Trade Tokens v1.0

A Structured Semantic Representation for Trading and Market Reasoning

Robert Hansen
Universal Semantic Systems (USS)

November 24, 2025

Abstract

Trade Tokens (TrT) are a Universal Semantic Token (UST) family specialized for expressing trading, market structure, portfolio state, and risk constraints inside a Universal Semantic Runtime (USR). This brief defines the core Trade Token vocabulary, structural invariants, and runtime integration pattern with the Universal Semantic Engine (USE) and financial cognition engines (FinCE, and related engines). The objective is to turn informal, text-level trading intent into a deterministic, auditable, and regulator-friendly semantic plan that can be executed and replayed independently of any specific broker or foundation model.

Contents

1 Position in the USS Stack

Trade Tokens live inside the broader Universal Semantic Systems (USS) architecture. The relevant layers are:

- **UST**: The universal schema for semantic tokens. UST defines how units of meaning are represented, typed, and related.
- **UST.Trade**: A domain family within UST that specializes the schema for financial markets, orders, risk, and portfolio reasoning.
- **USR**: The Universal Semantic Runtime. USR enforces invariants over UST graphs, runs semantic type checking, and normalizes meaning before execution.
- **USE**: The Universal Semantic Engine. USE receives validated plans from USR and sequences execution steps, including calls into domain engines.
- **FinCE and related engines**: Cognition engines for financial reasoning, pricing, portfolio analytics, and strategy evaluation.

Informally, the trade path looks like this:

```
Natural language strategy
        |
        v
UST.Trade graph (Trade Tokens)
        |
        v
USR (validation, invariants, normalization)
        |
        v
USE (execution plan + scheduling)
        |
        v
FinCE + broker adapters (orders, fills, reports)
```

The key point is that Trade Tokens are the smallest semantic unit that *must* remain stable if we want deterministic replay, audit, and cross-engine interoperability.

2 Design Goals

Trade Tokens v1.0 are intentionally conservative. They aim to cover the core of systematic trading workflows without encoding every conceivable financial instrument. The design is guided by the following goals.

2.1 Deterministic Semantics

- Given the same Trade Token graph and the same external market data, the runtime should derive the same orders and position changes.
- Any non-determinism (for example, stochastic exploration of position sizes) must be explicitly marked as such and confined to isolated components.

2.2 Auditability and Replay

- Every trading decision, from signal generation to order placement, should be reconstructable from a combination of:
  (a) the Trade Token graph at that time,
  (b) the market data snapshot, and
  (c) the runtime execution log.
- Regulators, auditors, and risk teams should be able to ask "Why did the system place this order?" and receive a semantic explanation that does not depend on opaque neural activations.

2.3 Risk Transparency

- Exposure, leverage, margin usage, and risk limits must be first-class tokens, not implied by scattered configuration.
- Trade Tokens should allow the runtime to answer questions such as:
  - What is the maximum daily loss this strategy will tolerate?
  - What are the per-asset and per-sector exposure caps?
  - Under what conditions does the system forcibly flatten?

2.4 Broker and Venue Neutrality

- TrT is not a broker API. It describes *intent* in abstract semantic terms.
- Broker or exchange specific details live in adapters that project Trade Tokens into concrete order messages (for example, FIX, REST APIs, or proprietary protocols).

2.5 Composability and Reuse

- Complex strategies should be built from composable modules: entries, exits, sizing rules, and risk profiles.
- These modules should be reusable across portfolios and engines without rewriting the semantics.

3 Core Trade Token Vocabulary

This section describes the v1.0 token set. The list is intentionally minimal. Additional tokens can be introduced in future versions as long as they obey the same invariants.

3.1 Market Objects

3.1.1 ASSET

Represents a tradable instrument.

- `symbol`: canonical identifier, such as "AAPL".
- `class`: equity, future, option, crypto, index, and so on.
- `venue_hint`: optional, such as "NASDAQ" or "CME".
- `currency`: reporting currency, for example, USD or EUR.

An implementation may also attach reference data fields, but the core token focuses on identity rather than the full reference data surface.

3.1.2 POSITION

Represents actual or desired holdings.

- `asset_id`: reference to an `ASSET`.
- `side`: long, short, or flat.
- `size`: quantity in natural units (shares, contracts).
- `cost_basis`: average price of the position.
- `realized_pnl`: realized profit and loss for this position.

In many runtimes, `POSITION` tokens are maintained by USE and FinCE rather than generated by language models.

3.2 Order Intents

3.2.1 ORDER_INTENT

Represents a potential change to the portfolio.

- `action`: buy, sell, open, close, reduce, flip.
- `qty`: an amount expressed either:
  - in absolute units (for example, 100 shares), or

  - as a portfolio-relative value (for example, 2% of equity).

- `price_type`: market, limit, stop, stop-limit.
- `price_level`: for limit or stop orders.
- `time_in_force`: day, gtc, ioc, fok.
- `validity_window`: optional start and end timestamps.

3.2.2 ORDER_GROUP

Links intents into a higher-level structure, such as a bracket order.

- `entry_order_ids`: one or more `ORDER_INTENT` ids.
- `take_profit_order_ids`: profit-taking orders.
- `stop_loss_order_ids`: protective orders.
- `linkage`: all-or-none, one-cancels-other, or sequential.

3.3 Risk and Constraint Tokens

3.3.1 RISK_LIMIT

Defines a constraint over some exposure metric.

- `scope`: per-asset, per-sector, portfolio, or strategy.
- `metric`: daily_loss, max_drawdown, beta, var, and so on.
- `threshold`: numeric bound in normalized units.
- `action_on_breach`: block, scale_down, or liquidate.

3.3.2 MARGIN_RULE

Defines leverage constraints.

- `max_leverage`: maximum allowed leverage.
- `eligible_assets`: filter on `ASSET` class or label.
- `haircut_rules`: optional mapping between assets and margin haircuts.

3.4 Strategy Logic Tokens

3.4.1 ENTRY_RULE

Specifies when and how to open or increase a position.

- `condition_id`: reference to a semantic condition token (for example, a signal or indicator event).
- `target_position`: desired `POSITION`.
- `sizing_rule_id`: reference to a `SIZING_RULE`.

3.4.2 EXIT_RULE

Specifies how to reduce or close the position.

- `condition_id`: trigger condition.
- `exit_mode`: flatten, reduce, or reverse.

3.4.3 SIZING_RULE

Formalizes position size calculations.

- `method`: fixed, volatility-target, fraction-of-risk, Kelly-fraction, and so on.
- `params`: typed parameters. For example: target volatility, confidence level, or stop distance.

4 Canonical Strategy Frame

A canonical Trade Token frame for a standalone strategy is as follows.

```
STRATEGY {
  universe:        [ASSET...]
  entry_rules:     [ENTRY_RULE...]
  exit_rules:      [EXIT_RULE...]
  sizing_rules:    [SIZING_RULE...]
  risk_limits:     [RISK_LIMIT...]
  margin_rules:    [MARGIN_RULE...]
  order_templates: [ORDER_GROUP...] (optional)
}
```

Formally, in USS this becomes a typed graph where:

- nodes are token instances (for example, one `ENTRY_RULE`),
- edges are semantic relations (uses, guarded-by, constrained-by),
- global invariants are enforced by USR prior to execution.

5 Invariants and Validation

The core value of a semantic runtime is invariant enforcement. Trade Tokens are designed so that USR can express and mechanize a meaningful set of constraints before USE or FinCE are allowed to act.

5.1 Structural Invariants

- Every `ENTRY_RULE` must reference:
  (a) a valid `ASSET`,
  (b) a valid `SIZING_RULE`.
- Every `EXIT_RULE` must reference an existing or potential `POSITION`.
- Every `ORDER_INTENT` must bind to either:
  (a) an `ENTRY_RULE`, or
  (b) an `EXIT_RULE`, or
  (c) a maintenance process (for example, rebalance or roll).
- There must be no dangling references: ids must resolve inside the strategy frame or imported libraries.

5.2 Risk Invariants

- The implied exposure after applying all `ORDER_INTENT` tokens must respect all active `RISK_LIMIT` constraints.
- Leverage computed from `POSITION` and `MARGIN_RULE` must not exceed `max_leverage`.
- If executing a plan would breach an invariant, USR must:
  (a) mark the plan as invalid,
  (b) prevent USE from issuing real orders,
  (c) emit a structured error back to the operator or engine.

5.3 Temporal Invariants

Trading is inherently time-bound. Trade Tokens capture this via:

- `validity_window` on `ORDER_INTENT`,
- optional start and end times on `STRATEGY`,
- optional review intervals or checkpoints.

USN style runtimes can tag every token with temporal metadata for replay.

6 Runtime Flow

This section describes a typical end-to-end sequence from a high level instruction to orders on an exchange.

6.1 Phase 1: Parsing and Interpretation

1. A user or higher-level agent describes a strategy in natural language, configuration files, or a graphical interface.
2. A language model and semantic parser propose a candidate Trade Token graph.
3. The proposed graph is passed to USR for validation.

6.2 Phase 2: Validation in USR

1. USR checks structural invariants.
2. USR checks risk invariants with respect to current positions and configured limits.
3. If any invariants fail, USR emits a detailed error graph explaining which tokens are in conflict.
4. If all invariants pass, USR writes a normalized, canonical version of the Trade Token graph to the semantic store.

6.3 Phase 3: Execution Planning in USE

1. USE constructs an *ExecutionPlan*:
   - Event triggers (for example, bar close or price cross),
   - Evaluation steps (for example, indicator updates),
   - Decision steps (entry, exit, rebalance),
   - Order emission steps.
2. The plan is expressed entirely in terms of tokens and simple transformations so it can be simulated or dry-run.

6.4 Phase 4: Engine Integration

1. FinCE receives a request such as "evaluate ENTRY_RULE set for this portfolio at time t" and returns derived `ORDER_INTENT` suggestions.
2. USR re-checks invariants if the intents alter risk meaningfully.
3. Broker adapters map validated `ORDER_INTENT` structures into concrete API calls.

6.5 Phase 5: Logging and Replay

1. Every step emits a semantic log entry:
   - which rule fired,
   - which positions changed,
   - which invariants were checked.
2. To replay a day, the system re-applies:
   (a) the original Trade Token frame,
   (b) the recorded market data,
   (c) the recorded decisions.

7 Worked Examples

This section illustrates how Trade Tokens encode different trading styles.

7.1 Example 1: Simple Trend-Following Strategy

Universe: one equity, AAPL. The strategy buys when price crosses above the 50 day moving average and exits on a 5% drawdown from peak.

7.1.1 Universe and Positions

```
ASSET A1:
  symbol = "AAPL"
  class  = "equity"
```

7.1.2 Sizing Rule

```
SIZING_RULE SZ1:
  method = "vol_target"
  params = {
    target_vol = 0.15,      // annualized
    lookback   = 20         // days
  }
```

7.1.3 Entry and Exit Rules

```
ENTRY_RULE ER1:
  condition_id     = "close_above_50dma"
  target_position  = POSITION(A1, side="long", size="2%_equity")
  sizing_rule_id   = "SZ1"
```

```
EXIT_RULE XR1:
  condition_id     = "drawdown_5pct"
  exit_mode        = "flatten"
```

7.1.4 Risk Limits

```
RISK_LIMIT RL1:
  scope            = "portfolio"
  metric           = "daily_loss"
  threshold        = 0.02   // 2% of equity
  action_on_breach = "block_new_trades"
```

From these tokens, the runtime can:

- derive position size using the volatility targeting rule,
- compute implied risk relative to RL1,
- generate a canonical execution plan that is independent of broker.

7.2 Example 2: Multi-Asset Portfolio with Sector Caps

Consider a portfolio that trades a basket of technology stocks but wants to limit sector exposure and overall leverage.

7.2.1 Universe

```
ASSET A1: symbol = "AAPL", class = "equity", sector = "tech"
ASSET A2: symbol = "MSFT", class = "equity", sector = "tech"
ASSET A3: symbol = "NVDA", class = "equity", sector = "tech"
```

7.2.2 Risk Limits

```
RISK_LIMIT RL2:
  scope     = "sector:tech"
  metric    = "gross_exposure"
  threshold = 0.50   // 50% of equity

MARGIN_RULE MR1:
  max_leverage   = 1.5
  eligible_assets = ["equity"]
```

Any candidate plan that pushes technology exposure above 50% of equity will fail validation, regardless of individual symbol signals.

7.3 Example 3: Options Strategy Skeleton

The v1.0 schema is primarily spot-oriented, but it can represent options strategies at a coarse level.

```
ASSET O1:
  symbol   = "AAPL_2025_01_17_C_200"
  class    = "option"
  underlier_id = A1
```

The semantics of payoff and greeks can be delegated to FinCE, while Trade Tokens still express:

- the desired position in the option,
- the entry and exit rules,
- risk limits at the portfolio level (for example, vega or gamma caps).

8 Failure Modes and Safety Considerations

No representation is free from failure modes. This section outlines the main risks specific to Trade Tokens and suggests mitigation patterns.

8.1 Model Misinterpretation

A language model might:

- misread a human instruction,
- generate an inconsistent set of tokens,
- omit critical risk constraints.

Mitigation strategies:

- require USR validation before any plan is marked executable,
- require human review for plans that change risk parameters,
- maintain libraries of vetted strategy templates.

8.2 Semantic Drift

Over time, organizations may change how they interpret certain metrics or limits. Trade Tokens can address this by:

- versioning token schemas,
- tagging strategies with the schema version they rely on,
- providing migration tools that translate old graphs into new ones.

8.3 Execution Environment Mismatch

A strategy may be valid in abstract but incompatible with a specific broker (for example, differences in allowed order types). Broker adapters can handle this if they:

- expose their capabilities as semantic profiles,
- let USR check that a plan is compatible with the chosen venue,
- downgrade or restructure orders only with explicit operator consent.

9 Relationship to Other UST Families

Trade Tokens do not exist in isolation.

9.1 Semantic Tokens (ST)

General Semantic Tokens represent facts, entities, and events in the world. For trading, they can encode:

- macroeconomic releases,
- earnings announcements,
- news sentiment or classification.

Trade Tokens can depend on ST events as triggers or features without redefining them.

9.2 Teleo Tokens (TT)

Teleo Tokens represent goals, plans, and teleogenic structure at a more general level. A trading teleo structure might say:

> "Maximize risk-adjusted return while maintaining drawdown below some bound, subject to liquidity constraints."

Trade Tokens are a concrete instantiation of those goals for the financial domain. They refine high-level teleogenic structure into position-level actions.

10 Implementation Notes

While this brief is implementation-neutral, a reference implementation would likely include:

- a strongly typed schema (for example, in JSON Schema or Protocol Buffers),
- canonical serialization rules for hashes and signatures,
- a validation library tied into USR,
- adapters for at least one paper trading and one live trading environment.

10.1 Type System Considerations

A minimal type system for Trade Tokens should distinguish:

- nominal types (for example, asset identifiers),
- numeric measures with units (price, volatility, exposure),
- boolean conditions (for rules),
- enumerations (for action kinds, sides, and so on).

USN style runtimes can layer additional invariants on top, such as unit consistency (for example, do not mix dollar and euro exposures).

10.2 Performance Considerations

Semantic validation and plan construction must remain tractable.

- The typical strategy graph is small enough that validation cost is negligible relative to market data ingestion and backtesting.
- For large portfolios with thousands of assets, validation should be incremental: only the modified subgraph is re-checked.

Conclusion

Trade Tokens v1.0 define a compact yet expressive vocabulary for encoding trading intent in semantic form. By placing them within the Universal Semantic Runtime and Engine stack, USS turns fuzzy natural-language strategies into deterministic, auditable, and broker-neutral execution plans. Future versions will expand coverage for derivatives, cross-venue routing, and regulatory reporting, but the core principle remains:

> Trading decisions should be expressed and reasoned about as structured meaning, not as opaque sequences of text tokens.