ttTrader Protocol

The wire contract between ttTrader and ttTraderDash - v3 as built, v4 as specified

ttTrader

Generated 2026-10-15

Contents

I Part I - The contract **3**

1 FE/BE dashboard protocol v4 — design spec **4**
- 1.1 1. Thesis: semantics first, encoding second . . . 4
- 1.2 2. Design principles . . . 5
- 1.3 3. Answers to the FE's open questions . . . 5
- 1.4 4. Protocol shape (v4 spec) . . . 6
  - 1.4.1 4.1 Handshake . . . 6
  - 1.4.2 4.2 Request/response (all correlated, all answered) . . . 9
  - 1.4.3 4.2.1 `about` — module version information . . . 10
  - 1.4.4 4.2.2 `error` — the request-scoped / connection-level split . . . 15
  - 1.4.5 4.2.3 `candles` — the S2b candle query . . . 15
  - 1.4.6 4.3 Push (only for subscribed categories) . . . 16
  - 1.4.7 4.4 Reliability rules . . . 17
  - 1.4.8 4.5 Commands (G) . . . 18
- 1.5 5. State semantics (the part that kills phantom orders) . . . 18
  - 1.5.1 5.1 `rev` is a BE defect first (FE-reported, confirmed) . . . 18
  - 1.5.2 5.2 Fill embedded blocks are an FE dependency (G4) . . . 19
- 1.6 6. What changes on the BE side . . . 19
  - 1.6.1 6.1 Sequencing . . . 19
  - 1.6.2 6.1.1 S2 slices (delivery order) . . . 20
  - 1.6.3 6.2 Rollout mechanics . . . 21
- 1.7 7. Encoding decision . . . 22
- 1.8 8. Migration (I) . . . 22
- 1.9 9. Acceptance tests (BE side) . . . 22
- 1.10 10. Review-item status (FE sign-off folded in) . . . 23
- 1.11 11. Implementation addendum (2026-10-15) — S2a/S2b shipped, session block and `q` channel live 25

2 Frontend wire contract **27**
- 2.1 1. Does the protocol change? . . . 27
- 2.2 2. Transport and envelope . . . 27
- 2.3 3. Sections and their triggers . . . 28
- 2.4 4. The id scheme (normative) . . . 28
- 2.5 5. Field tables (generated against the `toJson` sources) . . . 29
  - 2.5.1 5.1 Order — `orderInfo_s::toJson` (section `order`, snapshot `orders[]`) . . . 29
  - 2.5.2 5.2 Execution / fill — `executionInfo_s::toJson` (section `fills`, snapshot `fills[]`) . . . 30
  - 2.5.3 5.3 Position — `positionInfo_s::toJson` (snapshot `positions[]`, embedded `pos`) . . . 31
  - 2.5.4 5.4 Snapshot `states` . . . 32
  - 2.5.5 5.5 Algo — `algoInfo_s::toJson` (snapshot `algos[]`, delta `algos`) . . . 32
- 2.6 6. Delivery and dedupe contract . . . 32
- 2.7 7. Migration (number -> string) . . . 33
- 2.8 8. Worked examples . . . 33
- 2.9 9. Evidence tooling . . . 34

2.10 10. v4 S1 additions (reliability floor) . . . 34
2.10.1 10.1 `seq` semantics (normative) . . . 34
2.10.2 10.2 Handshake: `welcome`, `hello`, `resumed` . . . 35
2.10.3 10.3 `resync`, snapshots and storm control . . . 36
2.10.4 10.4 Order and position revisions . . . 37
2.10.5 10.5 Heartbeat and backpressure . . . 37
2.11 11. v4 implementation state (2026-10-15) . . . 37
2.11.1 11.1 `caps` as emitted . . . 37
2.11.2 11.2 The `query` surface (S2a + S2b, as implemented) . . . 38
2.11.3 11.3 The session block (additive, `sessionBlock` cap) . . . 38
2.11.4 11.4 The `q`/`qr` pull channel (additive, `queryOps` cap) . . . 38
2.11.5 11.5 `sub` as built (S3-interim) . . . 39
2.11.6 11.6 Push and resync addendum (additive) . . . 39
2.12 12. Reserved v4 message types . . . 39

II Part II - The BE side **40**

3 Notification Pipeline — Private Trading & Account Data **41**
3.1 1. Purpose . . . 41
3.2 2. Canonical flow . . . 42
3.3 3. Event contract (ownership table) . . . 43
3.4 4. Processing rules per category . . . 44
3.4.1 4.1 Order reconciliation . . . 44
3.4.2 4.2 Executions -> fills -> local positions . . . 44
3.4.3 4.3 Wallet / balance + margin . . . 44
3.4.4 4.4 Emulated modify (cancel + replace) . . . 44
3.4.5 4.5 Frontend wire payloads (`EEV_2WSS_*` -> websocket) . . . 46
3.5 5. Capability steering . . . 47
3.6 6. Persistence & recovery (event sourcing) . . . 47
3.7 7. Failure handling summary . . . 48

4 BE reply to the FE v4 answers (2026-09-21) **49**
4.1 1. Adopted as-is . . . 49
4.2 2. The `rev` prerequisite — confirmed, and it is worse than reported . . . 50
4.2.1 BE fix required (tracked) . . . 50
4.3 3. Aggregates the BE will own (answers FE §3) . . . 51
4.4 4. Still open . . . 51
4.5 BE implementation addendum (2026-10-11) — money semantics, timestamps, tac-less states . . . 51
4.5.1 Fee semantics per payload (item F) . . . 51
4.5.2 Timestamp semantics per field (item G) . . . 52
4.5.3 tac-less states (whitelist) . . . 52
4.5.4 sim latency note . . . 52

Part I

Part I - The contract

Chapter 1

FE/BE dashboard protocol v4 — design spec

Status: **v4 spec — FROZEN for S1** (2026-09-21). The FE's draft-2 review raised no open design complaints; every review item (A1–A10, the `batchHz`-under-one-package clarification, v4 detection, and per-stage fixtures) is resolved in this revision — see §10 for the item-by-item table. Implementation has since passed S1: S2a (entity queries) and S2b (candles, trades) are shipped, the S3-interim `sub` acknowledgement is live, and a session-anchored aggregate block plus a second `q`/`qr` query channel landed as additive surfaces — the dated addendum at the end of this document records the implementation state; the frozen S1 body is unchanged. S2/S3 field shapes are also frozen, so the FE can build its handshake client and fixtures in parallel.

Inputs: `docs/01-system-protocol/frontend-wire-contract.md` (v3 as built), `W:\development\ttTraderDash\docs\f` `protocol-requirements.md` (FE requirements + two review rounds), `W:\development\ttTraderDash\docs\v4-protocol.md` (earlier FE proposal), and the failure evidence from the 2026-09-20/21 live sessions.

Scope: the **dashboard WebSocket only** (`ttTrader <-> ttTraderDash`). Not the venue-facing path.

1.1 1. Thesis: semantics first, encoding second

The FE's requirements doc reaches the right conclusion, and it matches what this side found independently: **the reliability bugs are protocol-semantics bugs, not encoding bugs.** Every failure hit in production was one of these three:

symptom (observed)	root cause	fix class
6 -> 70 phantom ACTIVE orders; working-order count never settles	push-only delivery, no authoritative live set, no resync; a missed terminal frame is permanent	A1–A5, E3
`modify rejected` order stayed ACTIVE ~50 min, 7 identical ETH stops	no defined semantics for a rejected operation	E5
ids arriving as JSON numbers -> phantom entities	identity carried as JSON number above 2^53	D6, E1

None of these is fixed by protobuf. All three are fixed by **request/response, authoritative snapshots, explicit revisions, and defined terminal semantics.** Encoding is a second-order, swappable decision (§7).

1.2 2. Design principles

1. **Bootstrap, then deltas, then reconcile.** `hello`/`resync` return an authoritative set per scope; deltas are idempotent; a gap has a defined recovery.
2. **Push is a subscription, not a broadcast.** The FE declares what it wants; the BE never sends a category nobody asked for.
3. **Push cadence is a protocol parameter.** Orders, fills, positions and marketdata all ride the regular transmission packages (§4.3); the cadence comes from `limits.batchHz` so the blotter can ask for full fidelity while a chart asks for less. No category gets a dedicated per-event frame.
4. **Identity is a string, always.** Never a JSON number, never parsed numerically, never reused.
5. **Every request is correlated and always answered** — success or a machine-readable error. Silence is a bug class (F3).
6. **Encoding is a transport detail behind one seam** so JSON and binary can coexist per connection.
7. **Additive and negotiated.** A v3 FE keeps working; the new surface is enabled by version + `caps`.

1.3 3. Answers to the FE's open questions

1. **Can the periodic order list become complete (A3)? What is the ceiling?** *Yes for the live set, and that is the right split.* The WS manager already holds the live orders (`m_vectorOrderActivePtr`) separately from the retired ring (`managerWebSocket.h:145`, capacity `ce_stOrderSize >> 3`). So: **make the periodic/`resync` order block an authoritative snapshot of the LIVE set only** (currently a few orders), and serve *terminal history* through the query API (B1) instead of pushing thousands of retired rows. That satisfies A2/A3 cheaply and bounds the payload by live orders, not lifetime orders. Ceiling: the live set is O(algos × resting orders) ~= tens; retired history is effectively unbounded and belongs in the query path, not the snapshot.
2. **Is history served from memory or recording files? Retention window?** From **memory for the recent window, files for the deep window**. The order/execution journals are already written to disk (`logs/orders`, `logs/executions`) and the reconcile journal exists. Proposal: memory ring for the last N per category (N ~= a few thousand, matching FE retention), file-backed paging beyond it. Retention is then bounded only by disk, and the BE must publish the boundary per category via `caps`/H3 so the FE can say "older than X is not available" rather than showing a spinner.
3. **Does the BE want to own turn/aggregate derivation (B5)? Yes — this is a BE responsibility, not an FE approximation.** Each algo already keeps its own balance sheet and realized PnL, and turns are derived in the BE today (the FE's `useTurnTracker` is a workaround for not receiving them). Owning it here removes the FE's approximation, removes a whole class of FE/BE mismatch, and is cheap because the data is already in hand. The only true limitation is the **reconciliation contract already recorded**: the account figure is per-account session PnL while algo figures are turn-derived, and exact per-algo realized/unrealized

reconciliation requires the BE to expose it per algo. Since we own both sides of the model, expose per-algo realized + unrealized and let the aggregate be exact. Equity time series and markout/VWAP/latency aggregates: also BE, sampled/aggregated server-side (B6) to bound payloads.

4. **Existing C++ serialization library to match, or free choice? Free choice — none is in the stack today.** The BE uses a purpose-built `jsonValue_c` with `toJson` on each payload struct; there is no protobuf/SBE/FlatBuffers dependency. If we go binary, protobuf is the pragmatic pick (first-class C++ + TS via `protobuf-es`), and the `.proto` becomes the single schema both sides generate from. Note the constraint that comes with it: the shared payload structs are an **ABI boundary** across the exe and every plugin DLL, so the wire schema must be generated *from* those structs, not replace them.
5. **Is a resume token (A6) feasible, or is full snapshot on connect the long-term design? Resume is feasible and worth doing, with a defined fallback.** The BE already stamps every frame with a monotonic `seq` and a `sessionId`, and each payload carries `rev`/`timeUpdate`. So `hello{lastSeq, sessionId}` can return either (a) the missed deltas if the BE still holds them in a bounded replay ring, or (b) a full authoritative snapshot when the token is too old or the session changed. Design both paths now; ship (b) first, then add the replay ring. Full-snapshot-on-connect stays as the correctness floor — at ~1,000 orders it is fine, and it stops being fine exactly when the query API lands.

1.4 4. Protocol shape (v4 spec)

1.4.1 4.1 Handshake

The v4 BE sends `welcome` unsolicited on connect. This is the v4 detection signal and is required for rollout: the FE currently sends nothing on connect (it opens the socket and waits for the snapshot), so probing with an unsolicited `hello` would risk a v3 BE rejecting or closing on an unknown inbound frame — breaking the compatibility this plan promises. A first-frame `welcome` means v4; anything else means the FE runs v3 exactly as today. No probing, no timeout heuristic.

```
FE -> { t:"hello",   d:{ v:4, sessionId?, lastSeq?, rid } }     (after welcome; carry-over/reconnect)
BE -> { t:"welcome", d:{ v:4, sessionId, sessionSeqToken, caps:[...], limits:{...},
                  exe:exeIdentity_s, result:{ok,code?,msg?} } }  (unsolicited on connect)
```

`caps` is mandatory (E6/H1). Unknown/unsupported `v` □ explicit `result.ok=false` with a machine-readable code; the FE then either refuses or runs reduced (I1).

Two counters, never the same value. The per-connection gap counter is the **envelope `seq`** (§4.4) and nothing else: the FE seeds its gap baseline from the **first frame's envelope `seq` + 1** and resets that baseline on connect. `welcome` deliberately carries **no `d.seq`** — writing the same counter twice invited the FE to seed from `d.seq` and count phantom gaps on every connect. `welcome` carries **`sessionSeqToken`** instead: the **session-scoped** high-water mark, which is **not** reset on reconnect within the process lifetime. `hello.lastSeq` echoes the **session token** (not the envelope seq); resume replays from it. `envelope seq` and `sessionSeqToken` have different lifetimes and must never be compared, seeded from, or substituted for one another; there is no `nextSeq`.

`exeIdentity_s` — the exe's own build identity, carried in `welcome` so a stale exe is observable on reconnect without issuing a query (§4.2.1.4). This named object is the **single definition**; §4.2.1 references it rather than restating the fields.

```
"exe": { "version": "0.9.0", "build": 8347, "git": "a0cfdf2", "dirty": false,
         "type": "debug", "abi": 4, "fingerprint": "6e3b1c2a9f4d8071" }
```

`abi`/`fingerprint` are included because `abi` alone is insufficient: a stale exe with an unchanged ABI integer otherwise looks healthy, and `fingerprint` is the derived value that catches it. `fingerprint` is a hex string (never a JSON number). `build` is a small integer.

`type` is opaque display text, but it is carried here for a specific operational reason: the fingerprint hashes a **debug/release build-configuration marker**, so a release plugin set built against a debug exe produces a fingerprint difference that is a **real fault**. With `type` in `welcome`, the reconnect-only path can *explain* that mismatch — "exe is debug, plugins are release" — instead of merely flagging it and waiting for an `about` query to diagnose it. It is the one field that turns the banner from a symptom into a cause.

Initial-data ordering (R1 — required, or S4 never fires). The BE **must not** send the initial snapshot immediately after `welcome`. It waits for either `hello` or a **grace period of ~250 ms**, then:

condition	BE response
`hello` with a usable `(sessionId, lastSeq)`	**replay** the missed deltas for every scope the ring covers, **snapshot** for any scope it cannot cover — all-or-nothing per scope — and state which in the response
`hello` with no token	full snapshot for all scopes
`hello` with an unusable token	full snapshot, with the stated reason: `tokenTooOld` | `sessionChanged` | `byteBudget`
no `hello` within the grace period	full snapshot for all scopes (v3-equivalent behaviour)

The static directories ride both paths. The initial baseline always carries the static directories — `algos`, `exchanges`, `instruments` — plus the marketdata and playback blocks, on the `hello` path as well as the grace path. They are a **connection baseline, not a resync scope**: a `hello`-sending client that received only the `orders`/`positions`/`account` scoped live sets would be unable to resolve a single fill, quote or position. The grace path releases them inside the v3-equivalent `t:"snap"`; the `hello` path releases an equivalent `t:"snap"` baseline frame (real envelope `seq`, no `nextSeq`, no entity blocks — the scoped snapshots stay authoritative for the live set) before the per-scope snapshots.

Without this ordering the FE has already received a complete snapshot before its resume request lands, so the replay saves nothing on the common reconnect path and the FE cannot tell whether to expect a snapshot or a replay. The snapshot-on-grace-timeout keeps a v3-style client working exactly as today.

The resume response carries the outcome explicitly, so the FE never has to infer it: `{ t:"resumed", d:{ scopes:[...], replayed:true|false, reason?:"tokenTooOld"|"sessionChanged"|"byteBudget", rid } }`.

The S1 replay ring is absent, so a `hello` that cannot be covered is answered deliberately `replayed:false` — `reason:"sessionChanged"` when the echoed session token names another connection (the normal reconnect case), `reason:"tokenTooOld"` when a same-connection token predates the replay window — followed by the per-scope snapshots. This is the defined S1 behaviour, not an accident; the ring depth is S4.

`caps` entries are **stable identifiers** the FE branches on, never repurposed. The set is **per-build and contains only the surfaces that build implements**: a cap present is a promise, so a surface joins the list with the slice that ships it, never before. In S1/S2a the set is `resume`, `query`, `about`; `candles`/`trades` join with S2b, `turns`/`aggregates` with S2c, and `sub` with S3. (The current build's superset — `resume`, `query`, `queryOps`, `sessionBlock`, `trades`, `candles`, `about`, `sub` — is recorded in the addendum at the end of this document and in `frontend-wire-contract.md` §11.1; a cap present is still a promise the BE answers it, and `turns`/`aggregates` stay out until S2c ships.) (`revOrdering` is a phase-2 capability and is deliberately absent until `markState` ships.)

`resync` is NOT a cap and must not be added as one. `resync` is the S1 gap-recovery primitive and exists in **every** v4 build by construction — the gap rule needs the envelope `seq`, which only exists in v4. `t:"resync"` and its `t:"snapshot"` answer are gated on **v4 detection** (the unsolicited first-frame `welcome`), never on `caps`. `caps` gates only the optional surfaces listed above.

`caps` is the **single gate** for an optionally-supported feature (not for `resync`): a FE hides a surface whose cap is absent, and a cap being present is a promise the BE implements it. `result.ok=false, code:"unsupported"` remains a fallback for an individual request, never the primary signal (§4.2.1).

`limits` (agreed shape — both sides read the same numbers; the FE treats `batchHz` as ceilings):

```
{ "limits": {
  "heartbeatMs": 5000, "staleThresholdMs": 45000,
  "periodicMs": 200,
  "maxPageSize": 500, "maxQueryRangeMs": 604800000,
  "resumeWindowMs": 300000, "resumeByteBudget": 4194304,
  "maxSubscriptions": 32, "maxInstrumentsPerSub": 256,
  "batchHz": { "orders": 5, "fills": 5, "positions": 5, "quotes": 5, "trades": 5,
               "acct": 1, "algos": 1, "instruments": 1, "exchanges": 1 },
  "history": {
    "orders":    { "memoryRows": 50000,  "oldestAvailable": 0 },
    "fills":     { "memoryRows": 50000,  "oldestAvailable": 0 },
    "positions": { "memoryRows": 5000,   "oldestAvailable": 0 },
    "turns":     { "memoryRows": 5000,   "oldestAvailable": 0 },
    "trades":    { "memoryRows": 100000, "oldestAvailable": 0 },
    "equity":    { "maxRangeMs": 604800000, "tiers": [ { "bucketMs": 10000, "maxRangeMs": 10800000
                                                       { "bucketMs": 60000, "maxRangeMs": 604800000
  } } }
```

`batchHz` under one package (semantics decided). There is **one frame cadence**, `limits.periodicMs` (default 200 ms = 5 Hz), and `batchHz` is a **per-category ceiling expressed as a divisor of that cadence**: a category at 1 Hz appears in every 5th frame. The BE **always emits the frame** while any category is due, and **omits a category when it has nothing new** — so a category's absence from a package means **"no change"**, never "not sampled". A category is only ever stale by at most its own `batchHz` interval, never by an unknown amount. Consequently:

- The FE labels freshness from `periodicMs` (frame cadence) plus the category's `batchHz` (A8), not from arrival timing.
- Absence is a no-change signal; the FE must not treat it as a gap (consistent with §4.4 — `seq` advances per *sent frame*, and omission is not a hole).
- `maxHz` on a subscription may **raise** a category above the default where the default is a floor of usefulness rather than a cap on demand: a blotter asking for full fidelity gets `orders/fills/positions` at the frame cadence (`periodicMs`), i.e. the ceiling is the frame rate, and a category can never be served faster than one frame. The BE may **lower** the frame cadence for a connection that asks for less, never faster than `periodicMs`.

Two rules on `history`: `oldestAvailable` is the boundary the FE labels the UI with (beyond it the BE must say "not available", never a spinner) and must be published **per instrument** for candles and trades, where it is more useful than per category; per category is fine for orders/fills/positions/turns. **`oldestAvailable: 0` is a defined sentinel meaning unknown/not yet published** — the FE must show no boundary claim rather than "available from 1970". `heartbeatMs` / `staleThresholdMs` come from here so the FE retires its local 45 s heuristic. Candle interval sets and their depth stay on the **instrument** payload (per instrument); `caps` only says whether the instrument carries them.

Equity series shape (required — a single number per bucket cannot render the account chart): per point { `timestamp, equity, sessionPnL, spot, margin, byCurrency[]` } where each byCurrency entry is { `currency, exchg, exchgId, spot, margin, sessionPnL, marginSessionPnL` }. **Two tiers, published in `limits.history.equity.tiers`** (authoritative — the prose above must not restate them differently): a **10 s tier for ~3 h** (matches the FE's `session` cadence) and a **60 s tier covering 7 d** (matches the FE's day cadence and extends it to the 7 d view). Two tiers is the whole scheme; there is no separate 1 m/5 m pair. The tiers form is preferred over one series plus FE downsampling because it bounds payloads server-side.

Retention window the FE asked for (memory), file-backed paging beyond: orders/fills **24 h or 50k rows** (whichever is smaller), positions/turns **session ~=5k**, trades **24 h or 100k rows for the focused instrument**, equity **per `limits.history.equity.tiers` — the 10 s tier for the recent ~3 h and the 60 s tier covering the 7 d view** (two tiers; the "7 d tier" is the 60 s tier, not a third bucket size), markout/VWAP/latency **7 d, aggregated server-side**. Grounding: the FE's own caps are the bottleneck — 5,000 terminal orders per state against 3,332 cancels observed in ~40 min, so a single quoting session exhausts the FE cap within about an hour.

1.4.2 4.2 Request/response (all correlated, all answered)

```
FE -> { t:"resync",  d:{ scope:[...], rid } }
BE -> { t:"snapshot",d:{ scope, seq, data:{...}, complete:true, rid } }

FE -> { t:"query",   d:{ what:"orders"|"fills"|"positions"|"turns"|"equity"|"markout"
                          |"trades"|"candles"|"about",
                     filter:{ instrument?[] , algo?[] , venue?[] , state?[] , liveOnly? },
                         base?, intervalSec?,          // candles
                         horizonsMs?,                  // markout
                         from?, to?, cursor?, limit?, rid } }
BE -> { t:"queryResult", d:{ rid, items:[...], nextCursor?, complete, total?, hasMore?,
                             oldestAvailable?,   // 0 = unknown (see §4.1)
                             summary?,           // about only - see §4.2.1.3
                             result:{ok,code?,msg?} } }

FE -> { t:"sub",   d:{ category, instruments?[], algos?[], maxHz?, rid } }
BE -> { t:"subAck",d:{ category, state:{...}, rid } }
FE -> { t:"unsub", d:{ category, rid } }  ->  { t:"subAck", d:{...,result} }
```

`snapshot.data` shape per scope (both the initial snapshots and every `resync` answer). Each scope's `data` is the value itself — never wrapped in a scope-named key:

scope	data
`orders`	**array** of full order payloads (`orderInfo_s::toJson(true)`) — the authoritative live set
`positions`	**array** of full position payloads (`positionInfo_s::toJson(true)`)

scope	data
account	**array** of account blocks, exactly the value the v3 snap carries under acct: { id, eid, ccy, dec, balances:[{ ts, spot:{bal,avail,eq,locked,spnl}, mrg:{bal,avail,eq,locked,spnl} }] }

An empty array is a valid authoritative value ("none live" / "no asset has a balance") and is never replaced by an absent key or a wrapper. A client that receives {acct:[...]} or {account:[...]} for the account scope is looking at a contract violation and must fail loudly.

Balance history sampling: the balances array in each account block is a sampled series, not every raw balance event. The BE retains at most one entry per 10 s window per asset (stale or duplicate ts entries are skipped) so the fixed-size history ring spans hours; deltas (update frames) are unsampled and carry every event. Consumers must therefore treat balances as a ~10 s cadence equity series and never rely on it for sub-10 s balance changes.

what: "trades" and what: "candles" are required, not optional: the tape-as-query and per-instrument candle paging are both promised by this plan and had no transport without them. candles is scoped by { instrument, base, intervalSec, from, to }.

Markout horizons are a query parameter (A3). The FE's horizons are fixed at **T+1 s, +5 s, +30 s, +60 s**; the BE publishes exactly those by default and accepts horizonsMs[] to override. An immature horizon is **absent** from the result, never zero-filled — a zero would read as a measured flat move.

Candle field set (A-item 1): per instrument publish the list of series, each with base ("trade"/"midpoint"), intervalSec (numeric), the depth sent on connect, and a per-interval oldestAvailable. Per candle: { time, open, high, low, close, volume, final? } — time = bucket start ms; the FE replaces by time (idempotent); final lets it stop merging into a closed bucket. Interval sets and depth stay on the instrument payload, as agreed. The FE's base/interval **selectors become conditional**: a toggle renders only when the instrument publishes both bases (or more than one interval); otherwise it is a read-only label (A9). The S2b query result carries the same per-candle fields plus the series identity on each item — see §4.2.3 for the exact candles queryResult shape, ordering and cursor behaviour.

Error results always carry rid + a machine-readable code (F3); silence is a bug.

1.4.3 4.2.1 about — module version information

The first real query consumer, and the FE's first query client exercise. Requested once per **connection generation** (after welcome), plus lazily when the About surface mounts — no push.

```
FE -> { t:"query",       d:{ what:"about", rid } }
BE -> { t:"queryResult", d:{ rid, items:[ moduleInfo... ], complete:true,
                             summary:{...}, result:{ok:true} } }
```

about is advertised in caps; a BE that does not support it answers result.ok=false, code:"unsupported" (fallback, not the primary signal — the FE branches on the cap).

Purpose (state this honestly): the table answers "which build is this?" for a run, and the load state makes it a **stale-plugin / partial-deployment detector**. Both jobs matter: reconnects in place on a long-running desk can otherwise silently serve data from a different build, and a stale exe can silently prevent plugins loading at all.

1.4.3.1 4.2.1.1 Per-state field contract (codeable verbatim)

`items` contains **attempted** modules, not only successfully loaded ones — otherwise a deployment fault would render as a *shorter* clean list. Every item guarantees:

field	`loaded`	`rejected`	`missing`
`name`	guaranteed	guaranteed (module path / lib name)	guaranteed
`kind`	guaranteed	guaranteed (from config / scanned dir)	guaranteed
`state`	guaranteed	guaranteed	guaranteed
`abi`	guaranteed	**guaranteed**, except `missingExport`	absent
`fingerprint`	guaranteed	**guaranteed**, except `missingExport`	absent
`version`, `build`, `git`, `dirty`, `built`, `type`	guaranteed	**absent**	absent
`hostAbi`, `hostFingerprint`	absent	guaranteed when the module's own value is unreadable	guaranteed
`rejectReason`	absent	guaranteed	guaranteed (`notFound`)
`rejectDetail`	absent	guaranteed (opaque human text)	guaranteed

Rules:

- **`state` is the only key the FE may require unconditionally** (besides `name`/`kind`). It is stable: `loaded` | `rejected` | `missing`.
- **`state` must be derived from `state` alone — never inferred from the integrity values.** A module can be **rejected while both `abi` and `fingerprint` equal the host's**: rejection is decided by several checks, and a size mismatch leaves the version and fingerprint identical. The delivered fixture contains exactly such a row (`rejected` on a size mismatch with a matching fingerprint), so inferring "loaded" from matching integrity values is **provably wrong**, and any "optimisation" of `summary` that does so will mis-classify a real fault as healthy. Corollary for the banner: it must flag on `state != "loaded"`, and `consistent: false` can arise from non-loaded items **without** any fingerprint conflict.
- **The exe is represented twice, on purpose, and the client must not double-count it.** The exe's identity appears both as `welcome.exe` (the reconnect-cheap path) and as a `kind:"exe"` item in `items` (so the table is complete and sortable). A client comparing fingerprints to find "which module disagrees" must count the exe **once**, as follows:
  - the `welcome.exe` fingerprint is an independent vote **only when the exe is not already represented by a `loaded` `kind:"exe"` item** — otherwise the exe votes through that item;
  - if the exe **is** listed as an item but with `state != "loaded"`, that item carries **no fingerprint of its own** (per the per-state contract above it falls to `hostAbi`/ `hostFingerprint`), so the `welcome.exe` value becomes the exe's vote. The host's identity is still known, so the exe still votes exactly once — it never drops out of the comparison merely because its item load failed.

- **Fault attribution when fingerprints disagree: the majority is the reference, not the exe.** Do **not** compare every module against the exe. In the failure this whole mechanism exists for — the exe did not relink while the plugins were rebuilt — every plugin differs from the stale exe, so an exe-as-reference rule blames all the plugins and never the offender. The reference is the **majority fingerprint across the loaded modules plus the exe** (the exe voting once), with ties resolving to the exe as the host; the odd one out is the fault. When the exe is the minority and is not otherwise listed, report the exe itself as the fault rather than the plugin set.
- **A fingerprint literal is never a contract expectation.** The value is a function of build configuration, so it legitimately differs across machines and configs. The contract is: it is a hex string, and it is **equal across the exe and the plugin set on a consistent deployment** (asserted across `welcome.exe` and `items`). Pinning the literal would be a test defect.
- **`(kind, name)` is unique within one frame.** A module appears exactly once, even when it is reachable through both a `plugins[]` entry and a directory scan — the BE de-duplicates on `(kind, name)` and the FE may assert uniqueness in dev. There is no second row for a duplicate path.
- `abi`/`fingerprint` are emitted for a rejected module whenever the ABI export resolved. They are readable because `ui32AbiVersion` is the **first member** of the versioned `pluginAbiInfo_s`, read before any further layout is trusted (`storePlugin.cpp:42-44`).
- The one case where even `abi` is absent is `missingExport` — a DLL predating the handshake, from which nothing is read. `hostAbi`/`hostFingerprint` are then present instead so the row is still self-describing.
- **Exactly one of the module value / host value is present, per pair** — the invariant that guarantees the mismatch row always renders:

state	abi	hostAbi	fingerprint	hostFingerprint
`loaded`	present	**absent**	present	**absent**
`rejected`	present, *or* absent when `missingExport`	present **iff** `abi` is absent	present, *or* absent when `missingExport`	present **iff** `fingerprint` is absent
`missing`	absent	present	absent	present

  So `rejected` carries exactly one of `abi`/`hostAbi` (and exactly one of `fingerprint`/`hostFingerprint`); `loaded` carries neither host field; `missing` carries the host pair and neither module field.

- Absent means **the key is not emitted**, never null. The FE renders `--` and never fabricates.
- `hostAbi`/`hostFingerprint` are the exe's own values, always known.

```
{ "name": "algoBASIN", "kind": "algo", "state": "loaded",
  "version": "0.9.0", "build": 8347, "git": "a0cfdf2", "dirty": false,
  "built": "2026-09-22T03:48:04Z", "type": "debug",
  "abi": 4, "fingerprint": "6e3b1c2a9f4d8071" }

{ "name": "exchgLighter", "kind": "exchange", "state": "rejected",
  "abi": 3, "fingerprint": "1c9e77aa30b2f045",
  "rejectReason": "abiVersionMismatch", "rejectDetail": "host abi 4, plugin abi 3" }

{ "name": "indADX", "kind": "indicator", "state": "missing",
```

```
  "hostAbi": 4, "hostFingerprint": "6e3b1c2a9f4d8071",
  "rejectReason": "notFound", "rejectDetail": "expected in indicators/, not present" }
```

1.4.3.2 4.2.1.2 Opaque vs branchable (codeable verbatim)

- **Branchable (closed enums):** `state` and `kind`.
  - `state` drives the row renderer and the banner: `loaded` | `rejected` | `missing` (an unknown state is tolerated as `missing`).
  - `kind` drives sorting and About-page grouping: `exe` | `subsystem` | `algo` | `exchange` | `indicator`, ranked in that order and then by `name`. An **unknown `kind` must render verbatim and sort last** (it is a forward-compatibility value, not an error).
- **Opaque display text, rendered verbatim, never branched on:** `type`, `rejectReason`, `rejectDetail`, `name`. These may gain values in any release without a protocol change; a closed set would make every new value a dev-mode parser failure on the FE.
- Machine-readable rejection values are **structured fields** (`abi`, `fingerprint`, `hostAbi`, `hostFingerprint`), not embedded in `rejectDetail` — so no regex is needed to render "plugin abi 3 vs host abi 4".
- `fingerprint` and `hostFingerprint` are **hex strings**, never JSON numbers (opaque 64-bit, same rule as order ids). `abi`, `build` are small integers and are safe as numbers.
- `built` is always ISO-8601 UTC with a `Z` suffix.
- `items` order is **unspecified**; the FE sorts deterministically (kind rank `exe->subsystem->algo->exchange->indicator`, then `name`).
- `about` is **non-paged**: `complete` is always `true` and `nextCursor` is absent.

1.4.3.3 4.2.1.3 `summary` reduction invariant (codeable verbatim)

`summary` is **exactly** the reduction of `items` in the same frame and carries no independent information:

- `expected` = `loaded` + `rejected` + `missing`
- `expected` = `items.length`
- `consistent` = **no** item has `state != "loaded"` **and** each **ABI family** is internally singleton on `fingerprint` (§4.2.1.4.1) — the plugin family (`exe`/`algo`/`exchange`) agrees within itself, the indicator family agrees within itself, and `subsystem` is exempt. The two families legitimately differ and must **not** be compared to each other; a global single-fingerprint rule would report every healthy deployment as inconsistent.

```
"summary": { "expected": 91, "loaded": 90, "rejected": 1, "missing": 0, "consistent": false }
```

- The FE renders the deployment-fault banner from **`items`** and treats `summary` as a dev-mode assertion that the two agree.
- **The banner keys on `consistent` and on any `state != "loaded"` — never on a remembered count.** `expected` legitimately changes when plugins are added to or removed from a deployment while `consistent` stays `true`.
- `expected` counts **config `plugins[]` plus scanned indicator/plugin directories** (indicators are not enumerated in the config). Each absent module becomes its own `missing` item, so the FE never infers the inventory.

1.4.3.4 4.2.1.4 Fingerprint semantics (corrected)

`getPluginAbiFingerprint()` is an FNV-1a hash over the `sizeof/alignof` of every shared struct crossing the boundary, plus build-configuration markers (debug/release, compiler id+version, iterator-debug level, sanitizer) — and **no module identity**. Verified at `pluginAbi.h:79-149`.

Therefore:

- It **is** an ABI-surface hash, and within a **family** every member must be **equal** on it.
- `abi` is the **weaker** check (a hand-set integer that must be remembered); `fingerprint` is the derived one that catches a struct change nobody bumped for.
- The earlier "exe fingerprint may differ normally" framing is **withdrawn** — it was wrong.

There are three families, and comparing across them is a bug (§4.2.1.4.1 below). The single "all modules must be equal" rule is too broad: indicators use a *different* handshake struct, and the subsystem is outside the plugin ABI entirely. Comparing globally flags a healthy deployment.

1.4.3.5 4.2.1.4.1 ABI families (codeable verbatim)

family	members	`abi`	`fingerprint`
plugin	`exe`, `algo`, `exchange`	`4` (`ce_ui32PluginAbiVersion`)	one value per deployment
indicator	`indicator`	`2` (`indicatorApiV2_s::ui32ApiVersion`)	one value per deployment, **different from the plugin family**
none	`subsystem`	**`0`**	**`0000000000000000`** — placeholder, carries no ABI

Rules:

- **Compare `fingerprint` only within a family.** Plugin-family members must agree with each other; indicator-family members must agree with each other; the two families legitimately differ.
- **`subsystem` is exempt from both checks.** `abi: 0` and an all-zero `fingerprint` are the documented placeholder for a module outside the plugin ABI — not a fault, and it must never be counted in a majority vote.
- **The deployment-consistency predicate is therefore per family**, e.g. `pluginFp = {exe} ∪ {algo, exchange items}` must be singleton, `indicatorFp = {indicator items}` must be singleton, and the subsystem contributes nothing. A **stale exe** is still caught, because the exe is in the plugin family with the algos and exchanges.
- **Do not compare the exe against indicators.** They are different families by construction, so a difference there is expected and must not raise the banner.
- `welcome.exe` participates in the **plugin** family only.

`welcome` carries the exe's **`exeIdentity_s`** (§4.1 — the single definition of the fields, not restated here) so a stale exe is observable on reconnect without a query. Under the rule above, that `fingerprint` **is** comparable to the plugin set, and `type` is what lets the FE *explain* a mismatch rather than only flag it — the fingerprint hashes a debug/release marker, so a release plugin set against a debug exe is a real fault whose cause is the `type` difference.

1.4.4 4.2.2 `error` — the request-scoped / connection-level split

`t:"error"` is a **real frame type**, distinct from a failed `queryResult`. It has **two deliberately different emitters**, and the distinction is load-bearing:

shape	`seq`	`rid`	meaning
request-scoped	`> 0` (`++sendCounter`)	present	a failure answering a specific request; correlates by `rid`
connection-level	**`0`**	**absent**	a server-side problem with nothing to correlate — reported alongside the connection, not a request

```
BE -> { seq:0,  t:"error", d:{                  "result":{ok:false,code,msg} } }       (connection-level)
BE -> { seq:37, t:"error", d:{ rid:9, "result":{ok:false,code,msg} } }                  (request-scoped)
```

Rules the client must implement:

- **A `seq: 0` error does not advance the expected-sequence counter; a `seq > 0` error does.** The connection-level emitter deliberately stays outside the ordered stream — giving it a real sequence number would manufacture a spurious gap every time the BE reports a connection-level problem. Do **not** "simplify" the two paths together.
- **A rid-bearing `t:"error"` rejects its pending request**; a rid-less one is logged as connection-level and correlates with nothing.
- **A query is still answered by exactly one `queryResult`** with the same `rid`, success or failure (§4.2). The `t:"error"` handler is defensive for the reserved-`command` surface and the connection-level channel — it is not the primary query error route.
- Because error frames can arrive at `seq: 0`, they must be **matched before any `seq > 0` gate** in the client dispatcher. Gating dispatch on a positive sequence silently drops every connection-level error — which is a real bug we hit: the frame never reached the handler, the error counter stayed blind, and a request-scoped seq-0 error would have waited out the full request timeout.

1.4.5 4.2.3 `candles` — the S2b candle query

`what:"candles"` answers from the BE's retained **finalized** candle series. It is scoped by `{ instrument, base, intervalSec, from, to }`: `filter.instrument` is **required and non-empty** (a series belongs to exactly one instrument, so an unscoped candle query has no meaning), while `base` (`"trade"` | `"midpoint"` | `"exchange"`), `intervalSec` (positive seconds) and `from`/`to` (inclusive open-time bounds, ms) narrow it.

```
FE -> { t:"query", d:{ what:"candles",
                       filter:{ instrument:["instLighterBTC"] },
                      base:"trade", intervalSec:60, from?, to?, cursor?, limit?, rid } }
BE -> { t:"queryResult", d:{ rid,
                             items:[ { time, open, high, low, close, volume, final,
                                       base, intervalSec } ],
                             nextCursor?, complete, total?, hasMore?, oldestAvailable,
                             result:{ok} } }
```

- **`items`** are finalized candles, `time` = bucket start (ms). `open/high/low/close` and `volume` are decimal strings at the instrument's price/size precision. Every item is `final:true` because only finalized bars are published; a still-open bucket is never returned.
- **`base` and `intervalSec` are repeated on every item**, so a request that omits them still gets a self-describing flat array rather than silently merged series. When several series match they are interleaved by `time`, then `base/intervalSec`.
- **`oldestAvailable`** is the per-series retained boundary, the minimum non-zero boundary across the matched series; `0` is the unknown sentinel (§4.1). It is **not** clamped by `from` — it is the store boundary, so the FE can label the UI edge.
- **Paging is supported and is newest-first**: the first page is the newest `limit` bars, items are emitted ascending by `time` within the page, and `nextCursor` anchors on the page's oldest bar so the next page is strictly older. `limit` is clamped to `limits.maxPageSize`. A cursor below the retained boundary answers `result.ok=false, code:"outOfRange"` with `oldestAvailable`, never an empty page.
- **Errors**: a missing/empty `filter.instrument`, an unknown `base`, a non-positive `intervalSec`, a malformed `from/to` (`from > to`), or a malformed `limit` answer `code:"badFilter"`; a malformed or foreign cursor answers `code:"badCursor"`. An instrument with no matching series is **not** an error: it answers `items:[]`, `complete:true`, `oldestAvailable:0`.

1.4.6 4.3 Push (only for subscribed categories)

Revised 2026-09-21 (BE): the special-case immediacy requirement for orders and fills is withdrawn. They ride the **regular transmission packages** like every other category — the periodic push frame — rather than a dedicated immediate frame per event. Consequences:

- **All push categories are batched** into the periodic frame: `order`, `fill`, `fills`, `pos`, `quotes`, `trades`, `acct`, `algos`, `instruments`, `exchanges`.
- **The push cadence is the blotter latency**, so it is a protocol parameter, not a constant: `limits.batchHz` gains `orders/fills/positions` (default ~5 Hz — the FE's existing quote flush cadence — tunable so a blotter can ask for full fidelity). Cadence is 200 ms at 5 Hz, which is well inside human perception for a dashboard blotter.
- **Fill atomicity is preserved:** a fill is emitted in the same package as its embedded `order/pos` blocks, so the FE's atomic merge is unchanged (§5.2). A fill must never be split across packages from its embedded blocks.
- **Event identity still comes from the entity, not the package:** `(id, rev)` (phase 2) or `(id, timeUpdate)` (phase 1) plus the fill's own `id`, so batching introduces no new dedupe rule and replay after `resync` stays idempotent.
- **Ordering within a package** is by the entity's ordering key; across packages by `seq`. A package is the atomic unit for `seq`.
- **Marketdata payload slimmed:** per instrument send **last trade + BBO** plus **candles** (the change you asked for), not an unbounded trade tape. The BE has candles natively now, so the FE's chart no longer needs the tick tape; trades become a *query*, not a push.

1.4.7 4.4 Reliability rules

`seq` semantics (G1 — load-bearing, decided here). `seq` is a **per-connection send counter** and advances **only for frames actually sent on this connection**. Filtering is therefore never a gap: an unsubscribed category produces no frames and consumes no sequence numbers. The FE's gap rule is exactly "a received `seq` is greater than last+1". The alternative (a global sequence the FE must interpret in the light of its subscriptions) is rejected — it makes every filtered frame look like a gap and would cause resync storms.

- **Gap → resync**, and **resync is storm-controlled** (G2): at most **one resync in flight per scope**; requests raised while one is pending are coalesced into it; repeated failures back off (exponential, capped at **10 s**), and after a **documented maximum of consecutive failures** (5) the BE/FE stop retrying silently and **surface the failure to the operator** as a stale-data banner — the terminal state is "resync failed, data is stale", never an infinite silent retry (A7). The BE likewise **coalesces duplicate resyncs for the same scope** in flight rather than answering each — duplicate `rid`s get the same result, not double work.
- **Idempotent replay** (F2): replaying any frame after a resync is a no-op.

Phase-1 ordering, stated precisely (A6). The FE does **not** require ordered delivery. The FE's rule is:

- a **state transition is always applied** — an older cancel still lands on a live order;
- **only same-state updates are compared by `timeUpdate`**, and the newer wins;
- a **terminal state is never modified** (by anything, including a higher `rev`).

So `timeUpdate` is a *tie-break within a state*, not an arrival-order requirement, and the BE must not add ordering guarantees it does not need. Phase 2 replaces the tie-break with `rev`. - **Terminal immutability** (E3): once terminal, an entity is never modified or resurrected — this overrides any higher `rev`. - **Rejected-operation semantics** (E5): a rejected *modify* leaves the order **working at its previous parameters** (it still rests), emits an explicit reject with the reason, and bumps `rev` so the FE sees the event (BE fix landed 2026-09-21: `managerOrderBase.algoOrder.cpp` + `managerWebSocket.cpp`). A rejected *create* is terminal FAILED. A rejected *cancel* leaves the order working. - **Backpressure** (F4): dropped-by-backpressure marketdata is **not** a gap — coalescing happens before the sequence is assigned, so a dropped sample never leaves a hole. Orders and fills are the last thing dropped (they ride the same package but are never omitted while the package is sent); if the socket still cannot keep up the BE closes with **application close code `4013`** and reason `"backpressure"` (A4).

A `4013` close is an **expected** condition: the FE reconnects and issues a `resync` without showing the operator an error. Any other close code is a real failure and is surfaced. (1000/1001 remain normal/going-away; the reserved application range 4000–4999 carries our own codes.) - **Heartbeat** (F5): explicit `t:"beat"`, cadence and staleness threshold from `limits.heartbeatMs` / `limits.staleThresholdMs`; replaces the FE's 45 s local heuristic.

- **Clocks** (F6, A5): documented per timestamp. `ts` = **venue** exchange time (epoch-ms), `rts` = **BE receive** time (epoch-ms), so `rts - ts` is the feed latency. Order fields: `tcr`/`tac`/`tup`/`top` are **BE clock** epoch-ms (created, accepted, updated, terminal), which is what `getOrderAcceptanceLatencyMs` must use — the venue's own order time, where present, carries a distinct field name and is documented as venue time. Fill fields: the fill time is **venue** time (epoch-ms) and the BE receive time is emitted alongside it, so fill analytics (markout, VWAP, slippage, implementation shortfall, latency correlation) read payload clocks, never arrival time.

`tac - tcr` is defined as the blotter's acceptance latency (R3): a **BE-internal create -> accepted** measurement, which is what the LATENCY column reads today (~1 ms). This meaning is pinned, not incidental: if a **venue**

round-trip latency is wanted later it is a **separate, additionally-named metric** built from the venue-time field, and must never be a redefinition of this column. Do not "correct" `tac` to venue time — that would silently change what the column means.

1.4.8 4.5 Commands (G)

Every command (risk, pb, rec, files, new cancel-all/flatten) gets `rid` + mandatory ack; long-running ones return `accepted` then a terminal result; retries carry a dedup key.

1.5 5. State semantics (the part that kills phantom orders)

Ordering is a **two-phase** contract (FE-confirmed):

- **Phase 1 (v4 as shipped):** `timeUpdate` is the ordering key; `rev` is informational (display-only). This is required because the BE does not yet bump `rev` on every mutation — see §5.1.
- **Phase 2:** switch to `(id, rev)` — applied `if rev > current.rev`, ignored otherwise; a terminal state is never modified whatever the `rev`; a `rev` jump > 1 means intermediate states were missed and the FE applies the latest and logs a diagnostic (no replay needed); `timeUpdate` remains the display timestamp and retention key. The FE gates this on `caps: ["revOrdering"]`, so one build supports both and the switch is per connection.

entity	live set authority	terminal rule	ordering
order	**authoritative live** **snapshot** on `resync`/periodic	terminal immutable; rejected-modify stays working + rev bump	phase 1 `timeUpdate`, phase 2 `(id, rev)`
position	authoritative on `resync`	closed positions never resurrect	as above
fill	append-only	immutable once sent	`(id)` unique
quote/trade	sampled, non-authoritative	replaced by newer sample	per-instrument last

1.5.1 5.1 `rev` is a BE defect first (FE-reported, confirmed)

`ui32Rev` has exactly **four** write sites (`algoOrder.cpp:140` reset on create, `:257` rejected modify, `:272` accepted modify, `orderTrailingStopHandler.cpp:156`) against **25** order state assignments across `managerOrderBase.events.cpp`, `managerOrderBase.transport.cpp`, `managerOrderSim.cpp`, `managerOrderBase.algoOrder.cpp`, `managerOrderBase.cpp`, `orderStopHandlerBase.cpp` and `exchgManagerTradingT.h`. Fills, cancels, rejects, expiries and venue acks therefore all emit a state change with `rev` unchanged — which is why `(id, rev)` orders nothing today.

BE work item: centralize the transition on one `orderInfo_s::markState( newState )` that assigns `eState`, updates `infoTime.dtLastUpdate`, and increments `ui32Rev`; route all 25 sites through it. Rules: `rev` monotonic for the lifetime of the id, `rev = 0` is the create revision, `rev` is **not** reset on a state change, and a transition out of a terminal state is an **assertion**, not a convention. The two modify-attempt bumps stay, so an attempt advances `rev` before the venue answers.

This touches `orderInfo.h`, an ABI-visible struct shared by the exe and every plugin — it ships with a **full plugin-set rebuild/redeploy**, which is why it is sequenced with the v4 work rather than as a hotfix, and why phase 1 keeps `timeUpdate` ordering.

1.5.2 5.2 Fill embedded blocks are an FE dependency (G4)

A fill now rides the regular transmission package like every other category (§4.3), and it is emitted **in the same package as** its embedded `order/pos` blocks, so the FE's atomic merge is unaffected — a fill must never be split across packages from its embedded blocks. Note the FE **requires** those blocks today — `normalizeV3Fill` falls back to the embedded `order.id` when `oid` is absent. So the §7 "unify `t:fill` vs `t:fills`" cleanup must not simply drop them: any removal is a **coordinated change with an FE release**, never a BE-only cleanup.

Key point for A2: **the snapshot is authoritative for the LIVE set**, so the FE can safely drop any live entity absent from it. Terminal/history entities are never implied to be removed — they live in the query API (E4).

1.6 6. What changes on the BE side

1. **Message router** — `managerWebSocket` gains an inbound dispatch (`hello/resync/query/sub/ unsub/command`), with `rid` correlation and an always-answered error path.
2. **Authoritative live snapshot** per scope, built from the live vectors (already available).
3. **Query service** — in-memory rings first, file-backed paging second; filters and opaque cursors server-side.
4. **Subscription registry** — per-category, per-instrument/algo, with coalescing hints.
5. **Marketdata slimming** — emit candles + last trade + BBO; move the trade tape behind query.
6. **Aggregates** — per-algo realized/unrealized, turns, equity series; expose exactly.
7. **Caps/version/limits** in the handshake; per-category counters for observability (H3).
8. **Debug channel** (H2): session-marked raw-frame logging both sides.

The wire schema must be generated from the existing shared payload structs, which are an ABI boundary across the exe and every plugin — a schema change is a redeploy of the whole plugin set (see the repo build-and-deploy instructions).

1.6.1 6.1 Sequencing

Staged so the FE gets the reliability fixes early and the ABI-breaking work is batched:

stage	contents	FE impact
S1 — reliability floor	`markState` rev centralization (orders + positions); `hello/welcome` with `caps/limits/seq` semantics; `resync` for `orders/positions/account`; authoritative live-set snapshots; gap rule + resync storm control.	Fixes the phantom-order class. FE keeps `timeUpdate` ordering; `caps` has no `revOrdering` yet.

stage	contents	FE impact
S2 — on-demand data	`query`/`queryResult` for `orders`/`fills`/`positions`/`turns`/`trades`/`candles` with filters + cursors; BE-owned aggregates (turns, per-algo realized/unrealized, equity series, VWAP, markout); retention boundaries in `limits`. Delivered as S2a/S2b/S2c (§6.1.1).	FE retires its approximation inventory and the IndexedDB archive.
S3 — push slimming	`sub`/`unsub` with instrument/algo scoping and `batchHz`; marketdata becomes candles + last trade + BBO; trade tape behind query; per-category counters; heartbeat/clock-skew documentation.	Biggest volume reduction; FE chart/tape/toggle work (FE §2 items 1–6).
S4 — resume	Replay ring (~5 min + byte budget), all-or-nothing per scope, snapshot fallback with a stated reason.	Reconnect becomes cheap; no FE behaviour change beyond logging the reason.
S5 — optional binary	`.proto` generated from the payload structs, `protobuf-es` codegen, per-connection negotiation, JSON mode retained.	Only if S1–S4 measurement says size/cpu still hurts.

S1 is the only stage that is small enough to land quickly and is the one that stops the current class of production bugs. S1 and S2 both touch ABI-visible structures, so they should ship as one plugin-set rebuild rather than two.

FE retirements scoped to the stages (A10). S2 additionally retires on the FE side: the closed-turn retention plus the IndexedDB turn archive, and the `midTicks` buffers (max 256 per instrument) behind markout. S3 retires the tick-tape push path and the fixed per-instrument candle arrays (replaced by a map keyed by interval).

1.6.2 6.1.1 S2 slices (delivery order)

S2 as written has a large surface — eight `what` values, filters, cursors, aggregates and retention. It is delivered in **slices** so the FE gets value incrementally instead of waiting for the whole thing. Each slice is a complete, testable unit.

slice	contents	why this order
S2a — simple entity queries	`orders`, `fills`, `positions` with the `filter` object (`instrument`, `algo`, `venue`, `state`, `liveOnly`), `limit`, opaque `cursor`/`nextCursor`, `complete`, and `oldestAvailable` per category from `limits.history`.	The FE's approximation inventory is mostly *these three*; landing them first lets it start retiring its workarounds. They map directly onto data the BE already holds in the live vectors and journals.
S2b — marketdata queries	`trades` and `candles` with `base`/`intervalSec`/`from`/`to`, per-instrument depth and `oldestAvailable`.	Independent of the entity slice; unblocks the chart and the tape-as-query. Candle series are already produced by the BE.

slice	contents	why this order
S2c — derived aggregates	`turns`, per-algo realized **and** unrealized, `equity` (the two tiers), `markout` with `horizonsMs[]`.	Needs design (what a "turn" is, how markout horizons mature) and is the slice that closes the `PnLContext` inconsistency. Deliberately last: the FE can retire approximations before it gains exact aggregates.

Rules common to every slice (from §4.2 and §4.1):

- a `query` is answered by **exactly one `queryResult`** with the same `rid`, success or failure;
- unknown/unsupported `what` still answers the explicit unsupported result — never silence;
- opaque `cursor` values are the BE's; the FE treats them as strings and never constructs one;
- `oldestAvailable: 0` means "unknown / not yet published" (render no boundary claim), and beyond the retained window the BE answers `result.ok=false, code:"outOfRange"` rather than an empty page;
- retention boundaries come from `limits.history` and are **per instrument** for candles and trades; the `candles` `queryResult` reports the matched series' retained boundary (§4.2.3);
- `about` (S2's first consumer) is the reference implementation for the envelope shape.

1.6.3 6.2 Rollout mechanics

Fixture contract per stage. The FE suite is fixture-driven (contract tests + recorded `.ttpb` sessions), so it cannot start a stage until it has recorded frames; the BE provides one fixture set per stage so both sides work in parallel:

stage	fixtures the BE must produce
S1	a `hello`/`welcome` exchange with a real `caps`/`limits` payload; a `resync` for `orders`; a gap-then-resync sequence. A stubbed `caps`/`limits` with **concrete values** is enough — the FE branches on those strings, so values matter more than shape.
S2	paged `query`/`queryResult` for orders and candles, including a page boundary crossed while data arrives.
S3	one periodic package carrying orders, fills with their embedded blocks, tagged candle series at two intervals, and an `acct` update on a slower cadence.

Ordering of the FE's own work. The v4 handshake client (send/accept `hello`/`welcome`, parse `caps`/`limits`, gate every v4 behaviour on it, keep v3 working) is net-new FE work and must land with or before S1 to be useful. The **latency-KPI redefinition must ship in the same release as the batching change**, or the headline p50/p99 reads ~200 ms and measures the package interval — the FE replaces the inter-arrival metric with the payload-based one (`receiveTimestamp - exchangeTimestamp`, as `useFeedLatencyDistribution` already does). The blotter's per-order LATENCY column is unaffected.

Blotter flash note. The blotter flashes changed cells by diffing consecutive order arrays, so batching collapses several changes into one net diff — intermediate states are no longer flashed individually. Accepted as a visible (not silent) behavioural change.

1.7 7. Encoding decision

Recommendation: design transport-agnostic, ship JSON, add binary only if measurement says so.

- The FE already negotiates `permessage-deflate`; JSON compresses well, so the raw byte win from protobuf is smaller than benchmarks suggest.
- The real wins from protobuf are **parse CPU** (300+ msg/s with 1,000-order snapshots), **schema safety** (explicit presence), and **ids as `fixed64`**. The first two are largely addressed by this plan's semantics (fewer frames, authoritative live-only snapshots, slimmer marketdata); the third we already guarantee by sending ids as strings.
- Therefore: **one seam** in the FE socket layer (it already accepts `ArrayBuffer/Blob`) and one in the BE sender, so a future `.proto` is an additive change, not a rewrite.
- If we do go binary: **protobuf + `protobuf-es`** (Buf), one `.proto` shared by both sides, JSON mode retained for debugging and contract fixtures (D2/I3), and measure deflate-vs-raw for binary frames rather than assuming.

Order of attack if size is still a problem after the semantics land: (1) drop redundant keys from repeated delta frames, (2) tune/keep deflate, (3) then protobuf.

1.8 8. Migration (I)

- **Additive**: the v4 surface is new messages; existing v3 pushes keep working for a v3 FE.
- **Negotiated per connection** via `v` + `caps`.
- **Strict-vs-tolerant parser** (I2): the schema marks required/optional; the FE keeps its dev-mode strict validation and degrades in production.
- **Recorded sessions** (I3): either keep the JSON fixture format as the canonical recording (simplest, and JSON mode is retained anyway) or define a `.ttpb` migration; recommend the former until the binary format is proven.

1.9 9. Acceptance tests (BE side)

- `resync` for `orders` returns the authoritative live set; an FE that missed a terminal frame converges — the phantom-order scenario is reproducible as a test and passes.
- A rejected modify emits a reject with a `rev` bump and the order remains working; a rejected create is terminal FAILED.
- `query orders` with a filter + cursor pages without duplicates or gaps while data arrives.
- A gap injected in `seq` is detected and recovered by `resync`; a gap burst produces at most one in-flight resync per scope, and 5 consecutive failures surface as stale rather than retrying.
- **Initial-data ordering (R1):** a connect that sends `hello{lastSeq}` within the grace period receives a **replay** (and `t:"resumed"` with `replayed:true`), not a snapshot; a connect that sends nothing receives a full snapshot after ~250 ms; an unusable token receives a snapshot with the stated reason. A test asserts the replay path is actually taken on a warm reconnect.
- Unsubscribed categories produce zero frames; `sub` with `instruments` scoping sends only those.
- Caps/version mismatch yields an explicit error result, never silence.
- Counters per category are exposed and monotonic.
- `welcome` is emitted unsolicited as the first frame; a v3-style client (no `hello`) still receives a working v3 feed.

- Backpressure closes with application code **4013** and reason `"backpressure"`.
- A category omitted from a package is **not** counted as a gap and not treated as "not sampled" — a fixture with `acct` on a 5-frame divisor passes.
- Candle series carry `base` + `intervalSec` + `final`; an immature markout horizon is absent, never zero-filled.
- **about:** one item per attempted module, including `rejected`/`missing`; `state` present on every item; `abi`/`fingerprint` present for a rejected module whose ABI export resolved and **absent** (key not emitted) for `missingExport`/`missing`; `fingerprint` serialises as a hex string and `build` as a number; the `summary` reduction invariant holds (`expected == loaded + rejected + missing == items.length`, `consistent` as defined) and is asserted against `items` in a test; `about` is non-paged (`complete:true`, no `nextCursor`).
- **A rejected plugin is detectable:** a fixture with one deliberately mismatched module must produce `consistent:false` and a `rejected` item with structured `abi` values — this is the acceptance test for the absence hole, not just a field check.

1.10 10. Review-item status (FE sign-off folded in)

Spec frozen and FE-signed-off for S1 (2026-09-21). All fourteen review items are closed, and the FE's sign-off raised three remarks — R1 substantive, R2/R3 wording — all resolved below.

item	resolution in this revision
`about` module version information	**added** (§4.2.1) as the first real `query` consumer, requested once per connection generation. The FE's three codeable-verbatim requirements are landed in full: the **per-state field contract** (§4.2.1.1), the **opaque vs branchable** rule (§4.2.1.2), and the **`summary` reduction invariant** (§4.2.1.3).
`fingerprint` semantics (FE-raised)	**corrected** (§4.2.1.4): it is an ABI-surface hash with no module identity, so **all** modules must be equal on it and the earlier "exe may differ normally" framing is withdrawn.
absence semantics (FE-raised)	**fixed**: `about` reports **attempted** modules, not only loaded ones, with a per-item `state` and a `summary.consistent` reduction — otherwise a stale exe renders as a shorter clean list, defeating the detector.
R1 resume nullified by an early snapshot	**fixed (required)**: the BE must not send the initial snapshot until it receives `hello` or a **~250 ms grace period** elapses; then replay (per scope, all-or-nothing) or snapshot with a stated reason, and the outcome is reported explicitly via `t:"resumed"` (§4.1). Without this, S4's replay never fires on the common reconnect path.

item	resolution in this revision
R2 equity tiers stated two ways	**fixed**: two tiers are authoritative — 10 s for ~3 h, 60 s covering 7 d — and the retention prose no longer restates a 1 m/5 m scheme (§4.1).
R3 acceptance-latency meaning	**pinned**: `tac - tcr` is the blotter's BE-internal create->accepted latency; a venue round-trip latency would be a separate additionally-named metric, never a redefinition (§4.4).
A1 `oldestAvailable` sentinel	defined: `0` = unknown/not yet published, FE shows no boundary claim (§4.1)
A2 equity series shape	specified per point (`timestamp, equity, sessionPnL, spot, margin, byCurrency[]`) with `byCurrency` entry fields, and tiers published in `limits.history.equity.tiers` (§4.1)
A3 markout horizons	`horizonsMs[]` query parameter; BE defaults to T+1s/+5s/+30s/+60s; immature horizon is **absent**, never zero-filled (§4.2)
A4 backpressure close code	**4013** with reason `"backpressure"`; expected condition, reconnect+resync without operator alarm (§4.4)
A5 order/fill timestamp clocks	documented: `ts`=venue, `rts`=BE receive; order `tcr/tac/tup/top`=BE clock; fill time=venue + BE receive alongside (§4.4)
A6 phase-1 ordering precision	restated: state transition always applied, only same-state compared by `timeUpdate`, terminal never modified; no ordered-delivery requirement (§4.4)
A7 resync backoff ceiling	capped at 10 s, max 5 consecutive failures, then surfaced to the operator as stale — never silent infinite retry (§4.4)
A8 frame cadence in `limits`	`periodicMs` added (default 200) (§4.1)
A9 conditional selectors	base/interval toggles render only when both are published, else read-only label (§4.2)
A10 FE retirements scoped	turn archive + `midTicks` folded into S2, tick-tape push + fixed candle arrays into S3 (§6.1)
`batchHz` under one package	decided: one frame cadence (`periodicMs`), `batchHz` = per-category ceiling as a divisor; BE omits a category with nothing new, so **absence means no change**, never "not sampled"; `maxHz` may raise a category up to the frame rate (§4.1)
`maxHz` can raise the rate	confirmed: ceiling is the frame rate; a blotter gets orders/fills/positions at `periodicMs`
v4 detection	BE sends `welcome` **unsolicited on connect**; first-frame `welcome` = v4, otherwise v3 behaviour; no probing (§4.1)
Fixtures per stage	BE provides S1/S2/S3 fixture sets so both sides work in parallel (§6.2)

item	resolution in this revision
`t:"fill"/t:"fills"` unification	FE lands the embedded-block removal **in the same release as S3**, gated on (a) `oid` always present and non-empty, (b) `t:"pos"` + snapshot fully covering position changes — both verified in that release, not assumed. Note the embedded `pos` block is currently dispatched as a position update, not merely a fallback (§5.2)

Remaining as **implementation**, not negotiation: the `markState` centralization (ABI change, ships with a full plugin rebuild, which is why phase 1 keeps `timeUpdate` ordering), and the concrete `caps/limits` values in the S1 fixture.

1.11 11. Implementation addendum (2026-10-15) — S2a/S2b shipped, session block and `q` channel live

Dated addendum; the frozen S1 body (§§1–10) is unchanged by it. Wire state recorded as built in the BE (`src/managers/managerWebSocket.cpp`, `src/managers/frontendProtocol.{h,cpp}`); the contract-facing half is `frontend-wire-contract.md` §11.

- **S1** (reliability floor): shipped — `markState` rev centralization, the unsolicited `welcome`, `hello/resumed` with the closed reason set, `resync` + authoritative live-set snapshots + storm control, heartbeat and the 4013 backpressure close. The S1 replay ring is still absent: a usable token answers `replayed:false, reason:"tokenTooOld"` (the §4.1 S1 behaviour, not an accident).
- **S2a**: `query` answers `orders/fills/positions` from the entity-query store with the §4.2 filter object, the bounded-filter input limits, newest-first keyset cursors (category + filter fingerprint) and the per-category `oldestAvailable`. The store refresh is triggered only by a well-formed request, and it snapshots live vectors **plus** the retired rings, so terminal entities are queryable while the live-set snapshot deliberately excludes them.
- **S2b**: `candles` is live with the §4.2.3 shape and error codes; `trades` is live through the same pipeline over the websocket strand's value-copy trade-capture ring. Candle bars are published by the marketdata strand after finalization only (a still-open bucket is never returned; `final:true` on every item).
- **S2c** is not started: no turn records yet, so `turns/aggregates` are not in `caps` and the `turnsArchived` pull op answers the explicit `unsupportedOp` result.
- **S3-interim**: sub/unsub are answered with a real `subAck` acknowledgement (`result:{ok:true}`, `rid` echoed) because `caps` now advertises `sub`; per-category push filtering is not implemented and the FE accepts any superset of what it asked for.
- **Additive, beyond the stage table**:
  - a `session` block (spec item A) in the periodic package — session-anchored aggregates (per venue and per algo, realized net of fees / fees / unrealized, turn counters, exposure), re-pushed only when a fill moved the sources, account totals re-summed from the venue rows;
  - a `t:"q"` pull channel (`queryOps` cap): one `queryResult` per `qid`, ops `sessionBlock`, `instruments`, `ordersOpen`, `ordersArchived`, `positionsOpen`, `positionsArchived`, `executions`, `turnsArchived`.
- **`resync` positions answer grew**: the authoritative live set **plus** every position closed within the retained window (terminal records carrying final `rpnl/fees/tcl`), so the FE rebuilds the closed blotter from one scope.

- Open, unchanged: the `markState` centralization still ships with the full plugin-set rebuild (which is why `revOrdering` remains absent from `caps` and phase 1 keeps `timeUpdate` ordering), and the S4 replay ring.

Chapter 2

Frontend wire contract

Status: normative for the frontend. This document is the single authority for everything the backend (BE) puts on the frontend websocket; it is generated against and must stay in sync with the `toJson` sources: `include/ttTrader/general/orderInfo.h`, `positionInfo.h`, `executionInfo.h`, `algoInfo.h` and `src/managers/managerWebSocket.cpp` (envelope + sections).

When a wire dispute arises, `frontendWs.logTx` (see §9) records the serialized frame before enqueueing it to the websocket server. It is emission evidence, not a transport acknowledgement or proof that a frontend received the frame.

2.1 1. Does the protocol change?

Direct answer to the FE team, dated 2026-09-16:

- **Keys: no change.** FE fills embed `order` and optional `pos`, not `oid` or `pid`. The non-FE short rendering uses `oid`/`pid` instead. Booking-time semantics, `upnl`/`rpnl`, section names and `rev` are unchanged.
- **Types:** order and position `id` (including embedded objects), and the short rendering's `oid`/`pid`, change from numbers to decimal strings. Fill `id` remains a string; live venue execution ids may be nonnumeric. `rev` and other non-identity numerics are unchanged.
- **Values:** allocator-minted identities are approximately 1.2e17 and require persisted range reservations for restart safety (§4). Live execution ids retain the venue identity and scope; they are not globally unique.
- **Semantics:** one fill delta per `(order.id, id)` tuple per connected session; snapshots legitimately repeat stored fills. FE keys fills by that tuple, positions by `pos.id`/`positions[].id`, and order revisions by `(id, rev)`. Delete content-fingerprint dedupe. This is not an exactly-once transport guarantee across disconnect (§6).

Additive fields (2026-09-20, no existing key renamed or retyped):

- order `cr` — optional cancel reason, present only when the owning algo supplied one at pull time. Lets the blotter show *why* a resting order disappeared instead of only that it did (§5.1).
- algo `instr` — array of `{ id, eid }` naming the instruments an algo subscribed to and their venue, so the dashboard can render an algo's market coverage and exchange (§5.5).

2.2 2. Transport and envelope

- Endpoint: `ws://<host>:<port>` (config `frontendWs.host`/`frontendWs.port`, default `0.0.0.0:18181`). Compression (permessage-deflate) is negotiated transparently.
- Every outbound frame is a JSON object:

```
{ "seq": 4127, "t": "fills", "d": { "..." : "..." } }
```

field	type	semantics
`seq`	number	process-local outbound sequence for deltas; initial snapshot and error frames use `0`; snapshot `nextSeq` supplies the resynchronization point
`t`	string	message section (§3)
`d`	object	section payload

Inbound (FE -> BE) frames carry a request object (`t` + `action` + request fields, e.g. playback control); they do not affect the id contract.

2.3 3. Sections and their triggers

section (`t`)	trigger	payload
`snap`	session connect (full snapshot, targeted at that session, `seq` 0, payload carries `nextSeq`); every timer tick (delta, only sections that changed)	object; keys present as detailed below
`order`	every order lifecycle event: create requested, venue ack, update, cancel, terminal/failed	one `orderInfo_s::toJson` object (§5.1)
`fills`	every execution booked (fill)	one `executionInfo_s::toJson` object (§5.2)
`pb`	playback/recording status transitions	playback status object
`error`	BE-side error report	`{ "message": string }`
`exchg`, `instr`	reserved (declared, currently not emitted)	—

Snapshot (`snap`) payload keys: `nextSeq`, `states`, `algos`, `exchanges`, `acct`, `instruments`, `trades`, `quotes`, `orders`, `fills`, `positions`, `playbackFiles`, plus `playback`/`recording` when active. The timer-driven delta frames carry only the changed subsets (`algos`, `exchanges`, `acct`, `instruments`, `trades`, `quotes`, `recUpdate`, `pbUpdate`). Marketdata keys (`trades`, `quotes`) are suppressed in playback mode.

- `orders` / `fills` / `positions` are arrays of the corresponding objects (§5). Orders/positions combine active state and bounded retired-state rings; fills come from the bounded execution JSON ring. The snapshot does not replay arbitrarily old history — the FE keeps its own retained history.
- Merge rule: apply the snapshot, then consume deltas. Identity is §4 only — never content.

2.4 4. The id scheme (normative)

- **Format:** `id = (epochMs48 << 16) | seq16` — a millisecond timestamp in the high 48 bits, a 16-bit per-millisecond sequence in the low bits. One allocator instance per order-manager process mints order ids, sim execution ids, position ids and simulated exchange order ids (`include/ttTrader/general/uniqueIdAllocator.h`).

- **Range:** 48-bit milliseconds span about 8.9k years from the Unix epoch, not 283k years. A sequence overflow advances a logical millisecond; clock regression is clamped within the running allocator. Encoded time is not necessarily the event's wall time.
- **Restart safety:** timestamp+counter alone is insufficient. A restart can encounter clock rollback or a prior burst's synthetic millisecond. The allocator therefore reserves ranges durably before use: a reservation record at `<logRoot>/unique_id_reservation.bin` (24-byte record: magic, `throughMs` floor, checksum) is written before the first mint, holding a 60000 ms window ahead of the allocator; an exclusive owner handle is kept for the process lifetime and renewed from the manager timer; the floor only advances. Unused reserved ids are abandoned on restart — minting resumes beyond the persisted floor, never inside it. Corruption, a lock conflict (another live owner) or an unwritable store fails closed at init (`DDT_ERROR_CANNOT_OPEN_WRITE`); the process does not fall back to unreserved minting. Verified live: three consecutive runs produced disjoint id spaces across restarts. The guarantee still requires retained, non-rolled-back state and exclusive deployment ownership; deleting the state, restoring a stale copy or cloning it into concurrent producers invalidates it.
- **Magnitude:** allocator ids are ~1.2e17 — **exceed JavaScript's `Number.MAX_SAFE_INTEGER` (2^53)**. Order, execution and position identity fields serialize as strings to avoid rounding. This does not change unrelated identifiers such as numeric `playbackFiles[].id`.
- **Opaque:** use string equality, never numeric parsing, ordering, truncation or timestamp extraction. Live execution ids retain venue ids, which can be nonnumeric and scoped by instrument, venue or order. They share no ordering relationship with allocator ids.
- **Identity:** fill identity is the tuple `(order.id, id)` in embedded FE rendering, `(oid, id)` in short rendering. Use a structured tuple or an unambiguous encoding, not delimiter concatenation of arbitrary strings. A raw fill `id` alone is not a deployment-wide key. Order, fill and position identities are different entity kinds; never compare them across kinds.

2.5 5. Field tables (generated against the `toJson` sources)

Order and position ids are allocator-minted decimal strings; deployment uniqueness depends on §4. Execution ids are opaque strings with tuple scope. These tables cover trading entities, not every identifier in every section.

2.5.1 5.1 Order — `orderInfo_s::toJson` (section `order`, snapshot `orders[]`)

field	type	semantics
`id`	**string (opaque decimal, deployment-unique)**	order identity (parent id)
`rev`	number	modify revision; `(id, rev)` identifies the revision, while parent order identity remains `id`; display as `id:rev`
`tp`	number	order type (1 market, 2 limit, 3 stop-market, 4 stop-limit)
`tf`	number	time in force (0 GTC, 1 FOK, 2 IOC)
`st`	number	processing state (numeric values of the legacy lifecycle enumerators are frozen)
`sz`	string (decimal)	order size
`sl`	string (decimal)	remaining size
`gd`	number	good-til-date, unix ms — present only when set

field	type	semantics
ga	number	good-after-date (deferred activation), unix ms — present only when set
lm	string (decimal)	limit price — limit orders
tr	string (decimal)	trigger price — stop orders
trig	bool	stop triggered — stop orders
trd	string (decimal)	trailing delta — trailing stops
tdir	number	trigger direction — trailing stops
loc	number	resting location (broker / local / exchange)
nt	string	note: state message or user tag
cr	string	cancel reason — present only when the owning algo supplied one at pull time ("requote", "pull gate", "breakout", "flatten", "cycle closed", …). Absent on create/update and on manager-initiated cancels that carry no algo reason
iid	string	instrument config id
aid	string	owning algo config id
sd	number	order direction (buy family / sell family)
tcr	number	algo create time, unix ms
tac	number	exchange create-ack time, unix ms
tup	number	last update time, unix ms

2.5.2 5.2 Execution / fill — executionInfo_s::toJson (section fills, snapshot fills[])

field	type	semantics
id	**string (opaque, order-scoped)**	allocator-minted in simulation; retained venue execution id in live mode, possibly nonnumeric; key with order.id
iid	string	instrument config id
sd	number	order direction
sz	string (decimal)	filled size
px	string (decimal)	fill price
val	string (decimal)	notional = sz × px
feeCcy	string	fee currency
fee	string (decimal)	fee
ts	number	fill timestamp, unix ms
latMs	number	realized latency from venue create ack to this execution
order	object	the owning order serialized in full (§5.1)
pos	object	the position record this fill booked into (§5.3), present only when the fill booked a position

Short (non-embedded) rendering (`toJson(false)`) keeps the scalar fill fields above but replaces `order/pos` with `oid/pid`; it is not the FE payload. Its identity is `(oid, id)`:

field	type	semantics
`oid`	**string (opaque decimal, deployment-unique)**	owning order's parent id
`pid`	**string (opaque decimal)**	booked position id (uniqueness per §4); `"0"` is the no-position sentinel, not an entity identity

2.5.3 5.3 Position — `positionInfo_s::toJson` (snapshot `positions[]`, embedded `pos`)

field	type	semantics
`id`	**string (opaque decimal, deployment-unique)**	position record identity; a fresh position instance receives a fresh id after the prior instance closes, including close-then-reopen on the same instrument/algo
`long`	object	long bucket: `sz`, `val`, `px`, `fee` (decimal strings), `cnt` (number of executions)
`short`	object	short bucket: same shape
`iid`	string	instrument config id — snapshot rendering
`sd`	number	position direction — snapshot rendering
`aid`	string	owning algo config id — snapshot rendering
`eid`	string	exchange config id — snapshot rendering
`ccy`	string	quote currency — snapshot rendering
`tcr`	number	record create time, unix ms — snapshot rendering
`top`	number	open time, unix ms — snapshot rendering
`nt`	string	note
`rpnl`	string (decimal)	realized PnL
`upnl`	string (decimal)	unrealized PnL
`fees`	string (decimal)	accumulated fees
`tcl`	number	close time, unix ms — present only when closed
`tup`	number	last update time, unix ms

For S1-interim `snap` position deltas, the array contains the current live positions plus a flattening record for each position that reached zero. The flattening record is retained for at least the push containing the transition, including `tcl` and final realized `rpnl`, and is then omitted from later live pushes. A zero-size record without `tcl` is invalid. Position changes are published on the fill strand turn (and may also be included in the periodic

package); unchanged `positions` is omitted. Every containing envelope consumes the receiving connection's monotonic `seq`.

Booking-time semantics (unchanged, restated because it was the root of past disputes): `pid`/`pos` name the position record the fill booked into at booking time (`bookedPositionPtr`), never `orderPtr->positionPtr` — a later fill re-resolves the order's pointer once the booked record closed.

2.5.4 5.4 Snapshot `states`

field	type	semantics
`ex`	number	execution mode
`mode`	string	raw trading mode (`live/paper/simulation/playback`)
`pb`	number	playback state
`rec`	number	recording state

2.5.5 5.5 Algo — `algoInfo_s::toJson` (snapshot `algos[]`, delta `algos`)

field	type	semantics
`id`	string	algo config id
`info`	string	algo display name (snapshot rendering only)
`st`	number	algo status (`ENUM_ALGO_STATUS`)
`tst`	number	algo start time, unix ms
`rpnl`	string (decimal)	realized PnL, gross of fees
`rpnlNet`	string (decimal)	realized PnL, net of fees
`instr`	array	instruments the algo subscribed to, each `{ "id": string, "eid": string }`; empty array when none. Static after load and emitted on snapshots and deltas alike, so a frontend that missed the snapshot still learns the coverage. `id` joins the `instruments` section for symbol and contract details; `eid` is the venue

2.6 6. Delivery and dedupe contract

- **One fill delta per tuple per session.** The BE must not emit a second `t: "fills"` delta for the same `(order.id, id)` to the same connected session, or remint a fill merely to conceal a duplicate. This is a producer invariant, not a guarantee of receipt.
- **Snapshots are state, not new fills.** Initial/reconnect snapshots can repeat fills already seen in deltas or an earlier snapshot. Merge by tuple; do not book snapshot entries again as new trading activity. Retired history is bounded; a snapshot is not a complete offline fill replay.
- **Live-venue replay:** when a venue supplies a stable execution id and the replay resolves to the same retained order, the hub suppresses the duplicate before fan-out. Both numeric and string ids are scoped per order in `_executionDedupeCheck`; its retained window is bounded. Do not infer unlimited or durable cross-restart deduplication from this in-memory guard.
- **FE merge keys — and nothing else:**

  - fills: `(order.id, id)`; non-FE short rendering: `(oid, id)`
  - positions: embedded `pos.id` or snapshot `positions[].id`; ids are unique per position instance and are never reused after close
  - order revisions: `(id, rev)`; parent order identity remains `id`
- **Delete the content fingerprint.** The historical `symbolId|side|size|price|second` fingerprint collapses legitimate matching fills. Matching content with different tuples represents different fills.
- **Transport limits:** no application-level receipt acknowledgement or exactly-once delivery across disconnect is promised. `seq` is process-local, not a durable fill identity; `0` is used for snapshots and errors. It cannot establish global replay identity or prove delivery. Use snapshot `nextSeq` to resynchronize the current connection, not to deduce offline completeness.
- **IBKR public trade ids are separate.** A synthetic id stamped on receipt lets simulation deduplicate local redelivery of that same stamped event. An incoming venue replay receives a fresh stamp, so this does not identify non-consecutive venue replays. It is not the private execution-id contract.

2.7 7. Migration (number -> string)

- FE order/position `id` (including embedded `order`/`pos`) and the short rendering's `oid`/`pid` switch from JSON numbers to decimal strings. Fill `id` remains a string and preserves live venue values, including non-numeric ones. `rev` and non-identity numerics remain numbers.
- During a compatibility window, `String(v)` can normalize legacy numeric trading ids only if they arrived as safe integers. It cannot recover digits already lost by JSON number parsing. Upgrade FE string handling before a backend starts sending high-magnitude ids; do not claim arbitrary deploy order is safe with an old numeric-only FE.
- After migration, a numeric order/fill/position identity is a BE contract violation. Unrelated numeric ids (for example `playbackFiles[].id`) are not part of this type change.
- Replace fill-id-only and content-fingerprint keys with `(order.id, id)`; use `(oid, id)` only when consuming the explicit short rendering.

2.8 8. Worked examples

Illustrative fill frame (section `fills`), shaped against `toJson(true)`. The numeric-looking execution id is an example, not a live-id syntax rule:


```
{
  "seq": 5, "t": "fills",
  "d": {
    "id": "117281758267637806",
    "iid": "instLighterBTC", "sd": 6,
    "sz": "0.69142", "px": "75761.8", "val": "52383.223756",
    "feeCcy": "USDC", "fee": "0",
    "ts": 1789577610285, "latMs": 0,
    "order": {
      "id": "117281758267637804", "rev": 0, "tp": 1, "tf": 0, "st": 3,
      "sz": "0.69142", "sl": "0", "loc": 3, "nt": "close position market",
      "iid": "instLighterBTC", "aid": "algoEmber", "sd": 6,
      "tcr": 1789577610285, "tac": 1789577610285, "tup": 1789577610285
    },
    "pos": {
      "id": "117281753444450345",
      "long":  { "sz": "0.69142", "val": "52401.68467", "px": "75788.5", "fee": "0", "cnt": 1 },
      "short": { "sz": "0.69142", "val": "52383.223756", "px": "75761.8", "fee": "0", "cnt": 1 },
      "iid": "instLighterBTC", "sd": 1, "aid": "algoEmber",
```

```
      "eid": "exchgLighter", "ccy": "USDC",
      "tcr": 1789577534444, "top": 1789577534444, "nt": "",
      "tcl": 1789577610285,
      "rpnl": "18.460914", "upnl": "0", "fees": "0", "tup": 1789577610285
    }
  }
}
```

Position snapshot fragment (inside a `snap` frame's `positions` array):

```
{
  "seq": 0, "t": "snap",
  "d": {
    "nextSeq": 1,
    "positions": [
      {
        "id": "117281752379686944",
        "long":  { "sz": "14", "val": "107513", "px": "7679.5", "fee": "8.68", "cnt": 1 },
        "short": { "sz": "14", "val": "107513", "px": "7679.5", "fee": "8.68", "cnt": 1 },
        "iid": "instInteractiveBrokersNES", "sd": 2, "aid": "algoEmber",
        "eid": "exchgInteractiveBrokers", "ccy": "USD",
        "tcr": 1789577523796, "top": 1789577523796, "tcl": 1789577610287,
        "nt": "", "rpnl": "0", "upnl": "0", "fees": "17.36", "tup": 1789577610287
      }
    ]
  }
}
```

2.9 9. Evidence tooling

- `frontendWs.logTx: true` records serialized outbound frames before socket-server enqueue to `<logRoot>/wsTx/wsTx_YYYYMMDD.jsonl` (preamble: `event_id`, `timestamp_ns`, `t`, `seq`, `sid`, `len`, then the wrapper). Compare a frontend capture byte-for-byte, but do not equate a tx-log entry with FE receipt. Broadcast logging precedes per-session fan-out and logging itself can degrade on IO failure.
- Trade logs print `order_id` as `parent:rev` and `trade_id` as the retained execution id. Join fills by `(order parent id, trade_id)`, not `trade_id` alone; keep revision information for order-log joins. Restart stability of allocator identities depends on retained reservation state. See `docs/09-operations/export-contract.md`.

2.10 10. v4 S1 additions (reliability floor)

Stage S1 of the FE/BE v4 plan (`ttTrader/.kilo/algoPlans/fe-be-protocol-v4-plan.md`) adds the reliability surface below. It is **additive**: a v3 client that sends nothing keeps the v3 feed (section 3), because the BE releases the v3 `snap` on the R1 grace timeout.

2.10.1 10.1 `seq` semantics (normative)

`seq` is a **per-connection send counter**. It advances only for frames actually sent on the connection. A filtered-out or omitted category consumes no sequence number, so **omission is never a gap**. The FE gap rule is exactly "a received `seq` is greater than last+1". The counter is keyed per websocket session and resets on connect (v3 snapshots publish `nextSeq` = next value for that session; a v4 `welcome` carries no `d.seq` — the FE seeds its baseline from the first frame's envelope `seq + 1`), so a second client is unaffected by the first's traffic.

2.10.2 10.2 Handshake: `welcome`, `hello`, `resumed`

The v4 BE sends `t:"welcome"` **unsolicited as the first frame** on a connection. A first-frame `welcome` means v4; anything else means the FE runs v3.

```
FE -> { "seq":0, "t":"hello",   "d":{ "v":4, "sessionId"?, "lastSeq"?, "rid" } }
BE -> { "seq":N, "t":"welcome", "d":{ "v":4, "sessionId", "sessionSeqToken":T, "caps":[...],
                                  "limits":{...}, "exe":{...}, "result":{ "ok":true } } }
BE -> { "seq":N, "t":"resumed", "d":{ "scopes":["orders","positions","account"],
                                      "replayed":false, "reason"?, "rid":<hello rid> } }
BE -> { "seq":N, "t":"snapshot","d":{ "scope":"orders", "seq":N, "data":[...],
                                      "complete":true, "rid":<hello rid> } }
```

`resumed` shape (frozen). The scopes field is the **plural array `scopes`** (not a singular `scope`); `replayed` is a boolean; `reason` is optional. `rid` echoes the FE's `hello.rid` on the `hello` path, and the per-scope `snapshot` frames that answer a `hello` carry the same `rid`. The no-`hello` grace-timeout path is v3-equivalent and carries no `rid` (there is no request to correlate to).

`reason` is a **closed set of exactly three literals** — the FE must treat anything else as a contract violation:

`reason`	meaning
`tokenTooOld`	the token predates the replay window. **In S1 the replay ring is absent, so every usable token is effectively too old and yields `replayed:false, reason:"tokenTooOld"`.**
`sessionChanged`	the token's `sessionId` is not this connection's
`byteBudget`	the replay exceeded `limits.resumeByteBudget` (reserved for S4; not produced in S1)

Initial-data ordering (R1, required). The BE must not send the initial snapshot immediately after `welcome`. It waits for `hello` or a **~250 ms grace period**, then:

condition	response
`hello` with a usable token	replay the covered scopes (S4); in S1 no ring exists, so a snapshot with `replayed:false, reason:"tokenTooOld"`
`hello` with no token	full snapshot for all scopes (no `reason`)
`hello` with an unusable token	full snapshot, `reason` one of `tokenTooOld` | `sessionChanged` | `byteBudget`
no `hello` within grace	v3-equivalent `t:"snap"` for all scopes (no `rid`)

The static directories ride both paths. Whether the initial data is decided by `hello` or by the grace timeout, the baseline carries the static directories — `algos`, `exchanges`, `instruments` — plus the marketdata and playback blocks. They are a **connection baseline, not a resync scope**: the `orders/positions/account` scoped live sets alone reference instruments by `iid` and resolve nothing without it. The grace path carries them in the v3-equivalent `t:"snap"`; the `hello` path carries an equivalent `t:"snap"` baseline frame (real envelope `seq`, no `nextSeq`, no entity blocks) before the per-scope `snapshot` frames.

An unsupported `v` is answered with a correlated error result (never a snapshot): `{ "t":"error", "d":{ "rid":<rid>, "message":"unsupported protocol version", "result":{ "ok":false, "code":"unsupportedVersion", "msg":"unsupported protocol version" } } }`.

`caps` (stable identifiers) is **per-build and carries only the surfaces that build implements**: a cap present is a promise the BE answers it. In S1 the set was `resume`, `query`, `about`; the current build answers the superset documented in §11.1 (`resume`, `query`, `queryOps`, `sessionBlock`, `trades`, `candles`, `about`, `sub`). **`revOrdering` is deliberately absent in S1** (phase 1 keeps `timeUpdate` ordering); it appears only once `markState` is deployed and verified. The S2b `candles` `queryResult` shape (item fields, ordering, cursor behaviour) is specified in `protocol-v4-spec.md` §4.2.3.

`limits` is exactly:

```
{ "limits": {
  "heartbeatMs": 5000, "staleThresholdMs": 45000,
  "periodicMs": 200,
  "maxPageSize": 500, "maxQueryRangeMs": 604800000,
  "resumeWindowMs": 300000, "resumeByteBudget": 4194304,
  "maxSubscriptions": 32, "maxInstrumentsPerSub": 256,
  "batchHz": { "orders": 5, "fills": 5, "positions": 5, "quotes": 5, "trades": 5,
               "acct": 1, "algos": 1, "instruments": 1, "exchanges": 1 },
  "history": {
    "orders":    { "memoryRows": 50000,  "oldestAvailable": 0 },
    "fills":     { "memoryRows": 50000,  "oldestAvailable": 0 },
    "positions": { "memoryRows": 5000,   "oldestAvailable": 0 },
    "turns":     { "memoryRows": 5000,   "oldestAvailable": 0 },
    "trades":    { "memoryRows": 100000, "oldestAvailable": 0 },
    "equity":    { "maxRangeMs": 604800000, "tiers": [ { "bucketMs": 10000, "maxRangeMs": 10800000
                                                       { "bucketMs": 60000, "maxRangeMs": 604800000
  } } }
```

`oldestAvailable: 0` is the defined **unknown / not-yet-published sentinel** (the FE shows no boundary claim, never "from 1970"). The two equity tiers are 10 s for ~3 h and 60 s for 7 d.

2.10.3 10.3 `resync`, snapshots and storm control

```
FE -> { "t":"resync",  "d":{ "scope":"orders", "rid" } }
BE -> { "t":"snapshot","d":{ "scope":"orders", "seq":N, "data":{...}, "complete":true, "rid" } }
```

`scope` is a **single scope string** on both `resync` and `snapshot` — one scope per request. (Names, not a JSON array: `orders` | `positions` | `account`.) The `resumed` frame is the only one that carries a plural array, `scopes`.

The snapshot is the **authoritative live set** of that scope, built from the live vectors (`m_vectorOrderActivePtr` / `m_vectorPositionsPtr`). Two rules the FE depends on:

- The FE **may drop any live entity absent from it** — absence means "not live", not "unknown".
- Terminal/history entities are **never implied removed**; they live in the query API (S2).
- Because it is the live set, a snapshot entry is **always a full order payload** (`toJson(true)`): `id`, `rev`, `tp`/`tf` (type/TIF), `st`, `sz` (`sl` remaining), `loc`, `iid`, `aid`, `sd` (side), `tcr`/`tac`/`tup`, and the limit/trigger fields. The FE can materialise an order it has never seen from a snapshot entry alone — which is the whole point, since orders created during a gap are exactly the ones it has never seen. The live set does not carry terminal orders.

`snapshot.data` shape per scope. `data` is the value itself, never wrapped in a scope-named key: `orders`

and `positions` are arrays of full payloads (`toJson(true)`); `account` is an **array of account blocks — exactly the value the v3 snap carries under `acct`**: `{ id, eid, ccy, dec, balances:[ { ts, spot:{bal,avail,eq,locked,spnl}, mrg:{bal,avail,eq,locked,spnl} } ] }`. An empty array is a valid authoritative value; `{acct:[...]}` / `{account:[...]}` are contract violations and the FE must fail loudly on them.

At most one resync is in flight per scope; duplicates raised while one is pending are coalesced and answered with the same result. Repeated failures back off exponentially, capped at 10 s, and after 5 consecutive failures the BE stops retrying and answers `result.ok=false, code:"resyncStale"` instead of retrying silently.

2.10.4 10.4 Order and position revisions

Every order state transition routes through one helper (`orderInfo_s::markState`) that assigns the state, updates `dtLastUpdate` and increments `rev`. Rules: `rev` is monotonic for the lifetime of the id; `rev = 0` is the create revision; `rev` is **not** reset on a state change; a transition out of a terminal state is refused and asserted. The two modify-attempt bumps remain, so a modify attempt advances `rev` before the venue answers, and a **rejected modify leaves the order working with `rev+1`** (never terminal). Positions gained an additive `rev` field, advanced by `markUpdated()` / `markClosed()`; a closed position is terminal and is never mutated in place.

In S1 `rev` stays **informational** (phase 1 orders by `timeUpdate`, per the frozen spec) and `caps` does not advertise `revOrdering`.

2.10.5 10.5 Heartbeat and backpressure

`t:"beat"` is emitted at `limits.heartbeatMs` (5000 ms), shaped `{ "seq":N, "t":"beat", "d":{ "ts":<BE epoch-ms> } }`. A slow client is closed with the reserved application close code **4013** and reason `"backpressure"`; the FE treats this as an expected condition and reconnects + resyncs without operator alarm. 1000/1001 remain normal/going-away, and 4000–4999 is the reserved application range.

2.11 11. v4 implementation state (2026-10-15)

The BE has moved past the S1 floor: S2a (entity queries), S2b (candles and trades) and the S3-interim `sub` acknowledgement are implemented, and a session-anchored aggregate block and a second query channel shipped as additive surfaces. This section records the wire state as built; the earlier S1 text stands for what it says and is not amended here.

2.11.1 11.1 caps as emitted

`buildCaps()` now answers a superset of the S1 promise (the S1 list stands for the S1 build; a cap present is a promise the BE answers it):

```
{ "caps": [ "resume", "query", "queryOps", "sessionBlock", "trades", "candles", "about", "sub" ] }
```

- `resume` — the `hello/resumed` outcome with a stated reason; the S4 replay ring is still absent, so a usable token still yields `replayed:false` with `reason:"tokenTooOld"`.
- `query` — the S2a/S2b query surface (§11.2).
- `queryOps` — the `t:"q"` pull channel (§11.4).
- `sessionBlock` — the session aggregate, pushed and pullable (§11.3).
- `trades`, `candles` — S2b marketdata queries, joined with S2b.
- `about` — the module build-identity query (§10 and the spec's §4.2.1).
- `sub` — the S3-interim acknowledgement (§11.5).
- `turns/aggregates` are not in the list: S2c has no turn records yet; a `turnsArchived` pull answers `result.ok=false, code:"unsupportedOp"` with the requested op, never silence.

- `revOrdering` remains deliberately absent until the `markState` centralization is deployed and verified (spec §5).

2.11.2 11.2 The query surface (S2a + S2b, as implemented)

The `query`/`queryResult` shape of spec §4.2 is in force. The `what` values answered with data in the current build:

- `about` — the build-identity `items` + `summary` from the module registry (spec §4.2.1), non-paged, `complete:true`, no `nextCursor`.
- `orders` / `fills` / `positions` (S2a) — paged from the entity-query store. The store is refreshed per request and holds the live vectors **plus** the retired rings, so terminal entities are queryable; the authoritative live-set snapshot (§10.3) still excludes them.
  - filter object: `instrument[]`/`algo[]`/`venue[]`/`state[]` plus `liveOnly`; an absent axis restricts nothing, a present but empty array matches nothing. `state[]` accepts the documented state name or the numeric code as a decimal string (orders), `"open"`/`"closed"` (positions); a fill has no state.
  - filter inputs are bounded: at most 256 elements per axis, at most 128 characters per element; a request that exceeds the bounds answers `badFilter`.
  - paging is newest-first keyset on `(timeMs, id)`; the opaque cursor encodes category + a filter fingerprint + the last returned row's key, so a cursor is rejected as `badCursor` when it is malformed or belongs to another category/filter. `limit` clamps to `limits.maxPageSize`. A cursor older than the retained boundary answers `outOfRange` with `oldestAvailable`, never an empty page.
- `trades` (S2b) — the venue trade tape, the same pipeline; rows are flat and self-describing (the trade fields plus `iid`/`eid`).
- `candles` (S2b) — spec §4.2.3: `filter.instrument` required and non-empty, `base` one of `"trade"`/`"midpoint"`/`"excha`, `intervalSec` positive, `from`/`to` inclusive open-time bounds in ms; every item carries `base` + `intervalSec`, items in a page are ascending, the walk is newest-first via `nextCursor`, and `oldestAvailable` is the minimum non-zero per-series retained boundary.
- every other `what` (including a missing/unknown one) answers `result.ok=false, code:"unsupported"` with the request `rid` — never silence.
- error codes: `badFilter`, `badCursor`, `outOfRange`, `unsupported`.

2.11.3 11.3 The session block (additive, `sessionBlock` cap)

The periodic package gains a `session` key — re-pushed only when a fill moved the aggregated sources (position open/close/update or a working order), so an unchanged package omits it. Shape (spec item A; realized is net of fees, fees separate, `total = realized + unrealized`):

```
{ "session": {
    "id": "<session token>", "startedAt": 1789577000000, "markedAt": 1789577600000,
    "realized": "18.460914", "fees": "0", "unrealized": "0", "total": "18.460914",
    "venue": [ { "exchgId": "exchgLighter",
                 "realized": "...", "fees": "...", "unrealized": "...", "total": "...",
                 "turnsClosed": 1, "turnsOpen": 1, "openPositions": 0, "openOrders": 2,
                 "exposure": "52383.223756" } ],
    "algo":  [ { "algoId": "algoEmber",
                 "realized": "...", "fees": "...", "unrealized": "...", "total": "...",
                 "turnsClosed": 1, "turnsOpen": 1, "winCount": 1, "lossCount": 0 } ] } }
```

Account totals are re-summed from the venue rows at build time, so the account block and its rows cannot drift apart. The block is also pullable through `t:"q"` with `op:"sessionBlock"` (§11.4). Money fields are decimal strings, consistent with the `rpnl`/`fees`/`upnl` labels in §5.3 and in the BE implementation addendum to `protocol-v4-notes.md`.

2.11.4 11.4 The q/qr pull channel (additive, `queryOps` cap)

A second query transport for the same pipeline, one result frame per request id:

```
FE -> { t:"q", d:{ qid, op, limit?, cursor?, args? } }
BE -> { t:"queryResult", d:{ qid, ...the queryResult of the underlying surface... } }
```

op registry: `sessionBlock`, `instruments` (the static directory; `complete:true`, no paging, `result` carries the caps set), `ordersOpen`, `ordersArchived`, `positionsOpen`, `positionsArchived`, `executions`, and `turnsArchived` (explicit `unsupportedOp` until turn records exist). `args` accepts `algoId/exchgId/state` single values; `*Open` ops pin `liveOnly`. `qid` is opaque and echoed out of band on the result; the envelope `seq` still advances, so gap detection is unaffected. `qid` or `op` missing answers `result.ok=false, code:"badQuery"` with the `qid` echoed.

2.11.5 11.5 `sub` as built (S3-interim)

`sub`/`unsub` are answered with a real acknowledgement — `t:"subAck"` carrying `result:{ok:true}` and the request `rid` — because `caps` advertises `sub` and the FE probes one category at a time, gating on caps. Per-category push filtering is the S3 bandwidth win and is not implemented in this build: the acknowledgement is a capability signal, and the FE accepts any superset of what it asked for (every category still rides the periodic package). `unsub` acknowledges the same way.

2.11.6 11.6 Push and resync addendum (additive)

- the periodic `snap` package now carries `fills` (drained since the last tick), `positions` (live set plus pending closed/reduced records, re-published only when a fill touched it, §5.3) and `session` (§11.3). The standalone `fills` frame per execution is unchanged; the in-package copies exist so a FE that tracks the package still grows its blotter.
- a `resync` for `positions` answers the authoritative live set **plus** every position closed within the retained window — the terminal records with final `rpnl`/`fees`/`tcl`. The FE can rebuild the closed blotter from this one scope; `orders` and `account` answers are unchanged.

2.12 12. Reserved v4 message types

`command` is reserved and answered with `result:{ "ok":false, "code":"unsupported", "msg":... }` and the request `rid` — never silently dropped (F3: silence is a bug). `query`, `q`, `sub` and `unsub` are live as documented in §§11.2–11.5.

Part II

Part II - The BE side

Chapter 3

Notification Pipeline — Private Trading & Account Data

Status: normative for all exchange adapters and the order managers. Owner area: `src/managerOrderBase.*` (hub), `include/ttTrader/exchange/*` (plugin surface), `src/storePositionJournal.*` (persistence).

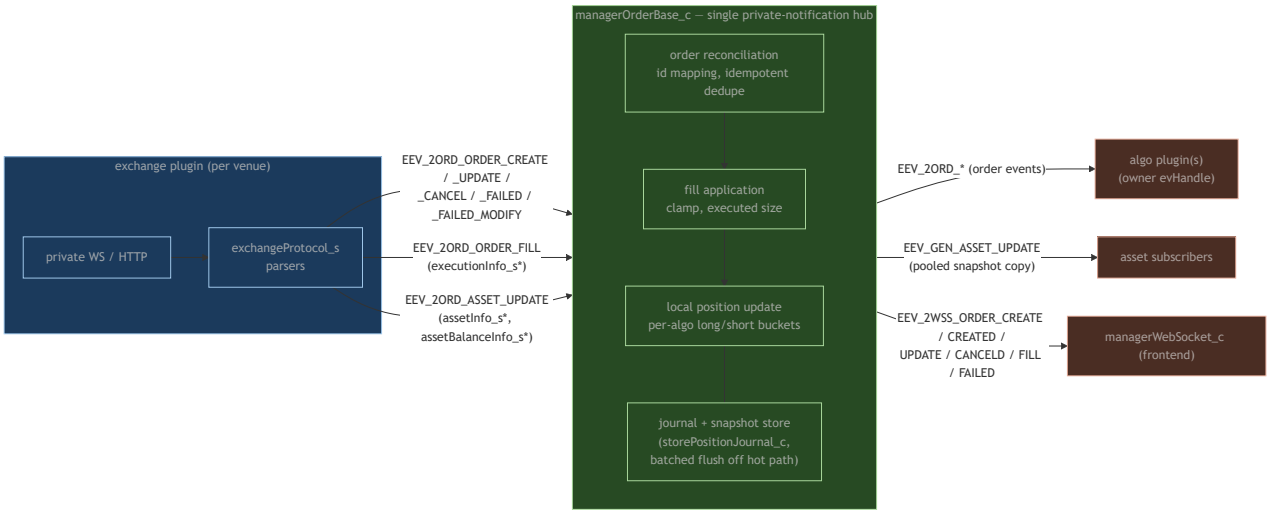
3.1 1. Purpose

Every private notification a venue emits — order lifecycle, execution, cancel, liquidation/forced deletion, wallet/balance+margin — flows through **exactly one hub**: `managerOrderBase_c`. The hub normalizes, reconciles, applies local position logic, persists to the journal, and only then fans out to consumers.

Rules:

- Exchange protocols never post to algo plugins, to the frontend (`ESRV_WEBSOCKET_FE`), or directly to asset/instrument subscribers. They post to `ESRV_ORDER` and stop.
- Positions are **local truth**: derived exclusively from executions routed through this pipeline. Venue position streams are parsed as signals but are never consumed as state (multi-algo netting and out-of-order arrival make venue position data unusable for per-algo buckets).
- The account interface carries **balance + margin values only** (used / available / locked / equity / session PnL).
- The frontend JSON message shapes (`EEV_2WSS_*`) are stable; every venue must reach full order/execution/wallet parity without changing their schema.

3.2 2. Canonical flow



3.3 3. Event contract (ownership table)

Event	Producer	Hub handler	Fan-out owned by hub
EEV_2ORD_ORDER_CREATE	protocol (venue ack of create / replace leg)	_onEventOrderCreate	algo EEV_2ORD_ORDER_CREATE, FE EEV_2WSS_ORDER_CREATED
EEV_2ORD_ORDER_UPDATE	protocol (modify ack, stop conversion, ratchet apply)	_onEventOrderUpdate	algo EEV_2ORD_ORDER_UPDATE, FE EEV_2WSS_ORDER_UPDATE
EEV_2ORD_ORDER_CANCEL	protocol (cancel ack)	_onEventOrderCancel; resumes into replace leg when fEmulatedModifyPending	algo EEV_2ORD_ORDER_CANCEL, FE EEV_2WSS_ORDER_CANCELD
EEV_2ORD_ORDER_FILL	protocol (execution stream, incl. liquidation/ADL attribution)	_onEventOrderFill: clamp to remaining, executed size, position/PnL	algo EEV_2ORD_ORDER_FILL, FE EEV_2WSS_ORDER_FILL
EEV_2ORD_ORDER_FAILED	protocol (reject) or hub itself	_onEventOrderFailed	algo EEV_2ORD_ORDER_FAILED, FE EEV_2WSS_ORDER_FAILED
EEV_2ORD_ORDER_FAILED_MODIFY	protocol (modify reject)	_onEventOrderFailedModify (rolls back staged infoModify)	algo EEV_2ORD_ORDER_FAILED
EEV_2ORD_ASSET_UPDATE	protocol wallet/margin parser, sim fill accounting	_onEventAssetUpdate (snapshot pool copy)	subscribers EEV_GEN_ASSET_UPDATE
entity id minting (order, sim execution, position, sim exchange order ids)	order-manager hub (managerOrderBase_c::m_uniqueIdAllocator)	n/a — not an event	locally minted trading identities; live venue execution ids are retained

Position lifecycle publication:

- _updatePosition captures an immutable post-fill position JSON snapshot on the execution before the frontend event is queued. This preserves the exact open, add, reduce, or flattening state when websocket delivery is delayed behind a later fill on the same mutable record.
- A flattening snapshot is marked with tcl and final rpnl. The websocket manager includes it in the next snap.positions push for at least that push cycle, while the live vector contains only still-open positions.
- Position ids are minted per position instance and are not reused after close; a later reopen receives a new id even on the same instrument and algo.

Ownership rules:

- The event ids in the left column are the **only** private events protocols may post, and only to ESRV_ORDER.
- EEV_GEN_ASSET_UPDATE is posted by nobody except the two hubs (production + simulation). Legacy direct posts from protocols are removed.
- Frontend events are posted exclusively by the hub. No protocol ever targets ESRV_WEBSOCKET_FE.
- Entity ids are minted exclusively by the order-manager hub through uniqueIdAllocator_c (include/ttTrader/gener timestamp + counter: (epochMs48 << 16) | seq16). Plugins never mint these ids. Order/position and simulated execution identities are ~10^17 decimal strings on the frontend wire. Log exchange_order_id remains an unquoted integer (see the export contract). Uniqueness across restarts relies on durable range reservations: a 24-byte record (magic, throughMs floor, checksum) at <logRoot>/unique_id_reservation.bin holds a 60000 ms window durably before the first mint, is held under an exclusive owner handle for the process lifetime, is renewed from the manager timer, and its floor

only advances; corrupt/locked/unwritable state fails closed at init (`DDT_ERROR_CANNOT_OPEN_WRITE`) instead of minting unreserved. Wall-clock advancement alone proves nothing. Live venue execution ids are retained as-is (possibly nonnumeric, order-scoped). Field-level authority including the composite fill key: `docs/01-system-protocol/frontend-wire-contract.md`.

3.4 4. Processing rules per category

3.4.1 4.1 Order reconciliation

- Orders are keyed by the packed client order id (boot-second epoch prefix + 24-bit counter); venue order ids map onto it once assigned. The mapping is owned by the hub via `orderInfo_s::infoId`; adapters resolve venue ids through `exchgStoreOrders_c`.
- Duplicate/out-of-order venue updates are suppressed by comparing the carried state against the local processing stage: an ack that repeats an already applied transition is dropped; a stale update (older sequence id on venues exposing one, e.g. OKX `seqId`, Bybit `seq`) is dropped and counted.
- Unknown client order ids (orders created outside this process) are ignored by parsers; reconciliation at startup covers them (see §6).

3.4.2 4.2 Executions -> fills -> local positions

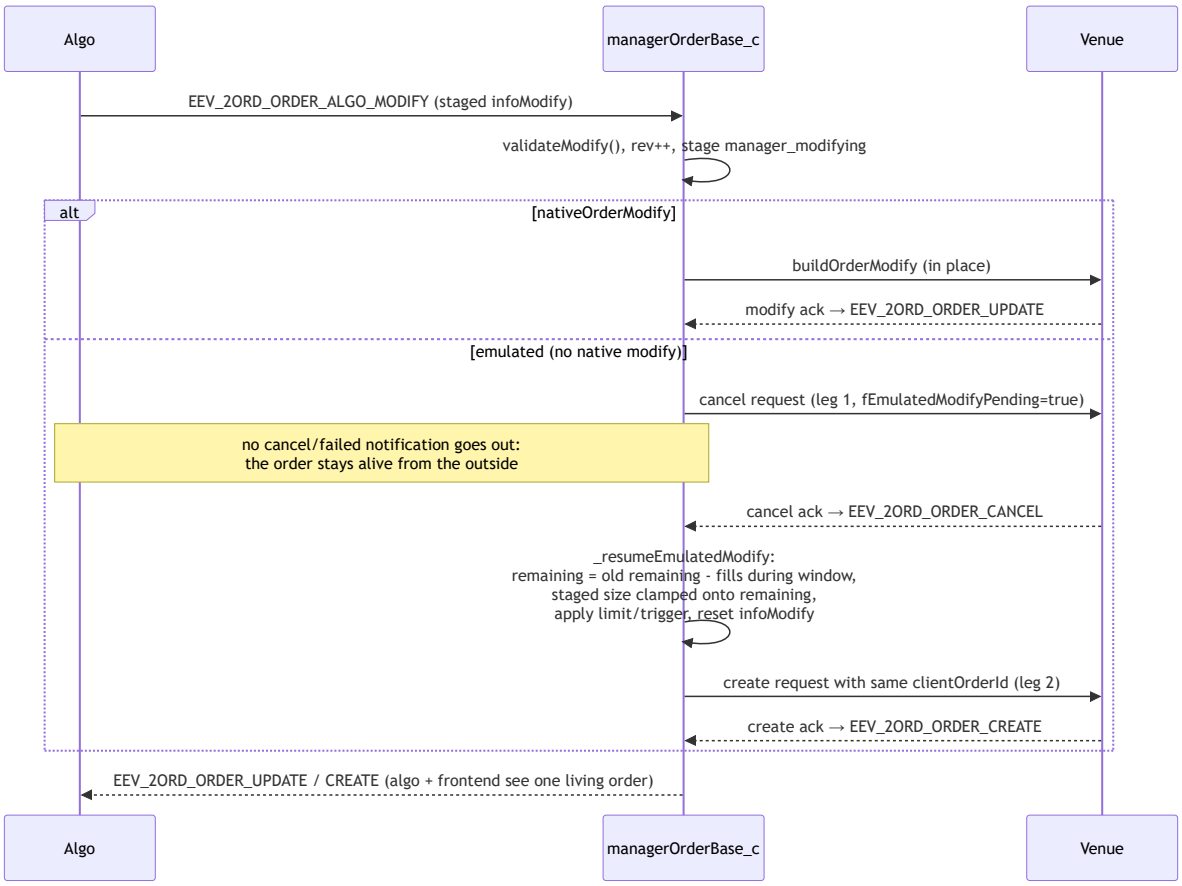
- A fill is clamped to the order's remaining size; overshoot books nothing extra and fees scale proportionally (fee netting stays consistent).
- Executed size accumulates; completion retires the order through the ring.
- `_updatePosition` books each fill into the per-algo bucket pair (`pvLong`/`pvShort`) keyed by instrument + position user. Closing fills realize profit only up to the actually open size; the overshoot flips into the opposite bucket instead of booking profit.
- **Non-algo-origin executions** (liquidations, ADL, venue-forced deletions) arrive on the same order stream. A fill whose client order id does not match a live order is attributed to the owning algo's bucket when the adapter can attribute it (venue phase work); otherwise it is logged, journaled with `algoId = 0`, and surfaced as a hub counter so silent corruption cannot occur.
- Simultaneous long and short buckets within one algo are legal and preserved; venue-netted zero positions do not collapse them.

3.4.3 4.3 Wallet / balance + margin

- Parsers build `assetBalanceInfo_s` (spot + margin categories: balance, available, equity, locked, session PnL), append it to the asset's balance ring, and post `EEV_2ORD_ASSET_UPDATE(assetPtr, ringBack)` to `ESRV_ORDER`.
- The hub copies the balance into a rotating snapshot pool before forwarding, so no subscriber can observe a wrapped ring slot. Pool size: `ce_stBalanceSnapshotPoolSize` (256); overflow wraps harmlessly because the forward is posted before the next hub turn.
- Timestamped signature inputs and clock-skew guards are adapter concerns and documented per venue README.

3.4.4 4.4 Emulated modify (cancel + replace)

Venues declare `nativeOrderModify` in `exchgCapabilities_s`. Without it the manager emulates modification:



Partial-fill safety between the legs:

- Fills during the cancel window shrink what can be replaced; the staged remaining volume is clamped down

(never resurrecting filled size).
- A fill that consumes the whole order wins: the normal fill path already retired the order, so the cancel ack falls into the terminal-order branch and the staged modify dies with it.
- If the cancel itself times out (`ce_i32CancelTimeoutSeconds`), the regular sweep expires the order into failed — including its pending modify.

3.4.5 4.5 Frontend wire payloads (`EEV_2WSS_* -> websocket`)

This is the contract the dashboard consumes. It is normative: the frontend relies on it instead of inferring identity from message content.

Envelope (every outbound frame):

```
{ "seq": 4127, "t": "fills", "d": { "…": "…" } }
```

- `seq` — process-local outbound delta sequence; initial snapshots and errors use `0`.
- `t` — message section. The frontend-relevant values:
  - `"snap"` — snapshot, `d` carries `states`, `algos`, `exchanges`, `acct`, `instruments`, `trades`, `quotes`, `orders`, `fills`, `positions`, `nextSeq`, `playbackFiles`;
  - `"order"` — single order create/update/failed (`orderInfo_s::toJson`), also used for the order snapshot;
  - `"fills"` — single execution (`executionInfo_s::toJson( true )`);
  - `"exchg"`, `"instr"`, `"pb"`, `"error"` — exchange/instrument/playback/ error sections.
- `d` — section payload (object for `order`/`fills`; the `snap` object listed above carries arrays).

Identity fields and their guarantees (field-level authority: `frontend-wire-contract.md`):

payload	field	identity
fills	`id`	opaque execution string — retained venue id (possibly nonnumeric/scoped), or allocator-minted simulated id; key with `order.id`
fills	`order`	owning order serialized in full; `order.id` is its decimal-string parent identity
fills	`pos`	booking-time position (`positionInfo_s::toJson( true )`); omitted if nothing booked; identity is `pos.id`
order	`id`	order parent id; opaque decimal string
order	`rev`	numeric modify revision; `(id, rev)` identifies a revision, not a new parent
position	`id`	position record id; opaque decimal string

- FE deltas and snapshots use `executionInfo_s::toJson(true)`: `order` plus optional `pos`, not `oid`/`pid`. The non-FE short form (`false`) instead emits decimal-string `oid` and `pid` (`"0"` if nothing booked). Its fill key is `(oid, id)`; the FE key is `(order.id, id)`. Encode tuples without ambiguous concatenation of arbitrary venue strings.
- Allocator restart uniqueness depends on durable reservation state (§3). Live fill ids are not deployment-global; do not key on raw `id` alone or compare identities of different entity kinds.
- Embedded `pos` and short-form `pid` refer to `bookedPositionPtr`, not the order's later `positionPtr`. They are alternate renderings, not two fields emitted together on the FE websocket.

- **One fill delta per tuple per connected session.** A second `fills` delta for `(order.id, id)` in that session is a producer defect. Reminting an id to conceal a replay is also a defect. Snapshots legitimately repeat retained fills and must merge by tuple without booking them as new activity.
- Both numeric and string venue executions are deduplicated per order by `_executionDedupeCheck` before fan-out, within its bounded retained window. This is not an unlimited persistent dedupe ledger or proof of FE receipt.
- No transport acknowledgement or exactly-once delivery across disconnect is promised. `seq` is process-local, with `0` for initial snapshots and errors; use snapshot `nextSeq` for resynchronization, not global fill identity.
- Delete content-fingerprint dedupe. Matching instrument/side/size/price/time with distinct tuples represents distinct fills.
- Algo payloads (snapshot `algos`, delta `algos`) carry `instr`, an array of `{ id, eid }` naming the instruments the algo subscribed to and their venue. It is filled by the algo store at load, static afterwards, and emitted on snapshots and deltas alike so a frontend that missed the snapshot still knows the coverage. `id` joins the `instruments` section for symbol/details.
- Order payloads may carry `cr`, the algo-supplied cancel reason, only when the owning algo set one at pull time (for example `"requote"`, `"pull gate"`, `"flatten"`). It is absent on creates/updates and on manager-initiated cancels that carry no algo reason; treat it as optional and never infer a reason from message content.

Id serialization (implementation note for the frontend):

- `executionId_s::toString()` preserves string venue ids and renders numeric execution ids as decimal strings. `uniqueIdToString()` renders order and position ids. These guarantees concern trading identities, not unrelated fields such as numeric `playbackFiles[].id`.
- Allocator values (~1.2e17) exceed JavaScript's 2^53 safe-integer range. FE order/position `id` and short-form `oid`/`pid` are now strings; execution `id` remains a string. `rev` stays numeric.
- Compatibility `String(v)` handles legacy numeric ids only when they were safe integers before coercion; it cannot restore rounded digits. Upgrade FE string handling before sending high-magnitude ids. Full migration and examples: `docs/01-system-protocol/frontend-wire-contract.md`.

3.5 5. Capability steering

The order-type/time-in-force behavior above is steered by the adapter-declared constexpr `exchgCapabilities_s` (`ce_capabilities` beside the wire code in each adapter). It exports across the DLL boundary once at load (`getExchgCapabilities`) and lands in the runtime consumption fields (`exchangeInfo_s::fNative*`). There are no capability JSON keys anymore; legacy `native*` keys fail the config load. Invalid combinations (trailing without stops, stops without trading) are compile-time errors inside the adapter. Simulation/playback force broker-side emulation regardless of declared bits.

3.6 6. Persistence & recovery (event sourcing)

`storePositionJournal_c` implements the journal + snapshot checkpoint model:

1. Every processed private notification is appended (batched, off hot path): `PJR_ORDER_EVENT`, `PJR_EXECUTION` (with `algoId` attribution + venue execution id), `PJR_WALLET`. Sequence numbers are monotonic per journal.
2. Periodic snapshots checkpoint the derived state: per-algo position buckets (including simultaneous long+short), open orders, last venue sequence ids, and the journal watermark they cover.
3. Restart recovery sequence:
   - load newest intact snapshot (`readLatestSnapshot`),
   - replay the journal tail past the watermark (`replaySince`),
   - bounded replay of venue execution/fill history covering only the offline gap (last journaled execution id/timestamp -> now; per-venue pagination),
   - seed account balances/margins through the account interface,

- reconcile open orders against the venue's open-order list (venue truth for orders, local truth for positions).

4. Durability vs latency: appends batch in memory and flush to the OS in bulk; fsync happens on explicit flush points only. The unflushed tail is the crash window; recovery treats it as "never happened" and the bounded venue gap replay covers it. A torn final record fails its CRC chain and is invisible to replay and overwritten by subsequent appends.

3.7 7. Failure handling summary

Condition	Behavior
Late fill after order went terminal	accounted exactly once (idempotent position lookup), no state transition
Fill whose order object is fully evicted	dropped, execution released, logged
Cancel timeout beyond 30 s	order expires into failed; late venue ack handled idempotently
Modify rejected by venue	staged values rolled back (`infoModify.reset()`), in-flight guard released
Partial fill inside emulated-modify cancel window	replace size clamped onto remaining
Duplicate `(order, execution id)`	suppressed before position application within the retained per-order dedupe window; one fill delta per tuple per connected session is required, while snapshots may repeat state; no content fingerprint or transport receipt guarantee
Wallet ring wrap under async subscriber	impossible through hub (pooled snapshot copies)

Chapter 4

BE reply to the FE v4 answers (2026-09-21)

Status: **accepted with one correction and one confirmation.** The FE's answers to §10 are adopted; the `rev` prerequisite section is correct and has been escalated to a BE defect.

4.1 1. Adopted as-is

FE answer	BE position
Batching: `order`/`fill`/`fills`/`pos` never batched; `quotes`/`trades`/`acct`/`algos`/`instruments`/`exchanges` batchable	**Agreed.** Matches the BE plan §4.3. Note the fill atomicity caveat: the BE will not batch a fill while it carries embedded `order`/`pos` blocks.
`maxHz` defaults (5 Hz blotter, 2 Hz quotes panel, 5–10 Hz focused, 1 Hz mobile, 1–2 Hz risk/acct)	**Agreed**, as a ceiling not a promise. Defaults live in the BE `caps`/limits so both sides read the same numbers.
Candles as the marketdata currency, tagged by base	**Agreed.** The BE has `trade` and (where available) `midpoint` series natively; both are instrument-scoped, which matches the FE's storage model.
Series tagged with base **and** interval, numeric interval (`intervalSec`), per-instrument interval set	**Agreed.** This is a genuine BE-side requirement — the BE must publish the interval set and how much history per interval it sends on connect (FE item 3).
Candle shape `{time, open, high, low, close, volume}`, `time` = bucket start ms, BE re-sends the **current** bucket, FE replaces by `time`	**Agreed.** The BE will also emit a final/closed flag so the FE can stop merging into a bucket.
Last trade + BBO covers the quotes panel; the tape needs a low-rate trade push or a query	**Agreed** — the tape becomes a `query` for the focused instrument.
Turn/aggregate ownership -> BE; the FE's approximation inventory becomes dead code	**Agreed.** See §3 below for the exact fields the BE will expose.
Resume sized by **time (~5 min) + byte budget**, all-or-nothing per scope, with a stated failure reason	**Agreed**, and better than a frame-count ring. The BE will report `resumeFailed: "tokenTooOld" | "sessionChanged" | "byteBudget"` and snapshot that scope.
JSON-only v4 as phase 1, binary as measured phase 2	**Agreed.** The BE plan already keeps encoding behind one seam.

FE answer	BE position
Four binary-preservation rules (schema-mappable shapes, ids as strings, per-connection negotiation, JSON debug mode)	**Agreed**, including unifying `t:"fill"` vs `t:"fills"` before freezing a schema.

4.2 2. The `rev` prerequisite — confirmed, and it is worse than reported

The FE's evidence is correct and the assessment is right: **do not make `rev` authoritative yet.** Verified on the BE side: **`rev` is not systematically incremented at all.**

`ui32Rev` has exactly **four** write sites in the entire codebase:

site	what it does
`managerOrderBase.algoOrder.cpp:140`	resets `rev = 0` on create
`managerOrderBase.algoOrder.cpp:257`	increments on a rejected modify *(added 2026-09-21)*
`managerOrderBase.algoOrder.cpp:272`	increments on an accepted modify
`orders/orderTrailingStopHandler.cpp:156`	increments on a trailing-stop adjustment

Against that, the order **state** is assigned in **25 distinct places** — `managerOrderBase.events.cpp` (`executed`, `failed`), `managerOrderBase.transport.cpp` (`active`, `expired`, `failed`), `managerOrderSim.cpp` (`active`, `canceled`, `executed`, `failed`), `managerOrderBase.algoOrder.cpp` (`failed`, `deleted`, `pending`, `active`, `canceled`), `managerOrderBase.cpp` (`deleted`, `idle`, `active`), `orderStopHandlerBase.cpp`, `exchgManagerTradingT.h`, and more.

So a fill, a cancel, a reject, an expiry and a venue ack **all emit a state change with `rev` unchanged** — exactly what the FE captured (`st=2 rev=0 -> st=6 rev=0`). `rev` today only tracks *modification attempts*, which is why `(id, rev)` orders nothing useful.

Conclusion for the plan: `(id, rev)` ordering is a **BE work item first**, and the FE's caution is adopted — v4 keeps `timeUpdate` as the ordering key until the BE bumps `rev` on every mutation, at which point the FE implements `(id, rev)` ordering with terminal immutability overriding any higher `rev`.

4.2.1 BE fix required (tracked)

Centralize the transition, so no site can forget: a single `orderInfo_s::markState( newState )` that assigns `eState`, updates `infoTime.dtLastUpdate`, and increments `ui32Rev` — then route all 25 assignment sites through it. Companion rules:

- `rev` is monotonic for the lifetime of the id; `rev` is **not** reset on a state change.
- `rev = 0` is the create revision.
- Terminal states are immutable: no assignment may follow a terminal state, and that must be an assertion, not a convention.
- The modify-attempt bumps at `algoOrder.cpp:257/272` remain, so a modify attempt advances `rev` even before the venue answers.

Per the repo build rules this touches `orderInfo_s`, which is an ABI-visible struct shared by the exe and every plugin — implementing it requires a full plugin-set rebuild/redeploy, and it is best done together with the v4 work rather than as a hotfix.

4.3 3. Aggregates the BE will own (answers FE §3)

Authoritative and exact, per the FE's list: per-algo realized + unrealized, per-symbol, per-exchange, and the account session PnL. The reconciliation rule to document:

- Each algo keeps its **own** balance sheet, so per-algo figures are the primary truth.
- The account figure is per-account **session** PnL; the sum of per-algo figures reconciles to it only once the BE exposes realized **and** unrealized per algo (which it now will).
- When they do not reconcile (e.g. a position opened before the session boundary, fees booked at account level, or a manual/venue-side adjustment), the BE must state the delta and its cause rather than letting the FE present two bases side by side — which is exactly the `PnLContext` inconsistency the FE flagged (`fromPositions/bySymbol/byAlgo` from BE positions vs `fromTurns/byExchange` from FE turns).

Also owned by the BE: turns, equity time series (bucketed, B6), VWAP deviation, and markout aggregates — which is what retires the FE's `useTurnTracker`, equity sampler, `useMarkoutMetrics` buffers and the `vwapDeviation` dependency on 1m trade candles.

4.4 4. Still open

1. **History retention per category** — the BE's proposed boundary (memory ring for the recent window, file-backed paging beyond) needs the FE to say what recent window it wants, so the `caps` figures are useful rather than arbitrary.
2. **`caps/limits` payload shape** — the BE will propose the exact fields (version, batching defaults, interval sets, history depth, retention boundary) as part of the v4 schema draft.
3. **Ordering-key transition** — confirm the FE is fine with `timeUpdate` remaining authoritative in v4 phase 1 and switching to `(id, rev)` only once the BE's `markState` centralization lands (it is a full plugin redeploy).

4.5 BE implementation addendum (2026-10-11) — money semantics, timestamps, tac-less states

Implemented in ttTrader (BE) this session; this note is the contract-facing half (spec items F, G, and the state whitelist from the tac requirement).

4.5.1 Fee semantics per payload (item F)

payload	field	semantics
position records (live + retired)	`rpnl`	realized, **net of fees** (position getPnL(false) includes booked fees)
position records	`fees`	the fees booked against this position, separate
position records	`upnl`	unrealized, mark-to-market, gross (fees for the open part not yet booked)
session block	`realized` / `fees` / `unrealized` / `total`	`realized` is the sum of the position `rpnl` values (net), `fees` separate, `total = realized + unrealized` — the account block is the re-summed venue rows, so rows and totals cannot drift

Gross-only payloads: none carry a gross-only realized figure; consumers wanting gross compute `rpnl + fees`. This satisfies "net + separate fees" for the session block and labels every other realized figure as net.

4.5.2 Timestamp semantics per field (item G)

All timestamps are **epoch-ms, UTC, one clock** (the BE host clock, `dateTime_t::now()`).

field	when
`tcr` (orders)	algo-object creation, before any dispatch
`tac` (orders)	gateway hand-off — production: the post into the exchange adapter transport; sim: the simulated venue ack, AFTER the configured `orderLatencyMs` hop
`tup` (orders, positions)	last state transition (ack/modify-ack/fill/failed-modify/retire)
`tcl` (positions)	position close (terminal)
`top` (positions)	position open
`tcr` (positions)	position record creation

Invariant: `tac >= tcr` (a violation is logged BE-side as a clock fault); `tcl >= top`; `tup` is monotone per record.

4.5.3 tac-less states (whitelist)

`tac` is present and nonzero for every order in EOS_ACTIVE(2), EOS_FILLED(3), EOS_DELETED(5), EOS_CANCELED(6), EOS_EXPIRED(18). Records may legitimately lack a nonzero `tac` only in: **EOS_IDLE(0)** (created, never dispatched), **EOS_FAILED(4)** (never dispatched), **GAD-deferred EOS_PENDING** (activation scheduled, dispatch not yet due), and **EOS_CANCELING / manager-side processing states** that predate the first gateway hand-off. The FE whitelist (IDLE/FAILED/PENDING) is correct as-is; no loosening was required.

4.5.4 sim latency note

BROKER-located orders (locally held protective stops) never cross a gateway: their create->activate is in-process (<1 ms) and no simulated latency applies. The configured `orderLatencyMs` models the hop for EXCHANGE-located orders (stamped at the simulated ack) and delays fill eligibility, modify landing and cancel landing. Lighter tier reference: maker placement 0 ms, taker/cancel/ modify 300 ms (standard/plus), 140/0 ms (premium) — apidocs.lighter.xyz/docs/account-types.