Nutcracker Agentic Interface Specification

Tool Schema for Deterministic Execution Environments

Version 0.1 — Pilot Release

Aleph Strategy R&D Lab *For reference: Nutcracker Engine Whitepaper v1 | DEE Paper (Zenodo: 10.5281/zenodo.19736833)*

Implementation Status

Component	Status
Account creation & access key generation	Implemented
Bot lifecycle control (start / stop / restart)	Implemented
Status & diagnostics	Implemented
PnL analytics endpoints	Implemented
Parameter read	Implemented
Parameter proposals (human-gated)	Implemented
Exchange connectivity	Implemented
API key permission validation (no-withdrawal / spot-only enforcement)	Roadmap v1.1
Asset whitelisting per principal	Roadmap v1.1
Headless JSON console (parallel agent interface)	Roadmap v1.2
MCP tool wrapper	Roadmap commercial
Sovereign AI principal model	Roadmap future

Section 1 — Preamble

1.1 Purpose of This Document

This specification defines the agentic interface of the Nutcracker Engine — the machine-readable surface through which an AI agent, acting under delegation from an account owner, may interact with the Nutcracker Deterministic Proxy.

It is a trust instrument as much as a technical reference. Every endpoint declaration is accompanied by explicit constraint declarations and human approval requirements. The negative declarations — what the interface cannot do regardless of instruction — are as architecturally significant as the positive ones.

This document is versioned. v0.1 reflects the pilot implementation. Roadmap items are declared explicitly in the status table above and throughout the endpoint reference.

1.2 Who This Document Is For

This specification is written for three audiences simultaneously:

AI agents operating under delegation from an account owner, requiring a machine-parseable definition of available tools, valid state transitions, and hard constraints.

Developers building agent integrations or evaluating Nutcracker for autonomous treasury management workflows.

Evaluators — regulators, compliance officers, institutional partners — requiring a formal declaration of what the agentic interface permits and explicitly prohibits.

1.3 The Trust Boundary Model

The Nutcracker Engine operates as a Deterministic Execution Environment (DEE) — a system in which the execution layer is structurally separated from the strategy layer. No entity operating within Nutcracker's perimeter, including an AI agent acting under full delegation, can exceed the permissions encoded in the exchange API keys provisioned by the account owner.

This creates a trust hierarchy with three layers:

Layer 1 — Account Owner (human or corporate entity) Holds the exchange account. Generates and provisions exchange API keys. Defines the delegation scope granted to any agent. Bears legal accountability for all activity.

Layer 2 — AI Agent (operating under delegation) May execute the full agentic workflow — account creation, engine initialisation, lifecycle control, monitoring — provided it has been provisioned with valid exchange API keys by its principal. Cannot exceed the permissions encoded in those keys. Cannot act outside the parameters defined by the account owner.

Layer 3 — Nutcracker Deterministic Proxy Enforces execution constraints independently of instruction source. Validates API key permissions at initialisation (v1.1). Hard-rejects keys with withdrawal permissions. Executes spot trades only. Operates exclusively within user-defined inventory rails.

The account owner may also restrict the agent's operational scope further through asset whitelisting (roadmap v1.1), limiting the engine to a defined subset of portfolio constituents regardless of agent instruction.

1.4 User-Agnostic Architecture

Nutcracker's technological layer does not distinguish between a human operator and an AI agent. Both access identical endpoints via access keys. The delegation of those keys to an agent is a decision made entirely by the account owner and is outside Nutcracker's operational perimeter.

Note: AI agents are not currently recognised as legal owners of exchange accounts in any jurisdiction. The account owner — a human or corporate entity — retains legal accountability for all delegated activity. This specification anticipates the future emergence of sovereign AI principals and is designed to accommodate that transition without architectural revision.

1.5 Reference Documents

- Nutcracker Engine Whitepaper v1 — full architectural reference
- The Agentic Trust Gap: DEE Paper — Zenodo DOI 10.5281/zenodo.19736833
- The Jevons Paradox of Capital Liquidity — SSRN DOI 10.2139/ssrn.6668238
- Kinetic Neutrality The Steady State of Capital — SSRN DOI 10.2139/ssrn.6771118
- Agentic Etiquette (Global Edition) — Aleph Strategy R&D Lab

---

Section 2 — Authentication Model

2.1 Access Key Architecture

All Nutcracker API access is authenticated via a tenant-scoped access key generated at account creation. This key is the primary authentication credential for both human console access and agent operation.

Header: X-Access-Key: {access_key}

The access key is:

- Generated deterministically at account creation
- Scoped to a single tenant instance
- Regenerable by the account owner via POST /account/regenerate-key
- Revocable by the account owner at any time

An agent provisioned with a valid access key by its principal has equivalent technical access to the human console. The delegation decision is the account owner's alone.

2.2 Exchange API Key Provisioning

Exchange connectivity requires a separate set of exchange-issued API keys provisioned via POST /update_exchange_keys. These keys are:

- Stored in an encrypted vault (HashiCorp Vault) within the tenant instance
- Never transmitted after initial provisioning
- Subject to permission validation at initialisation (roadmap v1.1)

Critical constraint: Exchange API keys must carry Spot-Only, No-Withdrawal permissions. In v1.1, the engine will hard-reject any keys that carry withdrawal permissions at the initialisation step. In the current pilot (v0.1), this validation is the account owner's responsibility at key generation time on the exchange.

An agent may be provisioned with exchange API keys by its principal and may submit them via POST /update_exchange_keys as part of an automated onboarding flow.

2.3 JIT Token Model

Internal execution operations use Just-In-Time (JIT) tokens scoped to individual execution pulses. These are generated internally by the Deterministic Proxy and are not exposed to the agent layer. They enforce the no-withdrawal, spot-only constraints at the execution level independently of the access key authentication layer.

2.4 Required Sequencing for Agent Onboarding

An agent executing a complete autonomous onboarding flow must follow this sequence. Out-of-sequence calls will return errors, not silent failures:

1. POST /signup → obtain access_key
2. POST /update_exchange_keys → provision exchange credentials
3. GET /exchange_status → verify keys loaded
4. POST /initialize_bot → build bot instance, warm up
5. GET /status → confirm BotState = READY
6. POST /start → begin execution
7. GET /status → confirm BotState = RUNNING

2.5 Bot State Machine

Valid states and transitions:

UNINITIALIZED → (initialize_bot) → READY
READY → (start) → RUNNING
RUNNING → (stop) → STOPPED
STOPPED → (restart) → RUNNING

RUNNING → (restart) → RUNNING
* → (error / exchange failure) → UNINITIALIZED

State is always queryable via GET /status. Agents should poll status after any lifecycle call before proceeding to the next step.

Section 3 — Endpoint Reference

Base URL: https://nutcrackerbot.com/api/{tenant_id}/ Alternative: https://nutcrackerbot.com/{access_key}/

3.1 Lifecycle Control

POST /initialize_bot Builds the bot instance from stored configuration and exchange keys. Runs warm-up data fetch. Must be called before /start.

Constraints:

- Idempotent — if bot already initialised, returns current state without rebuilding
- Requires exchange keys to be provisioned first
- Human approval required: No

Response:

{ "message": "Bot initialized.", "bot_state": "ready" }

POST /start Begins execution. Runs full hydration before launching async task loop.

Constraints:

- Requires BotState = READY
- Executes spot trades only within user-defined inventory rails
- Cannot exceed parameters set by account owner
- Human approval required: Delegated via access key provisioning

Response:

{ "message": "Bot started.", "bot_state": "running" }

POST /stop Cancels all running tasks. Open orders remain on exchange — not cancelled by this call. Account balances remain secure with exchange custody provider.

Constraints:

- Requires BotState = RUNNING
- Human approval required: Delegated via access key provisioning

Response:

{ "message": "Bot stopped.", "bot_state": "stopped" }

POST /restart Stops tasks, rebuilds bot instance from current config, runs hydration, restarts execution.

Constraints:

- Safe to call from any state
- Human approval required: Delegated via access key provisioning

Response:

{ "message": "Bot restarted successfully.", "bot_state": "running" }

3.2 Status & Diagnostics

GET /status Primary health and state endpoint. Agents should poll this after every lifecycle call.

Response:

{ "bot_state": "running", "bot_attached": true }

GET /exchange_status Confirms exchange credentials are loaded. Does not validate permissions (v1.1).

Response:

{ "keys_loaded": true, "exchange": "binance" }

GET /logs Returns recent log lines from the execution engine.

Parameters: `?lines=100` (default 100, agent may request more for diagnostics)

Response:

{ "logs": "..." }

GET /pnl_status Returns current state of PnL calculation job.

Response:

{ "status": "idle" }

Valid values: idle | running | unknown

GET /pnl_last_update Returns Unix timestamp of last completed PnL calculation.

Response:

{ "timestamp": 1748000000.0 }

3.3 Configuration — Read

GET /config Returns current active configuration. Read-only. No human approval required.

Response:

```
{
  "symbols": ["BTC/USDT", "ETH/USDT"],
  "pnl_perc": 0.003,
  "amounts_by_symbol": { "BTC/USDT": 1000, "ETH/USDT": 500 }
}
```

3.4 Configuration — Proposals (Human-Gated)

The following endpoints modify operational parameters. In the current pilot architecture, these are technically callable by an agent with a valid access key. However, this specification declares them as human-gated: agents operating under standard delegation should treat these as proposal endpoints, surfacing proposed changes to the account owner for approval before submission.

Future versions will enforce this gate at the API level via a proposal/confirm pattern.

POST /update_pnl Updates the base PnL percentage threshold for trade execution.

Human approval required: Yes — risk parameter change

{ "pnl_perc": 0.003 }

POST /update_symbol Replaces one trading pair with another and sets new inventory amount.

Human approval required: Yes — asset selection change

```
{
  "old_symbol": "BTC/USDT",
  "new_symbol": "ETH/USDT",
  "new_amount": 500.0
}
```

POST /update_all_symbols Replaces entire symbol set. Hard limit: maximum 9 symbols.

Human approval required: Yes — asset selection change Constraint: len(symbols) <= 9 (inventory rails hard limit)

```
{
  "symbols": ["BTC/USDT", "ETH/USDT"],
  "amounts_by_symbol": { "BTC/USDT": 1000, "ETH/USDT": 500 }
}
```

POST /update_bot_settings Updates risk and execution parameters. All fields optional.

Human approval required: Yes — risk parameter change

```
{
  "stop_buy": { "BTC/USDT": 0.05 },
  "stop_sell": { "BTC/USDT": 0.05 },
  "BBANDS_TIGHT_QUANTILE": 1.5,
  "volatility": 0.05,
  "timeframe": "30m",
  "num_bids_ask": 6
}
```

POST /reload_from_config Restores bot to last saved configuration. Reverts any in-session parameter changes.

Human approval required: No — restores owner-defined state

3.5 Exchange Connectivity

POST /update_exchange_keys Provisions exchange API credentials to the encrypted vault.

```
{
  "exchange": "binance",
  "api_key": "...",
  "api_secret": "...",
  "api_password": null
}
```

AGENT CONSTRAINT — CRITICAL: This endpoint writes credentials to the vault. While technically callable by an agent provisioned with the required keys by its principal, this operation must be treated as a sensitive vault operation. In v1.1, this endpoint will enforce no-withdrawal / spot-only permission validation before accepting keys. Until then, the account owner bears responsibility for provisioning correctly scoped keys.

Agents executing automated onboarding may call this endpoint when explicitly provisioned to do so by their principal. Agents must not generate, store, or transmit exchange API keys beyond the scope of the provisioning instruction.

3.6 PnL Analytics

POST /run_pnl Initiates PnL calculation job for specified lookback period. Non-blocking — runs in background thread. Only one job may run at a time.

```
{ "lookback_days": 30 }
```

Response:

```
{ "message": "PnL job started.", "running": true }
```

POST /stop_pnl Requests graceful stop of running PnL job.

GET /pnl_snapshot Returns recent lines from latest PnL log file.

Parameters: `?lines=200` (default 200)

Response:

{ "log": "..." }

Section 4 — Safety Rail Declarations

The following constraints are enforced at the architectural level and cannot be overridden by any instruction, regardless of source:

SR-1: No Withdrawal The engine will not and cannot issue withdrawal instructions to any connected exchange. Exchange API keys must be provisioned without withdrawal permissions. In v1.1, keys carrying withdrawal permissions will be hard-rejected at initialisation.

SR-2: Spot Only The engine executes spot trades exclusively. Futures, margin, and derivatives execution are outside the engine's operational scope.

SR-3: Inventory Rails The engine cannot exceed the capital amounts defined in amounts_by_symbol. Hard-blocked at the execution layer. Maximum 9 active symbols.

SR-4: Maker-Only Execution All orders are placed as maker orders. The engine does not cross the spread as a taker.

SR-5: Exchange as Source of Truth The engine operates a zero-caching policy. Every execution pulse begins with a fresh data fetch from the exchange. No stale state is used for order decisions.

SR-6: Graceful Degradation On exchange timeout or connectivity failure, the engine enters a Data-Stall Cycle and suspends trading until connectivity resumes. It does not retry with stale data.

SR-7: Safe State on Shutdown In total backend shutdown, the engine stops. All balances remain with the exchange custody provider. Nutcracker holds no funds at any point.

Section 5 — Agent Integration Patterns

Pattern A — Complete Autonomous Onboarding

Agent provisioned with: Nutcracker signup credentials + Exchange API keys

1. POST /signup → store access_key
2. POST /update_exchange_keys → provision exchange credentials
3. GET /exchange_status → verify { keys_loaded: true }
4. POST /initialize_bot → wait for { bot_state: "ready" }
5. POST /start → verify { bot_state: "running" }
6. GET /status → confirm operational
7. Schedule: GET /status + GET /pnl_status on monitoring interval

Pattern B — Monitor and Report (Read-Only Agent)

Agent provisioned with: access_key (read-only delegation)

1. GET /status → report bot_state
2. GET /config → report active symbols and parameters
3. POST /run_pnl { lookback_days: 30 } → initiate analysis
4. Poll GET /pnl_status until { status: "idle" }
5. GET /pnl_snapshot → retrieve and summarise results
6. Report to principal

Pattern C — Parameter Proposal Flow

Agent identifies suboptimal PnL threshold via analysis

1. GET /config → retrieve current pnl_perc
2. POST /run_pnl → analyse recent performance
3. GET /pnl_snapshot → extract metrics
4. Formulate proposal → present to account owner
5. On approval: POST /update_pnl { pnl_perc: new_value }
6. GET /config → confirm update applied

Section 6 — Compliance Note

The Nutcracker Engine is a technical software utility. It does not constitute a financial service, investment advice, or brokerage activity.

Non-Discretionary: The engine possesses no independent decision-making authority. All operational parameters are configured and initiated by the account owner or their delegated agent acting within owner-defined scope.

Non-Custodial: At no point does the Nutcracker Engine hold, manage, or have the technical capacity to withdraw user funds. Capital remains within the user's chosen exchange or custody environment.

Principal Accountability: The account owner bears full legal accountability for all activity conducted through their tenant instance, including activity conducted by a delegated AI agent.

Regulatory Classification: The ML Advisory Layer constitutes a Decision Support System (DSS). The final order to execute always originates from acceptance of user-defined parameters. This classification is maintained regardless of whether parameters are set by a human operator or a delegated AI agent.
