3`: long window, default 28, must be greater than `period2`

Output:

- `value`: `100 * (4 * avg1 + 2 * avg2 + avg3) / 7` where each average is `sumBuyingPressure / sumTrueRange`

Buying pressure is `close - min(low, previousClose)` and true range spans `max(high, previousClose) - min(low, previousClose)`. Zero true-range windows contribute zero.

```
{"id":"uo","type":"ultimateOscillator","params":{"period1":7,"period2":14,"period3":28}}
```

### 37.10.40 Vortex Indicator

Directory: `indVortex`

Type: `vortex`

Parameters:

- `period`: rolling window, default 14

Outputs:

- `plus`: `sum(|high - previousLow|) / sumTrueRange`
- `minus`: `sum(|low - previousHigh|) / sumTrueRange`

VI+ crossing above VI- marks an emerging uptrend and vice versa. A zero true-range sum stays in `EIVS_WARMUP`.

```
{"id":"vortex14","type":"vortex","params":{"period":14}}
```

### 37.10.41 Ease of Movement

Directory: `indEOM`

Types: `eom` or `easeOfMovement`

Parameters:

- `period`: SMA smoothing window, default 14
- `divisor`: volume box divisor, default 10000, must be positive

Output:

- `value`: SMA of `midpointChange * range * divisor / volume`

Zero-volume candles contribute zero to the smoothed window.

```
{"id":"eom14","type":"eom","params":{"period":14,"divisor":10000}}
```

### 37.10.42 Elder Force Index

Directory: `indForceIndex`

Type: `forceIndex`

Parameters:

- `period`: EMA period of the raw force, default 13

Output:

- `value`: EMA of `(close - previousClose) * tradeVolume`

```
{"id":"force13","type":"forceIndex","params":{"period":13}}
```

### 37.10.43 PPO

Directory: `indPPO`

Type: `ppo`

Parameters:

- `fastPeriod`: fast EMA period, default 12
- `slowPeriod`: slow EMA period, default 26, must be greater than `fastPeriod`
- `signalPeriod`: signal EMA period, default 9
- `input`: scalar candle input

Outputs:

- `value`: `(fastEma - slowEma) / slowEma * 100`
- `signal`: EMA of the PPO line
- `histogram`: PPO minus signal

The percentage form makes PPO values comparable across instruments with different price scales.

```
{"id":"ppo","type":"ppo","params":{"fastPeriod":12,"slowPeriod":26,"signalPeriod":9,"input":"close'
```

### 37.10.44 Momentum

Directory: `indMomentum`

Type: `momentum`

Parameters:

- `period`: comparison distance, default 10
- `input`: scalar candle input

Output:

- `value`: `input - input[period]`

The absolute counterpart of ROC, which reports the percentage change instead.

```
{"id":"mom10","type":"momentum","params":{"period":10,"input":"close"}}
```

### 37.10.45 ATR Channels

Directory: `indAtrChannels`

Types: `atrChannels` or `atrBands`

Parameters:

- `period`: SMA period of the middle band, default 20
- `atrPeriod`: Wilder ATR period, default 14
- `multiplier`: band distance in ATR units, default 2, must be positive
- `input`: scalar candle input for the middle band

Outputs:

- `middle`: SMA of the input
- `upper`: `middle + multiplier * ATR`
- `lower`: `middle - multiplier * ATR`

Distinct from Keltner Channels, which use an EMA middle band.

```
{"id":"atrChannels","type":"atrChannels","params":{"period":20,"atrPeriod":14,"multiplier":2}}
```

### 37.10.46 TRIN / Arms Index

Directory: `indTRIN`

Types: `trin` or `armsIndex`

Parameters:

- `period`: rolling window of candles, default 14

Output:

- `value`: `(advancing / declining) / (upVolume / downVolume)`

Advancing and declining candles are classified against the previous close; unchanged candles are excluded. The window must contain both advancing and declining candles with volume on both sides, otherwise the output stays in `EIVS_WARMUP`. Most useful on breadth-aggregated series rather than a single instrument.

```
{"id":"trin14","type":"trin","params":{"period":14}}
```

### 37.10.47 NVI / PVI

Directory: `indVolumeIndex`

Types: `volumeIndex` or `nviPvi`

Parameters:

- `input`: scalar candle input, default `close`

Outputs:

- `nvi`: negative volume index, starting at 1000
- `pvi`: positive volume index, starting at 1000

Each candle's percentage price change is added to NVI when volume falls versus the previous candle and to PVI when volume rises; equal volume leaves both unchanged.

{"id":"nviPvi","type":"volumeIndex","params":{"input":"close"}}

## 37.11 Complete Multi-Indicator Example


```
{
  "id": "instBTC-5m",
  "source": "close",
  "timeFrame": 5,
  "candle": {
    "source": "tradeLast",
    "timeFrameSeconds": 300,
    "historyDepth": 2048,
    "metrics": ["vwap", "range", "trueRange"]
  },
  "indicators": [
    {"id":"ema20","type":"ema","params":{"period":20,"input":"close"}},
    {"id":"sma50","type":"sma","params":{"period":50,"input":"hlc3"}},
    {"id":"wma20","type":"wma","params":{"period":20,"input":"close"}},
    {"id":"hma21","type":"hma","params":{"period":21,"input":"close"}},
    {"id":"bands20","type":"bollinger","params":{"period":20,"deviation":2,"input":"close"}},
    {"id":"atr14","type":"atr","params":{"period":14}},
    {"id":"rsi14","type":"rsi","params":{"period":14,"input":"close"}},
    {"id":"macd","type":"macd","params":{"fastPeriod":12,"slowPeriod":26,"signalPeriod":9,"input":'
    {"id":"roc10","type":"roc","params":{"period":10,"input":"close"}},
    {"id":"stoch14","type":"stochastic","params":{"period":14,"smoothPeriod":3}},
    {"id":"obv","type":"obv","params":{"input":"close"}},
    {"id":"psar","type":"psar","params":{"step":0.02,"max":0.2}},
    {"id":"ichimoku","type":"ichimoku","params":{"tenkanPeriod":9,"kijunPeriod":26,"senkouBPeriod":
    {"id":"cci20","type":"cci","params":{"period":20}},
    {"id":"mfi14","type":"mfi","params":{"period":14}},
    {"id":"keltner","type":"keltner","params":{"period":20,"atrPeriod":10,"multiplier":2}},
    {"id":"donchian20","type":"donchian","params":{"period":20}},
    {"id":"stddev20","type":"stddev","params":{"period":20}},
    {"id":"vwap20","type":"vwap","params":{"period":20}},
    {"id":"cmf20","type":"cmf","params":{"period":20}},
    {"id":"vprofile","type":"volumeProfile","params":{"period":100,"bins":24}},
    {"id":"ad","type":"ad"},
    {"id":"linreg20","type":"linreg","params":{"period":20,"deviation":2,"input":"close"}},
    {"id":"zscore20","type":"zscore","params":{"period":20,"input":"close"}},
    {"id":"rvol20","type":"rvol","params":{"period":20}},
    {"id":"chop14","type":"chop","params":{"period":14}},
    {"id":"elderRay13","type":"elderRay","params":{"period":13}},
    {"id":"adx14","type":"adx","params":{"period":14}},
    {"id":"supertrend","type":"supertrend","params":{"period":10,"multiplier":3}},
    {"id":"vwma20","type":"vwma","params":{"period":20,"input":"close"}},
    {"id":"fib20","type":"fibonacci","params":{"period":20}},
    {"id":"pivots","type":"pivotPoints","params":{"style":"classic"}},
    {"id":"wpr14","type":"williamsR","params":{"period":14}},
    {"id":"stochrsi","type":"stochRsi","params":{"rsiPeriod":14,"stochPeriod":14,"smoothPeriod":3}}
    {"id":"alligator","type":"alligator","params":{}},
    {"id":"trix14","type":"trix","params":{"period":14,"input":"close"}},
    {"id":"kst","type":"kst","params":{}},
    {"id":"chaikinOsc","type":"chaikinOscillator","params":{"fastPeriod":3,"slowPeriod":10}},
```

```
    {"id":"uo","type":"ultimateOscillator","params":{"period1":7,"period2":14,"period3":28}},
    {"id":"vortex14","type":"vortex","params":{"period":14}},
    {"id":"eom14","type":"eom","params":{"period":14}},
    {"id":"force13","type":"forceIndex","params":{"period":13}},
    {"id":"ppo","type":"ppo","params":{"fastPeriod":12,"slowPeriod":26,"signalPeriod":9,"input":"cl
    {"id":"mom10","type":"momentum","params":{"period":10,"input":"close"}},
    {"id":"atrChannels","type":"atrChannels","params":{"period":20,"atrPeriod":14,"multiplier":2}},
    {"id":"trin14","type":"trin","params":{"period":14}},
    {"id":"nviPvi","type":"volumeIndex","params":{"input":"close"}}
  ]
}
```

## 37.12 Historical Initialization

Historical candles are normalized to chronological order before dispatch. Each finalized historical candle advances the indicators exactly once. Path-dependent indicators such as EMA, ATR, RSI, MACD, and OBV therefore have the same state whether the candles arrived historically or through uninterrupted live processing.

An algo must still handle `EIVS_WARMUP`; the amount of requested historical data may be smaller than an indicator's configured warm-up requirement.

## 37.13 Error Handling

The entire series configuration fails when an enabled indicator has:

- no `id`
- no `type`
- an unknown native type
- a malformed descriptor
- invalid parameter types
- periods outside supported bounds
- an unknown candle input
- an allocation failure during initialization

Configuration loading is transactional for the indicator list: indicators appended during the failed load are destroyed and removed.

## 37.14 Current Scope

Operational:

- Per-indicator V2 DLL loading and module caching
- Opaque instance ownership with DLL-local destruction
- Native built-in indicators
- Finalized-candle cadence
- Historical warm-up
- Bounded output history
- Enum and numeric output lookup; names remain descriptor metadata
- Configurable candle inputs
- HFT-safe update paths

Not currently operational:

- Working-candle preview indicators
- Raw trade or raw BBO indicator cadence

- Indicator-to-indicator dependency graphs
- Runtime reconfiguration
- Indicator state-file persistence
- `buildMode: "dualValidate"` (exchange/local cross-validation; rejected at config parse)

The V2 API reserves cadence values for future working/raw dispatch, but all current native descriptors are `EIC_CLOSE_ONLY`.

## 37.15 Exchange-Native Streaming Candles

A candle series can use venue klines instead of local trade/BBO aggregation:

```
{
  "id": "instBTC-1m-xchg",
  "source": "exchange",
  "buildMode": "exchangeOnly",
  "timeFrame": 1,
  "historyDepth": 256,
  "indicators": [ {"id":"atr14","type":"atr","params":{"period":14}} ]
}
```

With `source: "exchange"`, the adapter subscribes the venue kline stream (Binance futures: one combined `<symbol>@kline_<interval>` SUBSCRIBE per instrument), provisional updates build the working candle, and the venue's final frame closes it immediately - no watermark wait. Historical warm-up still uses the venue REST klines. The watermark timer remains as a safety net that closes the working candle if the exchange stream stalls.

Candle series selection is driven by `source` plus the optional `buildMode`:

| buildMode | Meaning |
|---|---|
| `exchangeOnly` | venue kline frames only; requires `source: "exchange"`. A series on a venue that never pushes frames stays empty |
| `localOnly` | local aggregation only; requires a local `source` (`close`/`mid`). Venue frames are never accepted |
| `auto` (default) | with `source: "exchange"`: venue kline frames are authoritative once the stream proves alive; until the first venue frame the series aggregates locally from trades (fallback). With a local `source` it behaves as `localOnly` |
| `dualValidate` | not implemented - rejected at config parse |

The auto handover is clean: the first venue frame redefines the working window with venue values (no mixing of local trade aggregates into venue-authored ohlc/volume), and the venue stream stays authoritative from then on. Bitstamp uses `buildMode: "auto"` because its websocket v2 has no kline channel: REST seeds history, live windows aggregate from trades, and the certification suite's Marketdata Candles test self-skips with a clear reason while no venue frame has arrived.

Binance futures streams klines and certifies 27/27 with an exchange-stream series (2026-09-06); the earlier zero-frames defect is fixed.

## 37.16 Certification

Two complementary certification layers are provided:

1. `ttManagerOrderTests.exe IndicatorsV2` is the deterministic, credential-free authority for formulas, warm-up, circular history, exact-sequence reads, ABI fingerprinting, and DLL lifecycle.
2. `algoIndicatorTest.dll` validates the complete runtime path through configuration, exchange market data, candle finalization, indicator DLL dispatch, algo subscription, and enum-based reads.

Every `indicators/ind<Name>/` folder contains an `algoIndicatorTest.cfg`. These configs use Binance public market data in simulation mode. They place no orders and require no private credentials, but they require network access. See `../algos/algoIndicatorTest/README.md`.

Every folder additionally contains an `algoIndicatorTest-static.cfg` for the deterministic offline path: the same runtime certification against `exchgStaticIndicator` with no network access.

For deterministic end-to-end certification, `exchgStaticIndicator.dll` supplies 256 completed historical candles and one wall-clock-valid trade/BBO pair per second. Every indicator folder contains its own `algoIndicatorTest-static.cfg` for this path; the reusable sweep script `.kilo/cert-results/sweep-indicator-static.ps1` walks all folders sequentially. This path exercises the actual exchange plugin, market-data manager, candle finalizer, Indicator V2 DLL, and algo callback without network access.

Runtime certification shuts down gracefully: after printing its result, `algoIndicatorTest` posts the normal application exit request, the full shutdown waterfall runs (indicator instances destroyed through their owning DLLs, modules unloaded with zero live instances, asio context joined), and the process exits with code `0` by itself. Debug builds route CRT assert reports to stderr instead of modal dialogs, so a failing invariant can never stall a headless certification run.

Debug builds publish informational `steady_clock` timing statistics for enum-based exact-sequence reads. Timing values are not functional pass/fail gates because debug builds and host load are not stable benchmark environments.

The deterministic DLL check loads every plugin in the catalog, validates version/fingerprint/type/descriptor contracts, creates and configures an instance, exercises reset, destroys it, and unloads the module. This covers all plugins without requiring exchange access.

The deterministic suite also reports per-indicator debug update and output-read averages over 1,000 operations. These measurements are intended to expose regressions and relative hotspots. They are not release thresholds because debug instrumentation and host scheduling materially affect absolute latency.

The suite prints:

```
INDICATOR CERTIFICATION SUITE SUMMARY
RESULT: CERTIFIED
```

and returns process exit code `0` only when every selected certification check passes.

## 37.17 Adding A Native Indicator

1. Create `indicators/ind<Name>/indicatorDefinition.h`.
2. Define stable output IDs and an `indicatorDescriptor_s`.
3. Derive the implementation from `nativeIndicatorBase_c`.
4. Allocate all runtime-capacity storage during `configure()`.
5. Keep `onCandleClose()` allocation-free.
6. Use `decimal_t` for financial calculations.
7. Implement `_resetState()` and `_releaseInstance()`.
8. Add an `indicatorDll.cpp` export using `indicatorDllApi_c`.
9. Register the DLL target in `indicators/CMakeLists.txt` and the host type table in `storeIndicator_c`.
10. Add deterministic formula, warm-up, reset, invalid-config, enum-output, and DLL lifecycle tests.

Stable `indicatorOutput_e` values are part of the algo-facing contract. Descriptor IDs such as `value`, `upper`, or `histogram` support diagnostics and tooling and should also remain stable.

# Chapter 38

# A/D - Accumulation/Distribution Line

## 38.1 Formula

```
clv = ((close - low) - (high - close)) / (high - low)     [0 when high == low]
AD  = AD + clv * volume
```

Cumulative money-flow proxy: volume enters the line weighted by where the close sits inside the candle range.

## 38.2 Configuration

None. All candles contribute.

## 38.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | cumulative accumulation/distribution volume |

## 38.4 Implementation Details

- Single cumulative accumulator, O(1) per update.
- Valid from the first priced candle (no warm-up dependency).
- All arithmetic uses `decimal_t`.

## 38.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 39

# ATR - Average True Range

## 39.1 Formula

```
TR = max(High - Low, |High - PrevClose|, |Low - PrevClose|)
ATR = SMA(TR, period) for first N bars, then Wilder smoothing:
ATR_t = (ATR_{t-1} * (period - 1) + TR_t) / period
```

## 39.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Smoothing period |

Input is always OHLC (no configurable input source).

## 39.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | ATR value |

Status is `EIVS_WARMUP` until `period` true-range samples are accumulated.

## 39.4 Implementation Details

- First bar has no previous close; TR = High - Low only.
- Seed phase accumulates sum of TR values; divides by period once count reaches period.
- Post-seed uses Wilder smoothing recurrence (equivalent to EMA with alpha=1/period).
- All arithmetic uses `decimal_t`; abs() via std::max of positive/negative.
- No circular buffer needed; only previous close and running ATR are stored.

## 39.5 Certification

Use `ttManagerOrderTests.exe IndicatorsV2` for deterministic formula and DLL lifecycle checks. Use `algoIndicatorTest.cfg` for runtime subscription smoke testing.

## 39.6 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 40

# Bollinger Bands

## 40.1 Formula

```
Middle = SMA(Close, period)
StdDev = sqrt(sum((x_i - Middle)^2) / period)
Upper = Middle + deviation * StdDev
Lower = Middle - deviation * StdDev
Bandwidth = (Upper - Lower) / Middle
```

## 40.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 20 | 2 | SMA lookback period |
| deviation | decimal | 2.0 | >0 | Standard deviation mult |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 40.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | middle | decimal | SMA (middle band) |
| 1 | upper | decimal | Upper band |
| 2 | lower | decimal | Lower band |
| 3 | bandwidth | decimal | Band width / middle ratio |

Status is `EIVS_WARMUP` until `period` samples fill the buffer.

## 40.4 Implementation Details

- Uses `preallocatedWindow_c<decimal_t>` as circular buffer of size `period`.
- Computes population standard deviation (divides by N, not N-1).
- Mean and variance computed via two-pass iteration over the window.

- `sqrt()` from `<cmath>` operates on `decimal_t`.
- Four outputs published simultaneously; all invalid during warmup.
- All arithmetic uses `decimal_t`.

## 40.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 41

# CCI - Commodity Channel Index

## 41.1 Formula

```
tp  = (high + low + close) / 3
CCI = (tp - SMA(tp, period)) / (0.015 * meanAbsoluteDeviation(tp, period))
```

When the mean absolute deviation is zero (perfectly flat window), CCI publishes `0`.

## 41.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 20 | 1 | Rolling window |

## 41.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | CCI oscillator (unbounded) |

## 41.4 Implementation Details

- `preallocatedWindow_c<decimal_t>` holds the last `period` typical prices; two O(period) passes compute the mean and the mean absolute deviation.
- Output valid once the window is full.
- All arithmetic uses `decimal_t`.

## 41.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |

| File | Purpose |
| --- | --- |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 42

# CMF - Chaikin Money Flow

## 42.1 Formula

```
clv = ((close - low) - (high - close)) / (high - low)     [0 when high == low]
mfv = clv * volume
CMF = sum(mfv, period) / sum(volume, period)
```

A zero volume sum over the window publishes `EIVS_WARMUP`.

## 42.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 20 | 1 | Rolling window |

## 42.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | CMF, roughly in `(-1..1)` |

## 42.4 Implementation Details

- `preallocatedWindow_c<std::array<decimal_t, 2>>` stores `{mfv, volume}` pairs with rolling sums updated via replaced-element retrieval.
- O(1) per update after fill; no allocation in the update path.
- All arithmetic uses `decimal_t`.

## 42.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

| File | Purpose |
|---|---|
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 43

# Donchian Channels

## 43.1 Formula

```
upper  = highestHigh(period)
lower  = lowestLow(period)
middle = (upper + lower) / 2
```

## 43.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 20 | 1 | Rolling window |

## 43.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | upper | decimal | highest high of window |
| 1 | middle | decimal | channel midpoint |
| 2 | lower | decimal | lowest low of window |

## 43.4 Implementation Details

- `preallocatedWindow_c` of high/low pairs sized to the period; one O(period) scan per candle.
- Output valid once the window is full.
- All arithmetic uses `decimal_t`.

## 43.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

| File | Purpose |
|---|---|
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 44

# EMA - Exponential Moving Average

## 44.1 Formula

```
multiplier = 2 / (period + 1)
EMA_t = (value_t - EMA_{t-1}) * multiplier + EMA_{t-1}
```

Seed: SMA of first `period` values.

## 44.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Lookback period |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 44.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | EMA value |

Status is `EIVS_WARMUP` until `period` samples are accumulated, then `EIVS_VALID`.

## 44.4 Implementation Details

- Uses `emaAccumulator_c` from `nativeIndicatorBase.h` for stateful accumulation.
- Seed phase accumulates a running sum; once `period` samples arrive, seed = sum / period.
- Post-seed applies the standard EMA recurrence using `decimal_t` arithmetic.
- Inherits ring-buffer output storage and monotonic sequence enforcement from `nativeIndicatorBase_c`.
- No heap allocation in hot path; all state is stack or preallocated via `st_malloc`.

## 44.5 Certification

Run the shared deterministic certification suite from the repository build:

```
.build/clang-debug/ttManagerOrderTests.exe IndicatorsV2
```

The folder also contains `algoIndicatorTest.cfg` for full runtime configuration and market-data subscription smoke testing.

## 44.6 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

W:\development\ttTrader\indicators\indEMA\README.md

# Chapter 45

# HMA - Hull Moving Average

## 45.1 Formula

```
halfPeriod = floor(period / 2)
sqrtPeriod = round(sqrt(period))
raw = 2 * WMA(halfPeriod) - WMA(period)
HMA = WMA(raw, sqrtPeriod)
```

## 45.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Lookback period |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 45.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | HMA value |

Status is `EIVS_WARMUP` until both inner WMA windows and the outer sqrt-period window are full.

## 45.4 Implementation Details

- Uses two `preallocatedWindow_c<decimal_t>` buffers: `m_closes` (size=period) and `m_raw` (size=sqrtPeriod).
- `sqrtPeriod` is computed at configure time as the nearest integer square root of period.
- Inner `_wma()` helper computes weighted average over a sub-range of a circular buffer.
- Raw series = `2 * WMA(closes, halfPeriod) - WMA(closes, 0)` pushed into `m_raw`.
- Final output = WMA of `m_raw` buffer once it is full.
- Reduces lag compared to SMA/EMA while maintaining smoothness.
- All arithmetic uses `decimal_t`.

## 45.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / _wma / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 46

# Ichimoku Cloud

## 46.1 Formula

```
Tenkan  = (highestHigh(tenkan) + lowestLow(tenkan)) / 2
Kijun   = (highestHigh(kijun) + lowestLow(kijun)) / 2
SenkouA = (Tenkan + Kijun) / 2
SenkouB = (highestHigh(senkouB) + lowestLow(senkouB)) / 2
```

Outputs are the raw (undisplaced) values. The classic 26-bar forward displacement of the senkou spans and the 26-bar backward displacement of the chikou span are chart-drawing concerns and are intentionally left to consumers.

## 46.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| tenkanPeriod | uint32 | 9 | 1 | Conversion line lookback |
| kijunPeriod | uint32 | 26 | 1 | Base line lookback |
| senkouBPeriod | uint32 | 52 | 1 | Leading span B lookback |

## 46.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | tenkan | decimal | conversion line |
| 1 | kijun | decimal | base line |
| 2 | senkouA | decimal | (tenkan + kijun) / 2 |
| 3 | senkouB | decimal | midpoint of the senkouB window |

All four outputs publish valid values simultaneously once the senkouB window is full, so the warm-up equals the senkouB period.

## 46.4 Implementation Details

- One `preallocatedWindow_c` of high/low pairs sized to the senkouB period; each midpoint scans the newest N entries.

- O(senkouBPeriod) per update; no allocation in the update path.
- All arithmetic uses `decimal_t`.

## 46.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 47

# Keltner Channels

## 47.1 Formula

```
middle = EMA(typicalPrice, period)
ATR    = Wilder-smoothed average true range over atrPeriod
upper  = middle + multiplier * ATR
lower  = middle - multiplier * ATR
```

The EMA is seeded with the simple average of the first `period` typical prices; the ATR uses the same seeding as the native ATR indicator (mean of the first `atrPeriod` true ranges, then Wilder smoothing).

## 47.2 Configuration

| Key | Type | Default | Validation | Description |
|---|---|---|---|---|
| period | uint32 | 20 | >= 1 | EMA period of the middle band |
| atrPeriod | uint32 | 10 | >= 1 | Wilder ATR period |
| multiplier | decimal | 2 | > 0 | Band distance in ATR units |

## 47.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | middle | decimal | EMA of typical price |
| 1 | upper | decimal | middle + multiplier * ATR |
| 2 | lower | decimal | middle - multiplier * ATR |

Outputs become valid once both the EMA seed and the ATR seed are complete (`max(period, atrPeriod)` candles).

## 47.4 Implementation Details

- Reuses `emaAccumulator_c` from the shared base for the middle band.
- ATR state mirrors `indATR` (running sum, then Wilder smoothing).
- O(1) per update; no allocation in the update path.
- All arithmetic uses `decimal_t`.

## 47.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 48

# MACD - Moving Average Convergence Divergence

## 48.1 Formula

```
MACD line = EMA(fastPeriod) - EMA(slowPeriod)
Signal line = EMA(MACD line, signalPeriod)
Histogram = MACD line - Signal line
```

## 48.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| fastPeriod | uint32 | 12 | 1 | Fast EMA period |
| slowPeriod | uint32 | 26 | 2 | Slow EMA period |
| signalPeriod | uint32 | 9 | 1 | Signal EMA period |
| input | string | close | | Candle input source |

Constraint: slowPeriod must be > fastPeriod.

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 48.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | macd | decimal | MACD line |
| 1 | signal | decimal | Signal line |
| 2 | hist | decimal | Histogram (macd-signal) |

Status is `EIVS_WARMUP` until both EMAs are seeded and signal EMA has enough samples.

## 48.4 Implementation Details

- Uses three `emaAccumulator_c` instances: fast, slow, signal.

- MACD line computed as difference of fast/slow EMA outputs each bar.
- Signal EMA only receives input when both fast and slow EMAs are valid.
- Three outputs published simultaneously; all invalid during warmup.
- All arithmetic uses `decimal_t`.

## 48.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 49

# MFI - Money Flow Index

## 49.1 Formula

```
tp  = (high + low + close) / 3
flow = tp * volume
positive flow: tp > previous tp; negative flow: tp < previous tp
MFI = 100 - 100 / (1 + sum(positiveFlow, period) / sum(negativeFlow, period))
```

Conventions when the denominator is degenerate: negative flow sum zero with positive flow present publishes `100`; both sums zero publish `50`.

## 49.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Rolling window of flows |

## 49.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | MFI in the range `0..100` |

The first candle only records the comparison price and publishes `EIVS_WARMUP`; the output becomes valid once `period` real flows are accumulated (candle `period + 1`).

## 49.4 Implementation Details

- `preallocatedWindow_c<std::array<decimal_t, 2>>` stores `{positiveFlow, negativeFlow}` pairs with rolling sums updated via replaced-element retrieval.
- O(1) per update after fill; no allocation in the update path.
- All arithmetic uses `decimal_t`.

## 49.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 50

# OBV - On-Balance Volume

## 50.1 Formula

```
if Close > PrevClose: OBV += Volume
if Close < PrevClose: OBV -= Volume
if Close == PrevClose: OBV unchanged
```

## 50.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| input | string | close | | Comparison source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

Volume is always taken from `candle.base.decTradeVolume`.

## 50.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | Cumulative OBV |

Status is valid from the second sample onward (first sample sets baseline).

## 50.4 Implementation Details

- No period parameter; purely cumulative from stream start.
- Compares current input value against previous; adds/subtracts volume accordingly.
- First bar sets `m_decPreviousClose` without modifying OBV.
- Uses `_sampleCount() > 0` to gate validity after first update.
- All arithmetic uses `decimal_t`.

## 50.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 51

# Parabolic SAR

## 51.1 Formula

Wilder's stop-and-reverse with an accelerating trailing level:

`SAR_next = SAR + AF * (EP - SAR)`

- rising trend: `SAR_next` is clamped to the prior two lows; a low below SAR reverses to falling and sets SAR = EP
- falling trend: `SAR_next` is clamped to the prior two highs; a high above SAR reverses to rising and sets SAR = EP
- EP (extreme point) is the highest high while rising / lowest low while falling
- AF (acceleration factor) starts at `step` and increases by `step` on every new extreme point, capped at `max`

## 51.2 Configuration

| Key | Type | Default | Validation | Description |
|---|---|---|---|---|
| step | decimal | 0.02 | > 0 | AF increment per new extreme |
| max | decimal | 0.2 | >= step | AF cap |

## 51.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | trailing SAR level (flip side of trend) |

The first candle seeds the state and publishes `EIVS_WARMUP`; every later candle publishes a valid value.

## 51.4 Implementation Details

- O(1) per update; state is the SAR level, EP, AF, trend flag, and prior two highs/lows.
- Reversal publishes the previous extreme point as the new SAR (standard Wilder behavior).
- All arithmetic uses `decimal_t`.

## 51.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 52

# ROC - Rate of Change

## 52.1 Formula

```
ROC = ((value_t - value_{t-N}) / value_{t-N}) * 100
```

## 52.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Lookback period |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 52.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | ROC percentage |

Status is `EIVS_WARMUP` until `period + 1` samples are in the buffer.

## 52.4 Implementation Details

- Uses `preallocatedWindow_c<decimal_t>` with capacity `period + 1`.
- Oldest element (`at(0)`) is the denominator; newest is the current value.
- Output = `(newest - oldest) / oldest * 100`.
- O(1) per update after buffer fill; just push and read endpoints.
- All arithmetic uses `decimal_t`.

## 52.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 53

# RSI - Relative Strength Index

## 53.1 Formula

```
delta = Close_t - Close_{t-1}
gain = max(delta, 0)
loss = max(-delta, 0)
AvgGain = SMA(gain, period) then Wilder smoothing
AvgLoss = SMA(loss, period) then Wilder smoothing
RS = AvgGain / AvgLoss
RSI = 100 - 100 / (1 + RS)
```

## 53.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Smoothing period |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 53.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | RSI value (0-100) |

Status is `EIVS_WARMUP` until `period` deltas are processed.

## 53.4 Implementation Details

- First sample sets previous close without producing output.
- Seed phase sums gains and losses over first `period` deltas, then divides by period.
- Post-seed uses Wilder smoothing: `avg = (avg * (period-1) + new) / period`.
- Edge case: if AvgLoss == 0, RSI = 100 (avoids division by zero).
- All arithmetic uses `decimal_t`.

## 53.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 54

# SMA - Simple Moving Average

## 54.1 Formula

```
SMA_t = (sum of last N values) / N
```

## 54.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Lookback period |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 54.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | SMA value |

Status is `EIVS_WARMUP` until `period` samples are accumulated, then `EIVS_VALID`.

## 54.4 Implementation Details

- Uses `preallocatedWindow_c<decimal_t>` as a circular buffer of size `period`.
- Maintains a running sum (`m_decSum`); on push, adds new value and subtracts replaced value if buffer is full.
- Output = `m_decSum / period` once buffer is full; outputs 0 with warmup status before that.
- O(1) per update after initial fill; no iteration over the window.
- All arithmetic uses `decimal_t`.

## 54.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 55

# Standard Deviation

## 55.1 Formula

```
stdDev = sqrt( sum( (value - SMA(value, period))^2 ) / period )
```

Population standard deviation (divide by `period`), matching the Bollinger Bands internal computation.

## 55.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 20 | 2 | Rolling window |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 55.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | population standard deviation |

## 55.4 Implementation Details

- `preallocatedWindow_c<decimal_t>` with rolling mean and variance passes, identical in shape to `indBollingerBands`.
- Output valid once the window is full.
- All arithmetic uses `decimal_t`; `sqrt` operates on `decimal_t`.

## 55.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |

| File | Purpose |
|---|---|
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 56

# Stochastic Oscillator

## 56.1 Formula

```
%K = ((Close - LowestLow) / (HighestHigh - LowestLow)) * 100
%D = SMA(%K, smoothPeriod)
```

Where HighestHigh and LowestLow are over the lookback `period`.

## 56.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Lookback period for %K |
| smoothPeriod | uint32 | 3 | 1 | Smoothing period for %D |

Input is always OHLC (uses high, low, close directly).

## 56.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | k | decimal | Fast stochastic %K |
| 1 | d | decimal | Slow stochastic %D |

Status is `EIVS_WARMUP` until both the high-low window and the %K smoothing window are full.

## 56.4 Implementation Details

- Uses `preallocatedWindow_c<highLow_s>` to track high/low pairs over the lookback period.
- Scans entire window each bar to find highest high and lowest low (O(N)).
- %K = 0 when range is zero (avoids division by zero).
- %D computed via running sum over a second circular buffer of %K values.
- All arithmetic uses `decimal_t`.

## 56.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Chapter 57

# VWAP - Volume Weighted Average Price

## 57.1 Formula

```
tp   = (high + low + close) / 3
VWAP = sum(tp * volume, period) / sum(volume, period)
```

A rolling VWAP over the configured candle window. A zero volume sum (no traded volume in the window) publishes `EIVS_WARMUP` instead of a division-by-zero. For session-anchored behavior, set `period` to the number of candles in the trading session; the bounded window is the deterministic equivalent of the daily reset.

## 57.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 20 | 1 | Rolling window of candles |

## 57.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | volume weighted average price |

## 57.4 Implementation Details

- `preallocatedWindow_c<std::array<decimal_t, 2>>` stores `{tp * volume, volume}` pairs; rolling sums updated via replaced-element retrieval.
- O(1) per update after fill; no allocation in the update path.
- All arithmetic uses `decimal_t`.

## 57.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |

| File | Purpose |
|---|---|
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 58

# Volume Profile

## 58.1 Formula

The last `period` candles are binned by typical price into `bins` equal-width buckets spanning `[min(tp), max(tp)]`; each candle contributes its full volume to the bucket containing its typical price (`hlc3`).

```
POC = center of the bucket with the highest accumulated volume
VAH / VAL = edges of the value area
```

The value area grows from the POC by greedily absorbing the adjacent bucket with the larger volume until the area holds at least `valueAreaPercent` of the window's total volume.

## 58.2 Configuration

| Key | Type | Default | Validation | Description |
|---|---|---|---|---|
| period | uint32 | 100 | >= 1 | Rolling window of candles |
| bins | uint32 | 24 | 1..1024 | Price buckets across the window |
| valueAreaPercent | decimal | 70 | (0..100] | Target share of volume in the value area |

## 58.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | poc | decimal | price level of the point of control |
| 1 | vah | decimal | upper edge of the value area |
| 2 | val | decimal | lower edge of the value area |

When every typical price in the window is identical, all three outputs equal that price.

## 58.4 Implementation Details

- Candle window is `preallocatedWindow_c<sample_s>`; bucket volumes are a fixed `st_malloc`'ed array allocated during `configure()` and freed in the destructor.
- O(period + bins) recomputation per candle; no allocation in the update path.

- The profile is rebuilt from the raw window on every close, so reset/replay determinism and sparse sequences hold by construction.
- All arithmetic uses `decimal_t`.

## 58.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |
| algoIndicatorTest.cfg | Integration test config (Binance) |
| algoIndicatorTest-static.cfg | Offline cert config (exchgStaticIndicator) |

# Chapter 59

# WMA - Weighted Moving Average

## 59.1 Formula

```
WMA_t = sum(value_i * weight_i) / sum(weight_i)
weight_i = i + 1  (oldest=1, newest=N)
```

## 59.2 Configuration

| Key | Type | Default | Min | Description |
|---|---|---|---|---|
| period | uint32 | 14 | 1 | Lookback period |
| input | string | close | | Candle input source |

Supported inputs: open, high, low, close, hl2, hlc3, ohlc4, vwap.

## 59.3 Outputs

| Index | Name | Type | Description |
|---|---|---|---|
| 0 | value | decimal | WMA value |

Status is `EIVS_WARMUP` until `period` samples are accumulated, then `EIVS_VALID`.

## 59.4 Implementation Details

- Uses `preallocatedWindow_c<decimal_t>` as a circular buffer of size `period`.
- On each update after buffer is full, iterates all elements computing weighted sum where weight = index + 1.
- Divisor is the triangular number `N*(N+1)/2` computed inline as `l_ui64WeightSum`.
- O(N) per update; acceptable for typical period sizes in HFT context.
- All arithmetic uses `decimal_t`.

## 59.5 Files

| File | Purpose |
|---|---|
| indicatorDefinition.h | Descriptor and class declaration |
| indicator.cpp | configure / onCandleClose / reset |
| indicatorDll.cpp | DLL export entry point |

# Part VI

# Appendix A - Glossary

# Chapter 60

# Glossary

Terms used across the ttTrader manual. Links point to the chapter that explains each in depth.

| Term | Meaning |
|---|---|
| **algo / algo plugin** | A strategy compiled as a DLL, deriving from `algoFramework_c`; loaded by `storePlugin_c`. See Trading Algorithms. |
| **Indicator V2** | The OHLC-attached indicator plugin model; each indicator instance is declared in an instrument's `ohlc[].indicators[]` and referenced by id from algo parameters. See Indicators. |
| **Exchange adapter / plugin** | A venue DLL deriving from the exchange protocol layout (marketdata/trading/account channels). See Exchange Integrations. |
| **Store** | A singleton that loads and holds configuration-derived state (`storeExchange_c`, `storeAsset_c`, `storeInstrument_c`, `storeOhlc_c`, `storeAlgo_c`, `storePlugin_c`). See Architecture: Data Stores. |
| **Manager** | A long-lived orchestration component (`managerMain_c`, `managerMarketdata_c`, `managerOrder_c`, `managerExchange_c`, `managerWebSocket_c`, `managerPlayback_c`). See Architecture: Manager Hierarchy. |
| **Event handler / event system** | The lock-free inter-component messaging core: a preallocated MPSC ring per handler, drained on a single-writer strand. See System Protocol. |
| **Event target** | A stable, shared delivery handle (`eventTarget_t`) used to post to another component without holding a raw handler pointer. See System Protocol. |
| **Strand** | A Boost.Asio serialized execution context; one per event handler, giving single-writer semantics without locks. See Architecture: Strand-Based Async Model. |
| **MPSC ring** | Multi-producer / single-consumer preallocated queue used for cross-thread event delivery. |
| **Dispatch guard** | Per-handler barrier that lets a destructor wait for in-flight callbacks, preventing use-after-free during teardown. |

| Term | Meaning |
|---|---|
| **Asset** | A balance-bearing unit (e.g. `assetLighterUSDC`) declared in the config `assets` section and subscribed by algos. See Configuration: assets. |
| **Balance share (`balanceFraction`)** | An algo's per-asset share of the account, declared in the algo's `assets` entry; resolved across all enabled algos at load. See Balance shares and PnL isolation. |
| **Anchor / first-snapshot allocation** | The algo's fixed buying-power basis, captured as `startAvailable x share` on the first asset balance snapshot. |
| **Realized PnL** | Closed-trade profit/loss, net of fees; the only PnL that moves an algo's isolated balance. |
| **External delta** | The part of a live balance change not explained by any algo's credit (`anchor + realized`): a deposit/withdrawal, rescaled proportionally to each algo's share. |
| **Settlement / settling pass** | `managerMain_c`'s 1 Hz aggregation that waits for credit reports to stop before computing the external delta, so an in-flight PnL is not misread as a deposit. |
| **Trading mode** | `live` / `paper` / `simulation` / `playback`; selects the exchange `serverSettings` client set and the order manager. See Configuration: Trading modes. |
| **OHLC series** | A venue-streamed candle feed declared per instrument/timeframe (`source`, `timeFrame`, `historyDepth`, `gapFillEmpty`). |
| **Certificate / cert sweep** | A recorded certification run against exchanges or indicators; sweeps are dated artifacts kept in `archive/`. |
| **State file** | Per-algo JSON persistence (`stateFile`), written by the framework and read at startup. |
| **Credential ref** | A `cred:<scope>/<path>` reference resolved through the OS credential store instead of a plaintext secret. See Credential Store. |