DATA-LINK-1-FU2 — Read-Only Recon Report

Bericht erstellt: **2026-05-16** · Repository: `steve-tradingbot` · Scope: *read-only* Recon, kein Code · Operator: KW Baustoffe · Branch: `master`

> **Status:** `MULTI_STRATEGY_ENABLED=false` live; Risiko aktuell **0**. HARTE Stop-Regel: `MULTI_STRATEGY_ENABLED=true` bleibt NO-GO, bis DATA-LINK-1-FU2 implementiert und verifiziert ist.

Inhalt

1. Executive Summary

Architektur-Befund: Es gibt keine separate `MultiStrategyManager`-Klasse. Die Orchestrierung ist `main.py:run_scan_cycle` → `MultiStrategyRunner.run()` → `_evaluate_symbol()` → `_attempt_execution()`. Der Runner IST der Manager.

Problem-Surface-Größe: kleiner als ursprünglich angenommen. Der MS-Pfad ist *Initial-BUY only* (DCA + Pyramid hart blockiert, `multi_strategy_runner.py:320-323`). Es gibt **genau eine** `execute_buy`-Call-Site (`multi_strategy_runner.py:381-394`) — nicht 4–6 wie im Main-Scan.

Decision-Dict ist reich strukturiert: liefert `strategy_id`, `metadata`, `reject_reason`, `final_score`, `threshold`, `regime`, `base_regime`, `entry/sl/tp` direkt aus `decision_engine._build_decision`. → 1:1-Mapping auf `emit_decision`-kwargs möglich, kein Mock notwendig.

> **Empfehlung:** GO für Option B minimal. Real-strategy_id + decision_id propagation + ein einziger `emit_decision(action='buy')`-Call beim finalen BUY. Kein Reject-Logging in FU2 (eigene Phase). Kein Telegram. **~15–25 LoC Net-Change** in einem File.

2. MS-Runner-Dateien (Phase 1)

Datei	LoC	Rolle
`trading/strategies/multi_strategy_runner.py`	475	Pipeline-Orchestrator
`trading/strategies/decision_engine.py`	~400	`_build_decision()` (Decision-Dict Schema)
`trading/strategies/router.py`	—	Regime → active_strategies-Liste
`trading/strategies/registry.py`	—	`STRATEGY_REGISTRY` Map: 5 Strategien
`trading/execution/multi_strategy_position_sizer.py`	322	Position-Sizing
`trading/execution/decision_id.py`	—	`generate_decision_id()` (modulweit verfügbar)
`trading/main.py:1165-1180`	—	Einziger Caller (`run_scan_cycle`)

STRATEGY_REGISTRY enthält 5 Strategien mit string-IDs: `trend_follow`, `breakout`, `mean_reversion`, `vwap_mean_reversion`, `volatility_sweep`.

Tests existieren:

- `test_phase_n7.py` — 20 tests
- `test_phase_n8.py` — 26 tests
- `test_n8_1_balance_guard.py` — 24 tests
- `test_g10_4_1_etappe_2_migration.py`
- `test_t_split_2_emitter_wiring.py`
- `test_g1_fu_1a_safety_check_decision_id_imports.py`

~90+ MS-Tests insgesamt; etabliertes `_MockTrader`-Pattern mit `execute_buy_calls = []` Captures.

Feature-Flag live: `MULTI_STRATEGY_ENABLED=false` in `.env-runtime/bot.env:95`. Default in settings: `false`. `MULTI_STRATEGY_DRY_RUN=true` als zweiter Layer.

Log-Prefix `[MS]` in `multi_strategy_runner.py`: 7 logger-Calls + JSONL-File `/home/node/.openclaw/workspace/trading/logs/multi_strategy.log`.

Profile/Strategien/Kandidaten-Entstehung:

- *Kandidaten:* `main.py:run_scan_cycle` reicht `symbols` + `data_by_symbol` (gleicher Universe wie Main-Scan) an `MultiStrategyRunner.run()`. Kein eigener Scanner für MS.
- *Strategien:* aus `STRATEGY_REGISTRY`, pro Regime via `StrategyRouter.route(regime)` aktiviert.
- *Profile:* existieren NICHT als eigenes Konstrukt — jede Strategie hat `strategy_id`-Klassenattribut, das ist der "Profilname". *STRATEGY-GROUP-CONFIG-1* (Pin) sieht eigenständige Profile vor, ist aber Future-Work.

3. Call-Flow & Datenfluss (Phase 2)

```
main.py:run_scan_cycle()
  └─ if settings.MULTI_STRATEGY_ENABLED:                    # heute false
       └─ MultiStrategyRunner.run(symbols, data_by_symbol)
            └─ for sym in symbols_capped:                   # MAX_SYMBOLS-Cap
                 └─ _evaluate_symbol(symbol, df, timeframe, …)
                      ├─ regime = market_regime.detect_extended(df)
                      ├─ router_result = router.route(regime)
                      ├─ active_ids = router_result['active_strategies']
                      └─ for sid in active_ids:             # 1..5 pro Symbol
                           ├─ strategy = STRATEGY_REGISTRY[sid]()
                           ├─ strategy_result = strategy.score_signal(df, regime)
                           ├─ decision = decision_engine.evaluate(strategy_result, …)
                           │     → {decision: TRADE_CANDIDATE|REJECT, strategy_id, …}
                           ├─ if decision == TRADE_CANDIDATE and execution_allowed:
                           │     _attempt_execution(decision, df, regime)
                           │       ├─ N7.2 safety check
                           │       ├─ DCA/Pyramid-Gate (immer fail-Pfad)
                           │       ├─ doppel-position-check
                           │       ├─ portfolio = trader.get_portfolio()
                           │       ├─ N8.1 balance guard
                           │       ├─ position = position_sizer.calculate(decision, …)
                           │       └─ trade = trader.execute_buy(            # ⚠ HIER
                           │                    symbol=…, price=decision['entry'],
                           │                    quantity=position['quantity'],
                           │                    stop_loss=…, take_profit=…,
                           │                    allow_add=False,
                           │                    strategy_group='t1_core',    # ✓ schon da
                           │                    # ✗ kein decision_id=
                           │                    # ✗ kein strategy_id=
                           │                  )
                           └─ _log_decision(…)               # JSONL only, kein emit_decision
```

Datenpunkte am execute_buy

Feld	Quelle	Verfügbar?
`symbol`	scan	✓
`price`	`decision['entry']`	✓
`quantity`	`position['quantity']`	✓
`stop_loss / take_profit`	`decision[*]`	✓
`strategy_group='t1_core'`	T-SPLIT-2 hardcoded	✓
`decision['strategy_id']`	STRATEGY_REGISTRY	✓ **real value**
`decision['final_score'] / ['threshold']`	decision_engine	✓
`decision['metadata']`	decision_engine	✓
`decision_id`	—	✗ **muss generiert werden**

DCA/Add im MS-Pfad: NEIN. `multi_strategy_runner.py:320-323` blockiert `ENABLE_DCA_IN_MULTI_STRATEGY` und `ENABLE_PYRAMID_IN_MULTI_STRATEGY` hart als reject-Reason. MS ist **Initial-BUY only**.

4. Entscheidungsvorschläge Q1–Q6 (Phasen 3–7)

Q1 — Decision-Logging-Scope → Option B (final BUY only)

Aspekt	A (alles)	B (nur buy) ✓	C (eval+buy)
Aufwand	~50 LoC	**~15-20 LoC**	~35 LoC
Risiko	mittel (12 reject-Pfade mappen)	**niedrig**	mittel
Datenqualität	maximal	**Match: jeder echte MS-Trade hat decision_logs-Row**	gut
Scope-Creep	hoch	**gering**	mittel
Testbarkeit	hoch (12 Pfade × Tests)	**hoch (1 Pfad)**	mittel
FU2-Ziel-Fit	Overkill	**exakt: HARTE Stop-Regel erfüllt**	partial

Rejects + Evaluating **NICHT** in FU2. Begründung:

- 12 reject-Pfade im MS-Code (router-no-strategy, regime-detection-fail, routing-fail, strategy-not-in-registry, strategy_id-mismatch, decision-engine-internal-rejects, N7.2-block, DCA/Pyramid-block, doppel-position, portfolio-fail, balance-guard-fail, position-sizer-fail).
- Jeder Pfad hat eigene metadata-Anforderungen → großes Mapping-Sheet.
- Rejects haben *keine* decision_id-Konsequenz (sie produzieren keine trade_logs-Row) → kein Foreign-Key-Risiko, kein Cutover-Blocker.
- → Eigene Phase **MS-DECISION-OBSERVABILITY-1** P2 (Backlog post-FU2).

Q2 — strategy_id-Quelle → Echter Strategy-Name aus STRATEGY_REGISTRY

`decision['strategy_id']` enthält bereits einen aus 5 Werten: `'trend_follow'` / `'breakout'` / `'mean_reversion'` / `'vwap_mean_reversion'` / `'volatility_sweep'`. Direkt durchreichen.

Kein None-Fallback nötig — wenn der MS-Runner buy-execute erreicht, ist `strategy_id` garantiert gesetzt (decision_engine setzt es immer).

Kein erfundener Identifier wie `MS:<profile>:<symbol>`. Operator-Decision D3 aus DATA-LINK-1 ("kein Mock") bleibt konsistent.

Q3 — decision_id-Erzeugung → In `_attempt_execution` direkt nach den Safety-Gates

```
_attempt_execution(decision, df, regime):
    [alle Gates: N7.2, DCA/Pyramid, doppel-pos, balance, sizer]
    ↓ (alle pass)
    decision_log_id = generate_decision_id()
    emit_decision(action='buy', decision_id=decision_log_id,
                  strategy_id=…, …)
    trade = trader.execute_buy(decision_id=decision_log_id,
                               strategy_id=…, …)
```

Begründung:

- Caller (`_evaluate_symbol`) bekommt `decision_log_id` über kein direktes Channel zurück (nur als Teil des trade-dict in `_log_decision`).
- Generierung **erst nach** allen Gates: sonst hätten gates-failed Pfade verschwendete IDs.
- Symmetrisch zu `main.py:743` (`_decision_log_id = generate_decision_id()` direkt vor approved-buy-emit + execute_buy).

Q4 — Telegram-Notif → NICHT in FU2, Backlog `MS-NOTIFIER-1` P2

Befund: Kein Reporter-Call im gesamten `multi_strategy_runner.py`. Es gibt *keinen* Pyramid-/Erstkauf-Notifier-Hook im MS-Pfad. Anbindung wäre eigene Phase (Reporter-API-Mapping, Subkind-Decision, Anti-Spam-Key).

Q5 — Backfill → Kein Backfill, bestätigt

`trade_logs.decision_id` für historische `[MS]`-Trades: 0 betroffen (Flag heute false, 0 `[MS]`-log-Entries in 24h). Frische Tabula-Rasa ab Flag-Aktivierung.

Q6 — Teststrategie → Runtime-Tests mit `_MockTrader` + 1 AST-Guard

Existierendes Pattern in `test_phase_n7.py:82-105`:

```
class _MockTrader:
    state = {'cash': 10000.0, 'positions': {}}
    execute_buy_calls = []
    def execute_buy(self, symbol, price, quantity, stop_loss,
                    take_profit, allow_add=False, **kw):
        self.execute_buy_calls.append({...kw...})
```

→ Direkt erweiterbar zur Capture für `decision_id` / `strategy_id`-kwargs.

Test-Suite-Vorschlag (`test_data_link_1_fu2.py`)

#	Test	Pattern	Assert
1	`test_ms_final_buy_generates_decision_id`	runtime + mock	`execute_buy_calls[0]['decision_id']` non-empty string
2	`test_ms_emit_decision_receives_same_decision_id`	mock emit_decision	beide Calls identische `decision_id`
3	`test_ms_propagates_real_strategy_id`	runtime	`execute_buy_calls[0]['strategy_id']` in {5 Strategien}
4	`test_ms_propagates_strategy_group_t1_core`	runtime	`strategy_group == 't1_core'`
5	`test_ms_emit_decision_action_buy_only`	mock emit_decision	exactly 1 emit_decision mit `action='buy'`, 0 rejects/evaluating
6	`test_ms_disabled_flag_no_emit_decision`	runtime, `ENABLED=false`	0 emit_decision-Calls
7	`test_ms_dry_run_no_emit_decision`	runtime, `DRY_RUN=true`	0 emit_decision-Calls
8	`test_ast_decision_id_passed_to_execute_buy`	AST	`_attempt_execution` body enthält `decision_id=` kwarg

Regression-Guards: `test_phase_n7`, `test_phase_n8`, `test_n8_1_balance_guard`, `test_t_split_2_emitter_wiring`, `test_data_link_1`, `test_decision_log_metadata_emit`, `test_label_1`, `test_trade_log_sl_tp_emit`.

5. Minimaler Implementierungsplan

1 File-Touch: `trading/strategies/multi_strategy_runner.py`

Edits

1. **Imports** (top): `from db_emitter import emit_decision` + `from execution.decision_id import generate_decision_id`
2. **In** `_attempt_execution`, direkt vor `trader.execute_buy(...)` (Z. 381):

```
decision_log_id = generate_decision_id()
emit_decision(
    symbol=symbol, action='buy',
    decision_id=decision_log_id,
    strategy_id=decision.get('strategy_id'),
    score=decision.get('signal_score'),
    threshold=decision.get('threshold'),
    combined_score=decision.get('final_score'),
    regime=regime.get('regime'),
    base_regime=regime.get('base_regime'),
    regime_confidence=regime.get('confidence'),
    strategy_group='t1_core',
    environment=_G61_ENVIRONMENT,
    mode='paper',
    source='multi_strategy_runner',
    metadata={
        **(decision.get('metadata') or {}),
        'rr':              decision.get('risk_reward'),
        'strategy_score':  decision.get('strategy_score'),
        'regime_fit':      decision.get('regime_fit'),
        'risk_score':      decision.get('risk_score'),
    },
)
trade = trader.execute_buy(
    symbol=symbol,
    price=decision['entry'],
    quantity=position['quantity'],
    stop_loss=decision['stop_loss'],
    take_profit=decision['take_profit'],
    allow_add=False,
    strategy_group='t1_core',
    decision_id=decision_log_id,                    # NEU
    strategy_id=decision.get('strategy_id'),        # NEU
)
```

3. **Net change**: ~15-25 LoC (kommentierte Erklärung dazu).

Tests neu: `trading/tests/test_data_link_1_fu2.py` (8 tests, ~150 LoC).

SOT-1d Cutover: Build + Recreate + 3-Way MD5 (`multi_strategy_runner.py`).

Live-Verify: stochastisch unmöglich solange `MULTI_STRATEGY_ENABLED=false`. **Struktureller Verify** via 3-Way MD5 + Tests genügt. Funktionaler Verify erfolgt beim ersten manuellen Flag-Test (separate Operator-Decision).

6. Testplan (Phase 8)

Siehe Q6-Tabelle. Vor Cutover: alle 8 neuen + ~90 existierende MS-Tests grün. Nach Cutover: struktureller MD5-Match + Bot bleibt healthy (Flag=false ändert sich nicht).

7. Risiken

Risiko	Wahrscheinlichkeit	Impact	Mitigation
Import-Cycle `db_emitter` ↔ `multi_strategy_runner`	niedrig	mittel	db_emitter wird heute schon von main.py importiert; analoger Import — kein Zyklus
emit_decision-Stub im Test bricht beim Patch	niedrig	niedrig	bewährtes Mock-Pattern aus `test_t_split_2_emitter_wiring`
Flag-Aktivierung versehentlich durch FU2 ausgelöst	niedrig	hoch	Tests #6 und #7 (Flag-Schutz). Flag-Aktivierung bleibt eigener Step nach FU2
MS-Runner schreibt jetzt Postgres + JSONL (Duplikation)	gering	sehr niedrig	bewusste Übergangslösung; JSONL-Deprecation wäre Option C / separater Backlog
`decision['signal_score']` in N7 hardcoded 5.0/7.0	mittel	niedrig	dokumentieren in commit message + metadata-Felder als forensische Quelle
Foreign-Key `trade_logs.decision_id → decision_logs.decision_id` könnte Reihenfolge erfordern	niedrig	mittel	**emit_decision MUSS vor execute_buy** (Plan respektiert das)

Boundary-Check: 0× DB-Migration, 0× DB-Writes über das hinaus was main.py heute schon tut, 0× historische Mutations, 0× Strategy-Parameter, 0× Bot-/Worker-Restart vor SOT-1d, 0× Mainnet, 0× Push, 0× Secrets, 0× env dump.

8. GO/NO-GO Empfehlung

> **GO für DATA-LINK-1-FU2 Implementation (Option B minimal).**

Begründung:

- Scope klein und scharf umrissen (1 File, ~20 LoC, 8 Tests).
- Risiken-Profil niedrig (Flag bleibt `false`, kein Live-Impact).
- Erfüllt HARTE Stop-Regel: nach FU2-Closure ist `MULTI_STRATEGY_ENABLED=true` cutover-fähig.
- Lässt Raum für spätere Phasen (`MS-DECISION-OBSERVABILITY-1` für Rejects/Evaluating, `MS-NOTIFIER-1` für Telegram, Option C falls JSONL-Deprecation gewünscht).
- Konsistent mit existierenden Patterns (DATA-LINK-1, T-SPLIT-2, DECISION-LOG-METADATA-EMIT).

Voraussetzungen vor Implementation:

- Operator-Bestätigung Q1=B, Q2=real-id, Q3=in `_attempt_execution`, Q4=defer, Q5=no backfill, Q6=runtime+1 AST.
- Operator-Bestätigung kein Telegram in dieser Phase.
- Backlog-Pins für `MS-DECISION-OBSERVABILITY-1` + `MS-NOTIFIER-1` + `STRATEGY-GROUP-CONFIG-1` (existiert schon).

9. STOP / Operator-Decisions

Operator-Decision erforderlich:

- Bestätigung Option B?
- Abweichungen in Q1–Q6?
- GO für Implementation oder weitere Recon-Iteration?

Bericht generiert read-only auf Basis von: `multi_strategy_runner.py`, `decision_engine.py`, `router.py`, `registry.py`,

`multi_strategy_position_sizer.py`, `main.py`, `settings.py`, `tests/test_phase_n7.py`, `tests/test_phase_n8.py`, `tests/test_n8_1_balance_guard.py`, `tests/test_t_split_2_emitter_wiring.py`. Keine Edits, keine DB-Writes, keine Bot-Mutations.