NanoTick NSE — Market-Data Flow & Class Reference

Technical review of the implemented NSE F&O market-data pipeline and its key classes/entities

Repository: nanotick_nse · **Branch:** integration/md-app · **Commit:** 2948133 (pushed to origin) · **Suite:** 709 passed / 0 failed · **Date:** 2026-07-06

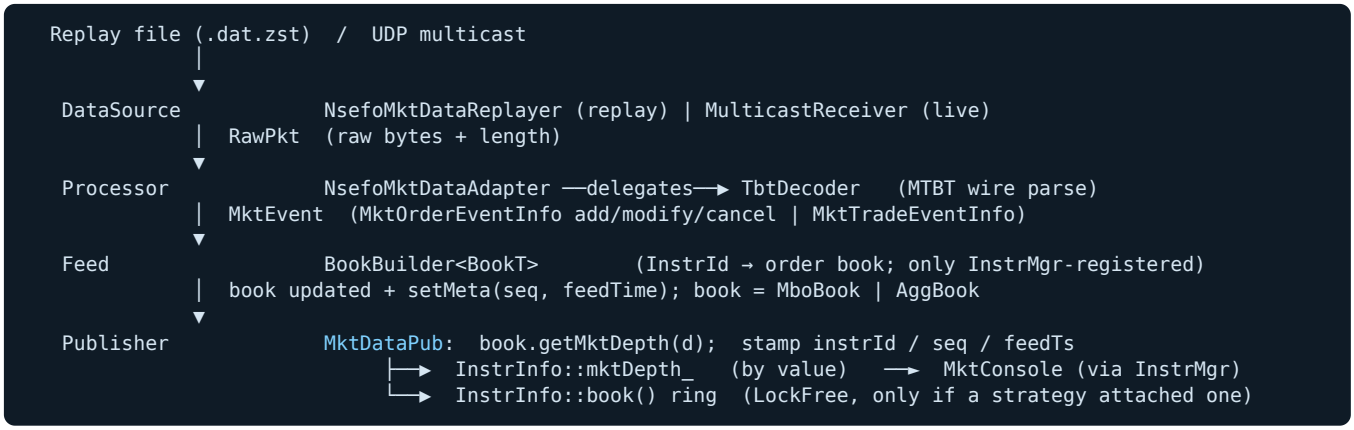
1. Overview

NSE F&O market data flows from the wire (replay file or live multicast) through a compile-time-composed pipeline of four orthogonal policies into a per-instrument depth snapshot that consumers read. The pipeline is identical for replay and live after the data source.

- **Compile-time composition** — the engine is a template; once instantiated the loop is monomorphic (no virtual dispatch per tick).
- **One pipeline, two sources** — swap only the DataSource for replay vs live.
- **No hot-path allocation** — reusable packet buffer; the book fills a POD snapshot in place.
- **Consumers never touch the live book** — they read flat `MktDepth` copies.
- **Generic core + thin venue binding** — reusable templates in `core/` / `mkt/common/`, NSE specifics as one-line bindings.

2. End-to-end pipeline

Composed by `trd::TradeEngine<DataSource, Processor, Feed, Publisher>`. Loop per record: `receive → process → (feed updates book) → publish`.



Step by step: (1) DataSource fills a `RawPkt` — one framed TBT record (file) or one datagram (live). (2) `NsefoMktDataAdapter` hands it to `TbtDecoder`, which emits normalized `MktEvent`s. (3) `BookBuilder` applies each event to that instrument's book. (4) `MktDataPub` extracts a `MktDepth` and writes it to the instrument (+ ring if a strategy subscribed).

3. Policy contracts (the four seams)

Policy	Required interface	Implementations
DataSource	`bool init()` · `bool receive(RawPkt&)` (false = none) · `bool eof() const`	NsefoMktDataReplayer, MulticastReceiver
Processor	`void init(Feed&)` · `bool process(const RawPkt&)`	NsefoMktDataAdapter<Feed>
Feed	`onOrder(MktOrderEventInfo)` · `onTrade(MktTradeEventInfo)` · `onHeartbeat(...)` · `book(InstrId)`	BookBuilder<BookT>
Publisher	`bool onBook(const AbstractBook&, Timestamp, uint64_t seq, InstrId)`	MktDataPub, ShmPub

4. Class & entity reference

4.1 Engine & pipeline runner (trd/, header-only)

Type	Responsibility & key API
`TradeEngine<DS,Proc,Feed,Pub>`	Owns nothing venue-specific; holds the four policies by reference and runs the loop. `ctor(source, processor, feed, publisher)`; `bool init()` (init source + `processor.init(feed)`); `uint64_t run(const std::atomic<bool>* extStop=nullptr)` → returns records processed; `void stop()`. After each record it publishes the just-updated book: `publisher.onBook(*book, feed.lastFeedTimeUs(), book->lastUpdateSeq(), id)`.

4.2 Data sources & transport (core/io, mkt/common, core/sock)

Type	Responsibility & key API
`RawPkt` `core::io`	Reusable fixed-capacity packet buffer (no hot-path alloc). `uint8_t* data()` · `uint32_t length()/setLength()` · `static capacity()=MAX_PACKET_SIZE`. Backing `buffer_[MAX_PACKET_SIZE]`.
`MktDataReplayer<HeaderT>`	Generic replay DataSource; frames length-prefixed records via `core::io::RecordReader<HeaderT>`. `init()/receive(RawPkt&)/eof()`. Record length comes from `RecordTraits<HeaderT>::recordLen()`.
`NsefoMktDataReplayer`	`= MktDataReplayer<tbt::Header>` (NSE binding).
`MulticastReceiver`	Live DataSource; non-blocking multi-stream UDP over `core::sock::McastReceiver`. `ctor(vector<McastStream>)`; `init()/receive(RawPkt&)`; `eof()` always false (live never ends). Venue-neutral.
`RecordTraits<tbt::Header>`	Replay framing binding: `recordLen(h) = h.msg_len`.

4.3 Decode (mkt/common, mkt/decode/nsefo)

Type	Responsibility & key API
`MktDataAdapter<Decoder,Handler>`	Generic Processor; the framework seam (future sequence-gap recovery lives here). `init(Handler&)`; `process(RawPkt)` → `decoder.decode(data, len, handler)`. Owns a `Decoder` by value.
`NsefoMktDataAdapter<H>`	`= MktDataAdapter<TbtDecoder, H>` (NSE binding).
`TbtDecoder`	NSE MTBT v6.7 wire parser. `template<Handler> bool decode(buf, len, handler)` — dispatches by message type: `N/M/X/G/H/J` → order add/modify/cancel → `onOrder`; `T/C/K` → `onTrade`; `Z` → `onHeartbeat`. Non-virtual method template (inlines into the hot loop). Swap protocols by swapping the decoder type.
`tbt::Header / OrderMsg / TradeMsg / HeartbeatMsg`	Wire structs. `Header{ int16 msg_len; uint32 seq_no (1-based/day; 0=heartbeat) }`; `HeartbeatMsg{ last_seq_no }`.

4.4 Normalized events (mkt/common/MktEvent.h)

Type	Fields
`MktFeedEventInfo`	base: `seq_` (feed seq), `instrId_`, `size_`, `price_`, `feedTime_`.
`MktOrderEventInfo : base`	+ `evtType_` (MboEventType), `side_` (OrderSide), `orderId_` (LongOrderId). MBO add/modify/cancel.
`MktTradeEventInfo : base`	+ `execType_`, `buyOrderId_`, `sellOrderId_`. Matched buy/sell.
`MktPriceEventInfo`	L1 quote (for venues that publish L1): `bidPrice_/bidSize_/askPrice_/askSize_/lastPrice_/lastSize_/feedTime_/sysTime_`.
`enum MboEventType`	`ADD, UPD, REM, TRD` (+ INVALID).

4.5 Feed (mkt/common/BookBuilder.h)

Type	Responsibility & key API
`BookBuilder<BookT>`	Event→book glue; keeps `InstrId → unique_ptr<BookT>`, built **only** for instruments registered in `InstrMgr`. `onOrder(e)/onTrade(e)/onHeartbeat(...)`; `book(InstrId)` (const & mutable); `lastFeedTimeUs()`; `lastInstrId()` (instrument touched by the most recent update); `bookCount()`.

4.6 Order books (mkt/book, lib nts_mkt_book)

Type	Responsibility & key API
AbstractBook	Base for all books. `instrId()`, `lastUpdateTime()`, `lastUpdateSeq()`; pure `virtual void getMktDepth(MktDepth& out) const` (single book→snapshot extractor: fills BBO + top-kDepth); protected `setMeta(seq, ts)` stamped per event. Non-copyable.
PriceLevel	Internal per-level value: `price_`, `qty_` (Size), `orders_`. Used by book storage & tests.
MboBook	Market-by-order (L3). `onNew/onCancel/onModify(MktOrderEventInfo)`, `onTrade(MktTradeEventInfo)`, `getMktDepth()`. Internals: `OrderMap orders_` (orderId → OrderNode); per side a `SideBook{ PriceLevelNode* bestLevel; LevelMap levels }`; `PriceLevelNode{ price, totalQty, orderCount, head/tail OrderNode*, prev/next }` (intrusive best→worst list); `OrderNode{ orderId, price, qty, side, prev/next, level }` (FIFO per level).
AggBook	Price-aggregated view. `onNew/onCancel/onModify/onTrade`, `getMktDepth()`; helpers `updateBid/Ask`, `removeBid/Ask`, `getBestBid/Ask`, `bidLevels()/askLevels()`. Storage: `std::map<Price,PriceLevel,greater> bids_`, `std::map<Price,PriceLevel> asks_`.

4.7 Snapshot & lock-free transport (mkt/common/MktDepth.h, core/lock)

Type	Responsibility & key API
MktDepth	Dependency-light POD snapshot handed to consumers. Fields: `instrId, seq, feedTsUs, hasBid/hasAsk, bidPx/askPx, bidQty/askQty, nBids/nAsks, bids[kDepth], asks[kDepth]` (`kDepth=5`). No pointers/containers → copyable, tear-tolerant, cannot crash a reader.
BookLevel	One depth level in `MktDepth`: `px, qty, orders`.
LockFree<T,N> `core::lock`	Generic single-writer / multi-reader **overwrite** ring (per-slot seqlock). `publish(const T&)` (writer, never blocks, overwrites oldest); `readLatest(T& out)`; `readNext(uint64_t& cursor, T& out)` (lapped reader skips to latest); `latestSeq()`; `capacity()`. `T` trivially copyable, `N` power-of-two ≥ 2. Readers keep no state in the ring (no registration, no cap).
Book (alias)	`= core::lock::LockFree<MktDepth, kBookSlots=16>` — the per-instrument strategy ring.

4.8 Publishers (mkt/common, lib nts_mkt_md)

Type	Responsibility & key API
MktDataPub	Publisher policy. `onBook(book, ts, seq, instrId)`: look up `InstrMgr`; `book.getMktDepth(d)`; stamp `instrId/seq/feedTs`; `instr->setMktDepth(d)` (console) and, if `instr->book()!=null`, `ring->publish(d)` (strategy). `bboChanges()` counter.
ShmPub	Shared-memory publisher — currently a no-op stub (planned real implementation).

4.9 Instruments (instr/, lib nts_instr)

Type	Responsibility & key API
InstrInfo	Full instrument master record. Identity/metadata: `getId/getSymbol/getInstrType/getExchange`, `priceMult/sizeMult/lotSize/contractSize`, `expiry/strike/optionType`, `TickRange`. Market data: latest `mktDepth()` / `setMktDepth()` (by-value snapshot, console reads) + optional `book()` / `setBook()` (strategy ring pointer, null by default).
InstrMgr	Global instrument registry (singleton). `instance()`; `InstrInfo* get(InstrId)`; `InstrInfo* get(const Symbol&)`; `bool add(InstrInfo*)`. Read-only after load (shared by engine threads).
GenInstrLoader	Loads the standard `#NTS-CONTRACT v1` CSV → builds InstrInfos + registers in InstrMgr. `bool load()`; counters `loaded()/futures()/options()/duplicates()/skipped()`; `instruments()` → `vector<InstrRef>`.
InstrRef	Lightweight index entry for symbol search/listing: `symbol, token, type`.

4.10 Application & console (app/, util/, core/ui, common/)

Type	Responsibility & key API
nanotick (app)	One binary, runtime-selected + compile-time-composed policies. CLI: `-m replay|live` (req), `-d YYYYMMDD` (req, drives PROD paths), `-e NSE_FO|BSE_FO|MCX_FO|NSE_CM|BINANCE` (req; only NSE_FO wired), `-b mbo|agg`. Startup dispatch: `dispatchExchange → dispatchBook → runInteractive<BookT>` (one nested switch, then a monomorphic loop). Runs engine thread(s) in background; main thread runs the console.
MktConsole `util`	Interactive MD console. `ctor(index, priceScale)`; `run()` (blocks until quit). Commands: `<text>` substring search (grouped futures/options, hides NSETEST), `<symbol>` / `<token>` select+show book, `<blank>` refresh, `list`, `help`, `quit`. Reads `InstrMgr::get(token)->mktDepth()`.
core::ui::Cli	Generic REPL (venue-neutral). `addCommand(name,help,Action)`, `onEmpty(Action)`, `onDefault(DefaultAction)`, `addHelpLine`, `setBanner`, `run(istream&)`; built-in help/quit.
ExchangeSegment	`enum class { NONE, NSE_CM, NSE_FO, BSE_FO, MCX_FO, BINANCE }` + `parseExchangeSegment()` / `exchangeSegmentName()`. Drives the `-e` policy switch.

Constant.h	`kContractBase="/home/common/contract"`, `kMktDataBase="/home/common/mkt_data"`, `kNsePriceScale=100` (paise→rupees for display).

5. Threading

Mode	Threads	Shape
Replay	1	One thread runs the whole composed `TradeEngine` (read → decode → book → publish). Deterministic.
Live	N = stream batches	Streams grouped into batches; **one thread per batch**, each running a full composed `TradeEngine` with its own `MulticastReceiver` (multiplexing that batch's streams), Adapter, Feed, MktDataPub.

> **Single-writer invariant:** instruments partition by stream/batch, so each instrument is produced by exactly one thread — keeping the per-instrument snapshot and ring writes lock-free (one writer per instrument, any number of readers).

6. Correctness of the hand-off

- **Console copy** (`InstrInfo::mktDepth_`) — by-value POD; a concurrent read can at worst tear one frame (mixed old/new fields), never crash (no pointers/containers). Cosmetic for a human console at ~1 Hz. `seq==0` ⇒ "no data yet".
- **Strategy ring** (`LockFree`) — readers **copy out** under a per-slot seqlock (read seq → copy → re-read; retry on mismatch). A reader never aliases slot memory, so a writer overwriting a slot cannot corrupt a consumer's in-hand copy; a reader > N behind laps (skips to latest) rather than reading garbage. The writer never blocks on readers.
- **Live-book safety** — consumers never iterate the live `MboBook` / `AggBook` containers (which the engine mutates); they only see flat `MktDepth` copies.

7. Libraries & status

Libraries: `nanotick` (base/common), `nts_instr`, `nts_core_io|sock|log|time|util|ui`, `nts_mkt_book`, `nts_mkt_md`, `nts_mkt_nse` (INTERFACE), `nts_util`.

- Implemented & green — 709 passed / 0 failed; pushed to `origin` at commit 2948133.
- Planned: real shared-memory publisher (ShmPub stub today), sequence-gap recovery (seam in NsefoMktDataAdapter), configuration, replay timing / benchmarking.