EDUMATCHER · MARKET DATA

CALF

Channel-ALF — the market data protocol, explained from zero

A beginner's guide to the CALF gateway: what it is, what it carries, and how to build a client that never loses a message.

`pm-md-gwy` `TCP :5570` `PROTO=CALF1`

What you will learn

A route from “what is a gateway?” to a working, gap-proof client

1 Why a gateway exists

The job it does between the matching engine and you

2 What the protocol is for

One text line format any language can read

3 What data is available

Seven channels, from best bid to circuit breakers

4 How a session works

Handshake, subscribe, snapshot, stream

5 When things go wrong

Error codes, silent disconnects, and gaps

6 How to replay what you missed

Sequence numbers and the replay window

The last few slides go deeper: full configuration, the complete channel matrix, and every error code.

PART 1

Why a gateway exists

The problem CALF was built to solve

First, what is market data?

Everything an exchange knows, published as it happens

Inside the matching engine

- **Orders arrive**
  buyers and sellers post prices
- **The book updates**
  best bid and ask move
- **Trades happen**
  a buy and a sell match
- **The session shifts**
  pre-open, auction, continuous, closed

What a consumer wants

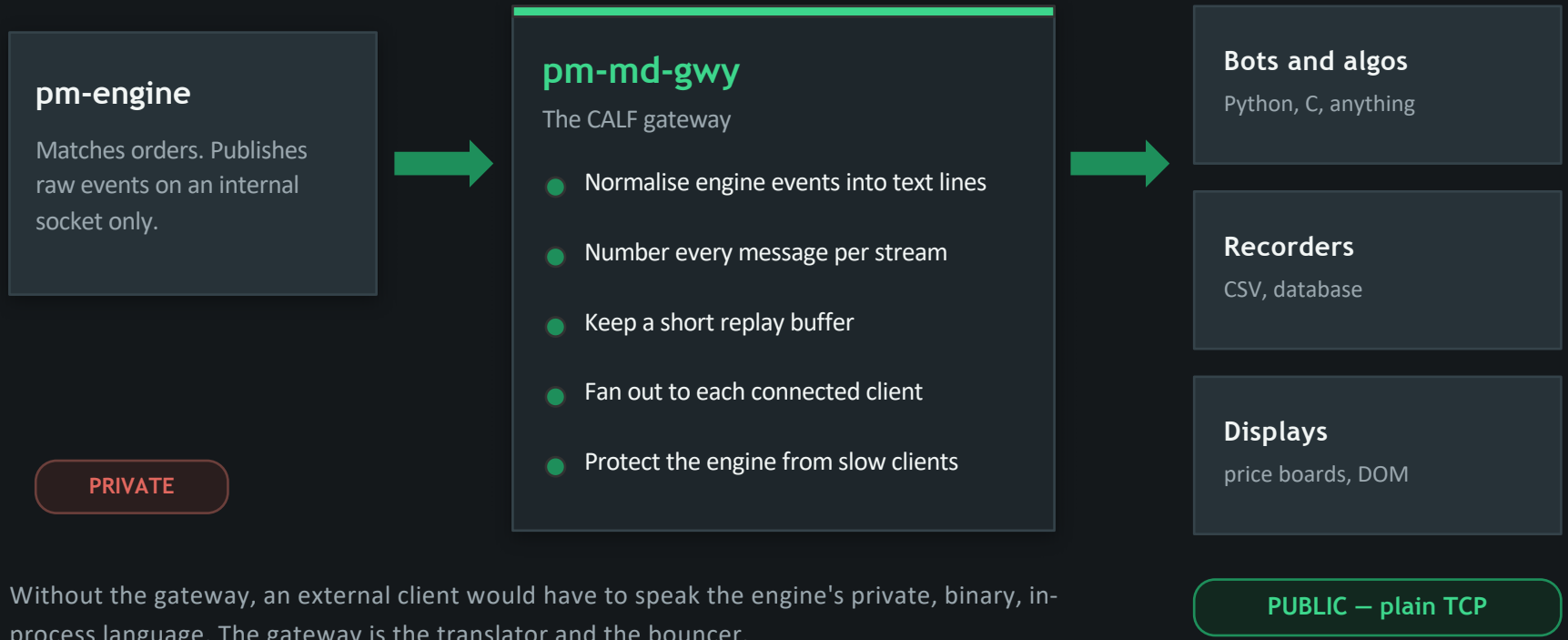
Every one of those events is a fact somebody outside the engine wants: a price board, a trading bot, a recorder, a classroom display.

```
AAPL
  bid  150.10 x 1200
  ask  150.12 x  900
  last 150.11 x  300
```

Market data is simply that stream of facts, delivered to people outside the engine.

So what is a gateway for?

It stands between a private engine and the public world



Without the gateway, an external client would have to speak the engine's private, binary, in-process language. The gateway is the translator and the bouncer.

CALF is one of four protocols

EduMatcher separates connectivity by purpose — CALF is the market data one

ALF	Order entry, human readable	`pm-alf-console` · `pm-alf-gwy`
BALF	Order entry, binary and fast	`pm-balf-gwy`
CALF	Market data out to the world	`pm-md-gwy`
RALF	Post-trade, clearing and audit	`pm-ralf-gwy`

All four are line-based or framed protocols over TCP. Learning one makes the next three familiar.

The purpose of the protocol

Four design decisions explain almost everything about CALF

Text, not binary

MSGTYPE|KEY=VALUE and a newline. You can read the wire with your own eyes and debug it with nc.

no schema compiler

No dependencies

A socket and a newline split. No ZeroMQ binding, no code generation, no client library required.

any language

Numbered streams

Every message carries a sequence number for its channel and symbol, so a hole is impossible to miss.

provable completeness

Recoverable

A short server-side buffer lets a reconnecting client ask for exactly what it missed.

no full restart

The feed at a glance

Everything you need to open a connection, on one slide

Process `pm-md-gwy`	Transport `plain TCP`	Port `5570`
Protocol token `CALF1`	Encoding `UTF-8 text`	Line ending `\n`
Max line `4096 bytes`	Heartbeat `1 s when quiet`	Idle timeout `5 s`
Replay window `30 s`	Channels `7`	Timestamps `ISO-8601 UTC`

Defaults shown. Every one of them is configurable — see the reference at the end.

PART 2

Reading the wire

One line format, one envelope, seven channels

Anatomy of a CALF line

Every message on the wire looks exactly like this

```
MD|CH=TOP|SYM=AAPL|SEQ=2|TS=2026-06-30T09:30:00.500Z|BID=150.11|BIDSZ=1400
```

MSGTYPE

What kind of line this is — MD, TRADE, SNAP, ERR …

first, before the first pipe

KEY=VALUE

Every field is named. Order is not significant.

separated by |

Newline

One message ends where the line ends.

\n terminates the frame

The one rule beginners always break

TCP is a byte stream, not a message stream. One recv() is not one line — it can be half a line, or three. Always buffer what arrives and split on newlines.

The envelope on every data line

Four fields come first, always, on every market-data message

CH	**Channel**	Which of the seven feeds this line belongs to
SYM	**Symbol**	The instrument, the index id, or * for market-wide
SEQ	**Sequence**	Counts 1, 2, 3 … per channel and symbol. Your gap detector
TS	**Timestamp**	Event time, ISO-8601 UTC with milliseconds

A stream is a pair

CALF does not have one sequence counter. It has one per (CH, SYM) pair — each is an independent stream.

```
(TOP, AAPL)   seq 1,2,3…
(TOP, MSFT)   seq 1,2,3…
(TRADE, AAPL) seq 1,2,3…
```

Track them separately and recovery becomes simple.

What data is available: seven channels

Each channel is one view of market activity — subscribe to only what you need

TOP

Best bid, ask and last

msg: MD

TRADE

Every executed print

msg: TRADE

DEPTH

Top-N order book ladder

msg: DEPTH

STATE

Session phase and halts

msg: STATE

INDEX

Index level updates

msg: IDX

AUCTION

Uncross results

msg: AUCTION

CB

Circuit-breaker detail

msg: CB

Pick by what you are building

A price board needs TOP. A tape needs TRADE. A ladder needs DEPTH.

TOP — the price everyone quotes

Best bid, best ask, last trade — the smallest useful view of a market

CH=TOP

A snapshot gives you the full picture. After that each MD line carries only the fields that changed — merge it into what you already hold.

Snapshot on subscribe	**Yes**
SYM=* allowed	**Yes**
Payload	**BID BIDSZ ASK ASKSZ LAST LASTSZ**

Use it for: price widgets, algos, watchlists

On the wire

```
SNAP|CH=TOP|SYM=AAPL|SEQ=1|BID=150.10|BIDSZ=1200
     |ASK=150.12|ASKSZ=900|LAST=150.11|LASTSZ=300
MD  |CH=TOP|SYM=AAPL|SEQ=2|BID=150.11|BIDSZ=1400
MD  |CH=TOP|SYM=AAPL|SEQ=3|ASK=150.13|ASKSZ=700
```

What your client displays

BID	ASK	LAST
150.11	150.13	150.12

merged from SNAP + 2 MD lines

TRADE — the tape

One line per executed trade. No snapshot: the tape starts when you do

CH=TRADE

Price, quantity and which side was the aggressor. Because a past print has no current state, there is nothing to snapshot — you receive trades from the moment you subscribe.

Snapshot on subscribe	No
SYM=* allowed	Yes
Payload	PX QTY SIDE

Use it for: time-and-sales, VWAP, OHLCV

On the wire

```
TRADE|CH=TRADE|SYM=AAPL|SEQ=44
     |TS=2026-06-30T09:30:01.100Z
     |PX=150.12|QTY=200|SIDE=BUY
```

Time and sales

09:30:01	150.12	200	BUY
09:30:03	150.11	500	SELL
09:30:04	150.12	100	BUY

DEPTH — the order book ladder

Level 2: the top N price levels per side, aggregated by price

CH=DEPTH

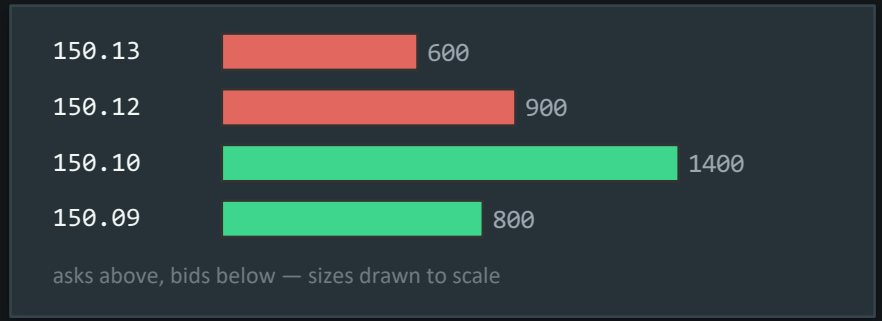
Each line carries the complete current ladder for both sides that hold liquidity — not a diff. Replace your ladder on every line, never merge it.

Snapshot on subscribe	Yes
SYM=* allowed	No
Payload	LEVELS BIDS ASKS

Use it for: DOM widgets, liquidity analysis

On the wire

```
DEPTH|CH=DEPTH|SYM=AAPL|SEQ=2|LEVELS=10
|BIDS=150.10:1400:4,150.09:800:2
|ASKS=150.12:900:2,150.13:600:1
each level is PRICE:QTY:ORDERCOUNT
```



STATE — is the market even open?

Session phases market-wide, plus halt and resume per symbol

CH=STATE

With SYM=* you follow the whole session's phase transitions. With a single symbol you learn when that instrument is halted and when it comes back.

Snapshot on subscribe	Yes
SYM=* allowed	Yes
Payload	SESSION PREV

Use it for: gating order flow, phase displays

On the wire

```
SNAP  |CH=STATE|SYM=*|SEQ=1|SESSION=PRE_OPEN
STATE|CH=STATE|SYM=*|SEQ=2|SESSION=OPENING_AUCTION|PREV=PRE_OPEN
STATE|CH=STATE|SYM=AAPL|SEQ=1|SESSION=HALTED|PREV=CONTINUOUS
```

A trading day



A halt interrupts one symbol; the session phase carries on.

INDEX — the market in one number

One IDX line every time the composite index is recalculated

CH=INDEX

Subscribe with an index id rather than a ticker, for example SYM=EDU50. A wildcard is never valid here — an explicit index id is always required.

Snapshot on subscribe	Yes
SYM=* allowed	No
Payload	LEVEL SESSION OPEN CHG PCTCHG HIGH LOW AGGCAP

Use it for: benchmarks, index dashboards

On the wire

```
IDX|CH=INDEX|SYM=EDU50|SEQ=12|LEVEL=5123.45
   |SESSION=CONTINUOUS|OPEN=5100.00|CHG=+23.45
   |PCTCHG=+0.46|HIGH=5130.10|LOW=5098.20
```

EDU50

5 123.45

+23.45 +0.46%

HIGH 5130.10

LOW 5098.20

OPEN 5100.00

AUCTION — how the open and close print

One line each time a symbol's auction uncrosses

CH=AUCTION

The equilibrium price, the matched quantity, how many trades it generated and any imbalance left behind. Like TRADE, there is no snapshot.

Snapshot on subscribe	No
SYM=* allowed	Yes
Payload	EQPX EQQTY TRADES IMBSIDE IMBQTY

Use it for: auction monitors, imbalance panels

On the wire

```
AUCTION|CH=AUCTION|SYM=AAPL|SEQ=1
  |EQPX=150.10|EQQTY=48200|TRADES=37
  |IMBSIDE=BUY|IMBQTY=1400
no cross? EQPX and IMBSIDE are omitted, not zero
```

EQUILIBRIUM	MATCHED	IMBALANCE
150.10	48 200	1 400 BUY

A halt always ends in a re-opening auction, published here with REASON=REOPEN.

CB — the detail behind a halt

Why a symbol stopped, at what price, and when it plans to come back

CH=CB

STATE tells you a symbol is halted. CB tells you the breaker level, the trigger and reference price, the scheduled resume time and whether the corridor was widened.

Snapshot on subscribe	Yes
SYM=* allowed	No
Payload	STATUS LEVEL TRIGGERPX REFPX RESUMEAT SRC …

Use it for: halt panels, resume countdowns

On the wire

```
CB|CH=CB|SYM=AAPL|SEQ=2|STATUS=HALTED|LEVEL=L2
  |TRIGGERPX=148.20|REFPX=150.10
  |RESUMEAT=2026-07-20T10:20:00.000Z|SRC=CB
CB|CH=CB|SYM=AAPL|SEQ=3|STATUS=ACTIVE|SRC=CB
```

The trap: a halt can be extended

If the indicative price sits outside the corridor when RESUMEAT arrives, the corridor widens (EXP counts up), a fresh call phase starts and RESUMEAT moves. A client that ignores EXP, CORRLO and CORRHI will report the symbol as overdue to reopen.

Two rules that catch everyone out

Which channels give you a baseline, and where the wildcard works

Do I get a baseline snapshot?

SNAP arrives the moment you subscribe — it is the SEQ you anchor on.

TOP	SNAP
STATE	SNAP
INDEX	SNAP
DEPTH	SNAP
CB	SNAP
TRADE	none
AUCTION	none

Where can I use SYM=*?

One subscription for the whole market — but only on four channels, and never on RESUME.

TOP	SYM=*
TRADE	SYM=*
STATE	SYM=*
AUCTION	SYM=*
INDEX	explicit
DEPTH	explicit
CB	explicit

A wildcard TOP subscription is the one exception to one-snapshot-per-stream: it sends one real SNAP per known symbol, then live MD — including for symbols added later.



PART 3

Using the protocol

A session, start to finish

The shape of every session

Six steps, always in this order

1	**Connect**	Open a TCP socket to port 5570
2	**HELLO**	Identify yourself and name the protocol
3	**WELCOME**	The gateway tells you what it supports
4	**SUB**	Ask for channels × symbols
5	**SNAP + stream**	Baseline, then live updates and heartbeats
6	**EXIT**	Close politely

No client library needed to try it — nc or telnet is enough.

As typed by hand

```
$ nc 127.0.0.1 5570
HELLO|CLIENT=demo01|PROTO=CALF1
< WELCOME|PROTO=CALF1|GW=md-gwy01
  |HBINT=1|REPLAY=30
SUB|CH=TOP,TRADE|SYM=AAPL
< SNAP|CH=TOP|SYM=AAPL|SEQ=1…
< MD|CH=TOP|SYM=AAPL|SEQ=2…
< TRADE|CH=TRADE|SYM=AAPL…
< HB|TS=…
EXIT
```

Step 1 — the handshake

HELLO says who you are; WELCOME tells you what this gateway can do

You send

```
HELLO|CLIENT=mybot|PROTO=CALF1
```

CLIENT is free text, up to 32 characters, used in gateway logs. PROTO must be exactly CALF1.

Read WELCOME before you subscribe

Two fields decide how your client behaves. Skipping them is the most common beginner mistake — you end up handling errors reactively instead of degrading gracefully.

WELCOME carries

GW	which gateway instance you reached
HBINT	heartbeat interval in seconds
REPLAY	how many seconds of replay are buffered
SYMBOLS	known instruments — optional, may be absent
REF	tick decimals per symbol, for display precision
CH_SUPPORTED	the channels this build actually supports

If CH_SUPPORTED is absent, assume an older gateway: TOP, TRADE and STATE only.

Step 2 — subscribe

One line asks for every channel crossed with every symbol

```
SUB|CH=TOP,TRADE|SYM=AAPL,MSFT
```

Two channels × two symbols = four streams, from one line.

`TOP · AAPL`	`TOP · MSFT`
`TRADE · AAPL`	`TRADE · MSFT`

Subscriptions are cumulative — a second SUB line adds to what you already have, it never replaces it. UNSUB removes pairs and is idempotent.

Subscribe only to what you need — it costs the gateway fanout and costs you parsing.

Common subscriptions

Best bid/ask, one symbol

```
SUB|CH=TOP|SYM=AAPL
```

Market-wide trade tape

```
SUB|CH=TRADE|SYM=*
```

Everything for one symbol

```
SUB|CH=TOP,TRADE,STATE|SYM=AAPL
```

Session phases only

```
SUB|CH=STATE|SYM=*
```

Order book ladder

```
SUB|CH=DEPTH|SYM=AAPL
```

Index level

```
SUB|CH=INDEX|SYM=EDU50
```

Step 3 — seed, then merge

The single most important habit in a market data client

SNAP

Everything currently known

```
BID 150.10  BIDSZ 1200
ASK 150.12  ASKSZ 900
```

MD

Only what changed

```
BID 150.11  BIDSZ 1400
```

Your state

Snapshot with the change folded in

```
BID 150.11  BIDSZ 1400
ASK 150.12  ASKSZ 900
```

Two channels break the pattern

DEPTH replaces rather than merges — every line carries the whole ladder. TRADE and AUCTION have no state to seed at all: each line is a standalone event.

Step 4 — staying connected

Heartbeats, timeouts and the polite way out

HB

Heartbeat

Sent about once a second whenever your streams are quiet. It keeps the session alive on its own — a pure listener never has to send anything.

PING / PONG

Liveness probe

You may send PING at any time; the gateway answers PONG. Useful when you want an explicit round-trip check.

Idle timeout

5 seconds

The connection closes only when nothing has flowed in either direction. In practice that means writes to your client stopped succeeding.

EXIT

Clean shutdown

One word closes the session. The gateway will also close it for you if you cannot keep up.

PART 4

When things go wrong

Errors, gaps and replay

How errors are handled

One message type, one code, and the session usually survives

```
ERR|CODE=INVALID_SYMBOL|MSG=unknown symbol ZZZZ
```

You made a mistake

AUTH_REQUIRED

PROTO_MISMATCH

INVALID_CHANNEL

INVALID_SYMBOL

BAD_MESSAGE

Fix the line and send it again. The session stays open.

You asked for too much

SUB_LIMIT

RATE_LIMITED

Back off: fewer symbols, or a slower send rate. The session stays open.

The data is gone

REPLAY_MISS

SLOW_CLIENT

Recover: accept a fresh snapshot, or reconnect and consume faster.

Rule of thumb: an ERR is a reply to something you sent, not the end of the session. Read the code, correct the request, carry on.

The failures that arrive silently

Two cases where you get no ERR at all — plan for them

Slow client

You cannot drain the stream fast enough and the outbound queue overflows.

What you see

The gateway drops the queue and closes the socket. ERR|CODE=SLOW_CLIENT may never reach you.

What to do

Treat any unexpected disconnect on a busy stream as this. Reconnect, then consume faster or subscribe to less.

Gateway at capacity

The connection limit is already reached.

What you see

The TCP socket is accepted and closed immediately — no HELLO, no WELCOME, no ERR line.

What to do

Treat an instant disconnect before WELCOME as a possible connection cap, not a protocol bug.

Spotting a gap

The sequence number does all the work

```
gap detected when   received_seq  !=  last_seq + 1
```



Two messages on (TOP, AAPL) never arrived — 102 should have been 100.

- **A SNAP is never a gap** — It re-baselines the stream — just record its SEQ.
- **Sequence never moves backward** — Anything at or below what you recorded is a replayed duplicate. Drop it.
- **Track per (CH, SYM)** — A gap on one stream says nothing about the others.

Keep one small map

```
last_seq = {}

key = (ch, sym)
prev = last_seq.get(key)

if type == 'SNAP':
    pass          # baseline
elif prev and seq <= prev:
    return        # duplicate
elif prev and seq != prev+1:
    recover(key, prev)

last_seq[key] = seq
```

Replaying what you missed

RESUME asks the gateway for everything after a sequence you already hold

```
HELLO|CLIENT=mybot|PROTO=CALF1
RESUME|CH=TOP|SYM=AAPL|LASTSEQ=99
RESUME|CH=TRADE|SYM=AAPL|LASTSEQ=41
```

One RESUME per stream

LASTSEQ describes a single stream's position — send as many lines as you have streams.

SYM=* is never valid here

Even on channels where the wildcard works for SUB. RESUME always replays one concrete symbol.

Expect duplicates

Replay overlaps live traffic. Drop anything at or below the SEQ you already recorded.

The replay window

The gateway keeps roughly the last 30 seconds of every stream — WELCOME|REPLAY= tells you exactly how much.



Inside the window you get the missing lines and then live traffic, seamlessly. Outside it, you get told — and on most channels a fresh snapshot follows.

Need a guaranteed baseline after reconnecting? Re-SUB the stream to force a SNAP rather than relying on RESUME alone.

Three things RESUME can return

Handle all three or your client will misreport its own history

1

Inside the window

The missing messages arrive, then live traffic continues.

Drop duplicates, carry on.

2

Too old — REPLAY_MISS

On TOP, STATE, INDEX, DEPTH and CB a fresh SNAP follows.

Accept it and reset that stream's state.

3

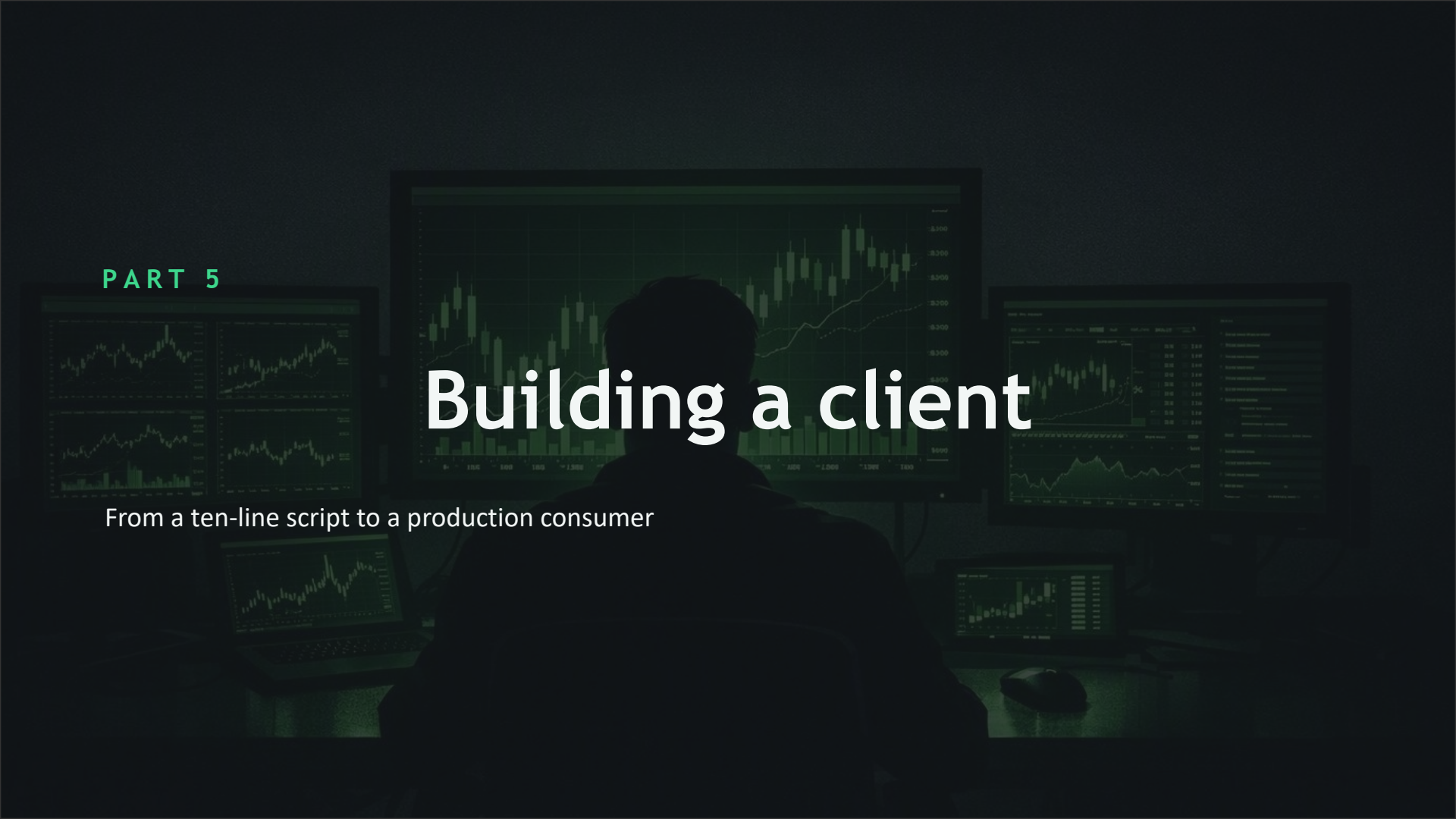
Nothing recorded

Zero replay lines. No error, no snapshot — the stream simply resumes live.

Not a failure. Re-SUB if you need a baseline.

TRADE and AUCTION are different — and it matters

A past print has no current state to snapshot, so after REPLAY_MISS nothing follows and those events are gone for good. If you present a tape, mark the hole — a record with an unmarked gap is worse than one that admits it.



PART 5

Building a client

From a ten-line script to a production consumer

Anatomy of a robust client

Six habits keep any CALF client correct, in any language

1 Frame properly

Buffer bytes, split on newlines. One recv() is not one message.

2 Handshake first

HELLO, wait for WELCOME, read SYMBOLS and CH_SUPPORTED before subscribing.

3 Seed then update

Initialise from SNAP, merge each MD, replace the ladder on each DEPTH.

4 Watch SEQ

Track per (CH, SYM). On a gap, RESUME — or re-SUB for a fresh snapshot.

5 Stay alive

Heartbeats need no answer. Treat a surprise disconnect as a slow-client warning.

6 Degrade gracefully

If CH_SUPPORTED lacks a channel, fall back instead of subscribing blindly.

A client in fifteen lines

No library, no dependencies — this is the whole point of CALF

```
import socket

sock = socket.create_connection(("127.0.0.1", 5570))
sock.sendall(b"HELLO|CLIENT=bot01|PROTO=CALF1\n")
sock.sendall(b"SUB|CH=TOP,TRADE|SYM=AAPL\n")

buf = bytearray()
while True:
    chunk = sock.recv(4096)
    if not chunk:
        break
    buf.extend(chunk)
    while b"\n" in buf:
        idx = buf.index(b"\n")
        line = buf[:idx].decode("utf-8").strip()
        del buf[:idx + 1]
        if line:
            print(line)
```

Why this works anywhere

Every language has a TCP socket and a string split. That is the entire dependency list for a CALF consumer.

What it does not do

No gap detection, no replay, no reconnect, no price precision. Fine for a smoke test — not for anything you rely on.

Pick the right starting point

Four levels, from “show me the wire” to “ship it”

`pm-calf-spy`	CLI	Watch the raw feed, human-readable or JSON. No code at all.	Exploring and debugging
`calf_parser`	Python · C	Just the line parser and serializer. No channel semantics.	Bootstrapping a new language
`calf_subscriber`	Python · C	A full worked example: subscribe, detect gaps, repair with RESUME.	Learning the protocol properly
`edumatcher.calf_client`	Python	Production client: reconnect with backoff, gap repair, price precision, cached market state.	Building something real

Writing a real Python client? Start from the library, not the example — it is deliberately standalone so it ports cleanly to new languages.

What you can build on it

Each of these is well under a hundred lines

Time and sales tape
`TRADE`

Live price board
`TOP`

DOM ladder widget
`DEPTH`

VWAP / OHLCV recorder
`TRADE`

Halt-aware alert bot
`STATE`

Index dashboard
`INDEX`

Auction monitor
`AUCTION`

Circuit-breaker panel
`CB`

CSV / Parquet recorder
`TRADE + TOP`

Browsers cannot open a raw TCP socket — use the WebSocket feed on the API gateway, or write a small server-side bridge that relays each CALF line onward.

PART 6

Technical reference

The detail you will want open on a second screen

Configuration reference

The market_data_gateway block — every key is optional, defaults shown

enabled	true	When false the gateway logs a warning and exits without binding
name	md-gwy01	Gateway id reported in WELCOME|GW=
bind_address	0.0.0.0	TCP listen address
port	5570	TCP listen port
heartbeat_interval_sec	1	HB cadence when idle; advertised as HBINT
idle_timeout_sec	5	Disconnect after this long with no traffic either way
replay_window_sec	30	Per-stream replay retention; advertised as REPLAY
max_symbols_per_client	200	Unique symbols across all channels; over it returns SUB_LIMIT
max_client_queue	10000	Outbound backlog; over it disconnects silently
max_connections	64	Simultaneous connections; over it the socket closes immediately
max_messages_per_second	200	Inbound rate limit; over it returns RATE_LIMITED
depth_levels	10	Price levels per side on DEPTH; surfaced as LEVELS=

All values must be positive integers, or the gateway refuses to start. An unreadable engine config disables known-symbol validation gateway-wide.

Channel matrix

Message type, snapshot behaviour, wildcard support and payload

Channel	Message	SNAP	SYM=*	Payload fields
TOP	MD	Yes	Yes	BID BIDSZ ASK ASKSZ LAST LASTSZ
TRADE	TRADE	No	Yes	PX QTY SIDE
STATE	STATE	Yes	Yes	SESSION PREV
INDEX	IDX	Yes	No	LEVEL SESSION OPEN CHG PCTCHG HIGH LOW AGGCAP
DEPTH	DEPTH	Yes	No	LEVELS BIDS ASKS
AUCTION	AUCTION	No	Yes	EQPX EQQTY TRADES IMBSIDE IMBQTY
CB	CB	Yes	No	STATUS LEVEL TRIGGERPX REFPX RESUMEAT SRC CORRLO CORRHI EXP

Every line also carries the CH, SYM, SEQ and TS envelope before these payload fields.

Error codes in full

Cause and action for every code the gateway emits

`AUTH_REQUIRED`	SUB sent before HELLO	Send HELLO first
`PROTO_MISMATCH`	Wrong or missing PROTO	Use PROTO=CALF1
`INVALID_CHANNEL`	Unknown CH value	Use one of the seven channel names
`INVALID_SYMBOL`	Unknown symbol, or SYM=* where it is not allowed	Wildcard only on SUB for STATE/TOP/TRADE/AUCTION — never on RESUME
`SUB_LIMIT`	Too many unique symbols subscribed	Reduce the requested symbol set
`RATE_LIMITED`	Inbound rate exceeded	Slow down; the connection stays open
`REPLAY_MISS`	Requested replay is outside the window	Accept the fresh SNAP and reset — except on TRADE/AUCTION, where none follows
`SLOW_CLIENT`	Outbound queue could not be drained	Reconnect and consume faster; this ERR may never arrive
`BAD_MESSAGE`	Malformed or oversized line, over 4096 bytes	Fix the line syntax or framing

An immediate disconnect before any WELCOME is not in this table — it usually means the gateway is at max_connections.

Starting the gateway

Prerequisites, commands and the overrides worth knowing

Prerequisites

- pm-engine running and publishing
- pm-md-gwy running
- symbols configured in the engine config

Run it

```
pm-engine --verbose
pm-md-gwy

# developer mode
poetry run pm-md-gwy
```

CLI overrides

`--bind ADDR`	TCP bind address
`--port PORT`	TCP listen port
`--engine-pub ADDR`	engine PUB socket, default 127.0.0.1:5556
`--index-pub ADDR`	index PUB socket, default 127.0.0.1:5558
`--log-level LEVEL`	CRITICAL … DEBUG
`-v / -vv / -q`	verbosity shortcuts
`--version`	print the installed version and exit

The gateway must read the same engine configuration as the engine itself.

Operational checklist

Work down this list when a feed is not behaving

- [ ] pm-engine is running and publishing
- [ ] pm-md-gwy is running
- [ ] the TCP port is reachable — nc 127.0.0.1 5570
- [ ] HELLO receives a WELCOME
- [ ] SUB receives the expected SNAP and live flow
- [ ] SEQ is tracked per stream, and one RESUME is sent per stream on reconnect
- [ ] on REPLAY_MISS the recovery SNAP is accepted and local state reset
- [ ] replayed messages at or below the recorded SEQ are dropped

An immediate disconnect before WELCOME, or a busy stream that drops without an error, both point at limits rather than syntax.

IN SHORT

Four things to remember

1 A gateway translates

The engine speaks privately; CALF speaks to everyone, in plain text over TCP.

2 Seven channels, one envelope

CH, SYM, SEQ and TS on every line — learn one channel and you know them all.

3 Seed, then merge

A snapshot gives you the baseline; increments only carry what changed.

4 Never ignore SEQ

It is the difference between a feed you can trust and one you merely hope is complete.

Next: open a socket, send HELLO, and watch the market arrive one line at a time.