NYSE +0.31% · NASDAQ +0.47% · LSE -0.12% · Eurex +0.18% · JPX +0.22% · HKEX -0.08% · CME +0.55% · SGX +0.11%



EQUITY TRADING INFRASTRUCTURE

PROTOCOLS AND CLIENTS

A Guide to the Protocols of EduMatcher and How to Build External TCP Clients of EduMatcher

MARKET DEPTH · AAPL +1.24 +0.83%

BID QTY		PRICE	ASK QTY
8,340		150.35	3,920
7,210		150.34	5,100
5,820		150.33	6,400
12,100	LAST	150.32	7,840
4,420		150.31	10,220
3,200		150.30	12,800

EduMatcher Protocols and Clients
For EduMatcher Version: v0.77.0
Copyright © Johan Persson
Generated on November 8, 2026
Prepared with LaTeX from Markdown sources for duplex printing. Sources for documentation available at https://github.com/johan162/EduMatcher.

Protocols and Clients

`edumatcher`, Version: v0.77.0

External protocols, session behavior, and client examples

Johan Persson <johan162@gmail.com>

Generated on November 8, 2026

Chapters

1 How to Use This Book 1

2 Protocols Overview 3

3 ALF Protocol Reference 9

4 BALF Protocol Reference 33

5 CALF Protocol Reference 45

6 RALF Protocol Reference 73

7 LALF Protocol Reference 81

8 REST and WebSocket API Reference 93

9 WebSocket API Reference 163

10 Market Data Feed (CALF) 173

11 Drop Copy 181

12 Protocol Support Library Examples 189

13 Drop-Copy Spy (`pm-dc-spy`) 209

14 CALF Protocol Spy (`pm-calf-spy`) 215

15 RALF Protocol Spy (`pm-ralf-spy`) 221

16 Message Reference 227

Contents

1 How to Use This Book **1**
- 1.1 How the book is organized . . . 1
- 1.2 Where to start . . . 2

2 Protocols Overview **3**
- 2.1 Why this page exists . . . 3
- 2.2 Protocol map . . . 3
- 2.3 ALF (Almost FIX) . . . 4
- 2.4 BALF (Binary ALF) . . . 5
- 2.5 CALF (Channel ALF) . . . 5
- 2.6 RALF (Reconciliation ALF) . . . 6
- 2.7 LALF (Log ALF) . . . 6
- 2.8 REST/WebSocket (`pm-api-gwy`) . . . 6
- 2.9 Drop copy and DC1 (`pm-dc-gwy`) . . . 7
- 2.10 Internal ZeroMQ message bus . . . 7
- 2.11 Quick selection guide . . . 7
- 2.12 See also . . . 7

3 ALF Protocol Reference **9**
- 3.1 What ALF is . . . 9
- 3.2 Scope & conformance . . . 9
- 3.3 Wire format . . . 10
  - 3.3.1 Line structure . . . 10
  - 3.3.2 Formal grammar . . . 10
  - 3.3.3 Whitespace . . . 10
  - 3.3.4 Case handling . . . 11
  - 3.3.5 Field order . . . 11
  - 3.3.6 Duplicate and unknown fields . . . 11
  - 3.3.7 Numeric formatting . . . 12
  - 3.3.8 No escaping . . . 12
- 3.4 Common field vocabulary . . . 12
  - 3.4.1 Supported `TIF` values . . . 13
  - 3.4.2 Supported `SMP` values . . . 13
- 3.5 Session establishment — `HELLO` (`pm-alf-gwy` only) . . . 13
  - 3.5.1 Formal syntax . . . 13
  - 3.5.2 Successful handshake . . . 13
  - 3.5.3 Pre-authentication error codes . . . 14
  - 3.5.4 `PING` / `PONG` . . . 14

3.6 Command families . . . 14
3.7 Single-leg NEW orders . . . 16
3.7.1 Generic form . . . 16
3.7.2 Formal grammar . . . 16
3.7.3 Supported single-leg order types . . . 16
3.7.4 Formal syntax by order type . . . 17
3.7.5 MARKET . . . 17
3.7.6 LIMIT . . . 17
3.7.7 IOC . . . 17
3.7.8 FOK . . . 17
3.7.9 STOP . . . 17
3.7.10 STOP_LIMIT . . . 17
3.7.11 ICEBERG . . . 18
3.7.12 TRAILING_STOP . . . 18
3.7.13 Auction-only TIF values . . . 18
3.7.14 Practical parsing notes . . . 18
3.8 AMEND - order amendments . . . 18
3.8.1 Formal syntax . . . 18
3.8.2 Supported amendment forms . . . 19
3.8.3 Examples . . . 19
3.8.4 Amendment semantics . . . 19
3.8.5 Priority rules . . . 19
3.8.6 Important identifier rule . . . 20
3.9 CANCEL - order and group cancellation . . . 20
3.9.1 Single order cancel . . . 20
3.9.2 Combo cancel . . . 20
3.9.3 OCO cancel . . . 20
3.9.4 Cancellation semantics . . . 20
3.10 NEW|TYPE=OCO - one-cancels-other pairs . . . 20
3.10.1 Formal syntax . . . 21
3.10.2 OCO field semantics . . . 21
3.10.3 OCO leg expressibility . . . 21
3.10.4 Examples . . . 22
3.10.5 OCO semantics . . . 22
3.11 NEW|TYPE=COMBO - multi-leg combo orders . . . 22
3.11.1 Formal syntax . . . 22
3.11.2 Combo field semantics . . . 22
3.11.3 Combo leg expressibility . . . 23
3.11.4 Examples . . . 23
3.11.5 Combo semantics . . . 23
3.12 QUOTE and QUOTE_CANCEL . . . 24
3.12.1 Formal syntax . . . 24
3.12.2 Example . . . 24
3.12.3 Quote semantics . . . 24
3.13 KILL . . . 24
3.13.1 Syntax . . . 24
3.13.2 Examples . . . 24
3.13.3 Semantics . . . 24
3.13.4 Acknowledgement behavior . . . 25
3.13.5 Examples . . . 25

3.14 DC — drop-copy relay control . . . 26
3.14.1 Syntax . . . 26
3.14.2 Semantics . . . 26
3.14.3 Acknowledgement behavior . . . 26
3.14.4 Examples . . . 26
3.15 Informational gateway commands . . . 27
3.15.1 STATUS . . . 27
3.15.2 ORDERS . . . 27
3.15.3 POS . . . 27
3.15.4 SYMBOLS . . . 27
3.15.5 SESSION . . . 27
3.15.6 QBOOT . . . 28
3.15.7 QLEGS . . . 28
3.15.8 HELP . . . 29
3.15.9 EXIT / QUIT . . . 29
3.16 Configuration reference . . . 29
3.17 What to watch out for during implementation . . . 30
3.18 Conformance notes . . . 31
3.19 See also . . . 31

4 BALF Protocol Reference 33
4.1 What BALF is . . . 33
4.2 Scope & conformance . . . 33
4.2.1 What BALF does not change . . . 34
4.2.2 Non-goals for BALF 1.0.0 . . . 34
4.3 Wire format . . . 34
4.3.1 Connection model . . . 34
4.3.2 Byte order . . . 34
4.3.3 Frame structure . . . 34
4.3.4 Flag semantics . . . 34
4.3.5 Identifier rules . . . 35
4.4 Frame format . . . 35
4.4.1 Header layout . . . 35
4.4.2 Header fields . . . 35
4.4.3 Sequence numbering rules . . . 35
4.5 Scalar encodings . . . 36
4.5.1 Price encoding . . . 36
4.5.2 Symbol encoding . . . 36
4.5.3 Gateway ID encoding . . . 36
4.5.4 Order ID encoding . . . 36
4.6 Message definitions . . . 36
4.6.1 Summary table . . . 36
4.6.2 LOGON (0x01) - Client -> Server . . . 37
4.6.3 LOGON_ACK (0x02) - Server -> Client . . . 37
4.6.4 NEW_ORDER (0x10) - Client -> Server . . . 38
4.6.5 ORDER_ACK (0x11) - Server -> Client . . . 39
4.6.6 CANCEL_ORDER (0x12) - Client -> Server . . . 40
4.6.7 CANCEL_ACK (0x13) - Server -> Client . . . 40
4.6.8 AMEND_ORDER (0x14) - Client -> Server . . . 40
4.6.9 AMEND_ACK (0x15) - Server -> Client . . . 40

4.6.10 `EXECUTION_REPORT` (0x20) - Server -> Client . . . 40
4.6.11 `STOP_TRIGGERED` (0x21) - Server -> Client . . . 41
4.6.12 `HEARTBEAT` (0x30) - Bidirectional . . . 41
4.6.13 `HEARTBEAT_ACK` (0x31) - Bidirectional . . . 41
4.6.14 `LOGOUT` (0x40) - Client -> Server . . . 41
4.7 Session lifecycle . . . 42
4.7.1 Sequence number rules . . . 42
4.8 Configuration reference . . . 43
4.8.1 `balf_gateway` fields . . . 43
4.9 What to watch out for during implementation . . . 43
4.10 Practical parsing notes . . . 44
4.11 See also . . . 44

5 CALF Protocol Reference 45
5.1 What CALF is . . . 45
5.2 Scope & conformance . . . 45
5.2.1 Supported in CALF `1.0.0` . . . 45
5.2.2 Out of scope in CALF `1.0.0` . . . 46
5.3 Transport and session model . . . 46
5.4 Wire format . . . 46
5.4.1 Line structure . . . 46
5.4.2 Parsing behavior . . . 47
5.4.3 TCP stream requirement . . . 47
5.5 Field conventions . . . 47
5.5.1 Reserved keys . . . 47
5.5.2 Wire value types . . . 47
5.6 Channel model . . . 48
5.6.1 Subscription rules . . . 48
5.7 Message catalog . . . 49
5.7.1 Session control messages . . . 49
5.7.2 Market-data messages . . . 49
5.8 Message definitions . . . 50
5.8.1 `HELLO` . . . 50
5.8.2 `WELCOME` . . . 50
5.8.3 `SUB` . . . 51
5.8.4 `RESUME` . . . 52
5.8.5 `SYMBOLS` . . . 52
5.8.6 `UNSUB` . . . 53
5.8.7 `SNAP` . . . 54
5.8.8 `MD` . . . 55
5.8.9 `TRADE` . . . 56
5.8.10 `STATE` . . . 57
5.8.11 `IDX` . . . 58
5.8.12 `DEPTH` . . . 59
5.8.13 `AUCTION` . . . 60
5.8.14 `INDIC` . . . 61
5.8.15 `CB` . . . 62
5.8.16 `HB` . . . 64
5.8.17 `PING` / `PONG` . . . 64
5.8.18 `ERR` . . . 64

5.8.19 EXIT . . . 65
5.9 Sequence and recovery semantics . . . 66
5.9.1 Stream identity . . . 66
5.9.2 Sequence rules . . . 66
5.9.3 First connect behavior . . . 66
5.9.4 Reconnect behavior (RESUME) . . . 66
5.10 Liveness and timeout rules . . . 68
5.11 Session lifecycle . . . 68
5.12 Gateway behavior requirements . . . 68
5.13 Configuration reference . . . 69
5.14 What to watch out for during implementation . . . 71
5.15 Conformance notes . . . 71
5.16 See also . . . 72

6 RALF Protocol Reference 73
6.1 1. What RALF is . . . 73
6.2 2. Scope & conformance . . . 73
6.3 3. Transport & session model . . . 73
6.4 4. Wire format . . . 74
6.5 5. Message catalog . . . 74
6.6 6. Message definitions . . . 75
6.6.1 HELLO (client → gateway) . . . 75
6.6.2 WELCOME (gateway → client) . . . 75
6.6.3 SUB / UNSUB (client → gateway) . . . 76
6.6.4 SNAP (gateway → client) . . . 76
6.6.5 EXEC (gateway → client) . . . 76
6.6.6 EOD (gateway → client) . . . 76
6.6.7 HB, PING/PONG, EXIT . . . 77
6.7 7. Roles & entitlements . . . 77
6.8 8. Sequencing & recovery . . . 77
6.9 9. Liveness & timeouts . . . 78
6.10 10. Error codes . . . 78
6.11 11. Configuration reference . . . 78
6.12 12. Conformance notes . . . 79
6.13 See also . . . 79

7 LALF Protocol Reference 81
7.1 What LALF is . . . 81
7.2 Scope & conformance . . . 82
7.2.1 Supported in LALF 1 . . . 82
7.2.2 Out of scope in LALF 1 . . . 82
7.3 Transport and session model . . . 82
7.4 Wire format . . . 83
7.4.1 Line structure . . . 83
7.4.2 Parsing behavior . . . 83
7.4.3 TCP stream requirement . . . 84
7.5 Message catalog . . . 84
7.6 Message definitions . . . 84
7.6.1 HELLO . . . 84
7.6.2 WELCOME . . . 85

7.6.3 LOG . . . 86
7.6.4 ACK . . . 87
7.6.5 ERR . . . 87
7.6.6 HB . . . 88
7.6.7 PING / PONG . . . 89
7.6.8 EXIT . . . 89
7.7 Backpressure . . . 89
7.8 Sequence semantics . . . 89
7.9 Configuration reference . . . 90
7.10 What to watch out for during implementation . . . 90
7.11 Conformance notes . . . 91
7.12 See also . . . 91

8 REST and WebSocket API Reference 93
8.1 Scope & conventions . . . 93
8.2 Endpoint index . . . 93
8.2.1 Bootstrap . . . 93
8.2.2 Trading REST . . . 93
8.2.3 Reference data . . . 95
8.2.4 Market data . . . 95
8.2.5 History . . . 95
8.2.6 Admin REST . . . 96
8.2.7 Auth and roles . . . 97
8.2.8 Key types and provisioning . . . 97
8.2.9 Common error codes . . . 98
8.2.10 Swagger/OpenAPI documentation . . . 101
8.2.11 Common reply shapes . . . 101
8.2.12 Response contract . . . 101
8.2.13 Canonical error envelope . . . 102
8.2.14 Pagination contract . . . 103
8.2.15 Write outcomes and safe retries . . . 103
8.2.16 Order correlation with client_tag . . . 104
8.2.17 Unsolicited cancels and cancel_reason . . . 104
8.2.18 Category examples . . . 104
8.2.19 Compatibility and deprecation policy . . . 107
8.3 Bootstrap . . . 108
8.3.1 GET /api/v1/bootstrap/trader . . . 108
8.3.2 GET /api/v1/bootstrap/mm . . . 110
8.3.3 GET /api/v1/bootstrap/admin . . . 111
8.4 Trading REST . . . 112
8.4.1 POST /api/v1/orders . . . 112
8.4.2 DELETE /api/v1/orders/{order_id} . . . 114
8.4.3 PATCH /api/v1/orders/{order_id} . . . 115
8.4.4 POST /api/v1/orders/{order_id}/replace . . . 116
8.4.5 GET /api/v1/orders . . . 117
8.4.6 GET /api/v1/orders/{order_id} . . . 117
8.4.7 POST /api/v1/oco . . . 118
8.4.8 DELETE /api/v1/oco/{oco_id} . . . 120
8.4.9 POST /api/v1/combos . . . 121
8.4.10 DELETE /api/v1/combos/{combo_id} . . . 122

8.4.11 POST /api/v1/quotes . . . 123
8.4.12 DELETE /api/v1/quotes/{symbol} . . . 125
8.4.13 POST /api/v1/mass-cancel . . . 126
8.4.14 GET /api/v1/symbols . . . 126
8.4.15 GET /api/v1/news . . . 127
8.4.16 GET /api/v1/companies/{symbol} . . . 128
8.4.17 GET /api/v1/session . . . 128
8.4.18 GET /api/v1/halts . . . 129
8.4.19 GET /api/v1/status . . . 129
8.4.20 GET /api/v1/healthz . . . 130
8.4.21 GET /api/v1/quotes . . . 130
8.4.22 GET /api/v1/quotes/legs . . . 131
8.4.23 GET /api/v1/positions . . . 131
8.5 Reference data . . . 132
8.5.1 GET /api/v1/reference . . . 132
8.5.2 GET /api/v1/reference/config-version . . . 133
8.5.3 GET /api/v1/reference/symbols . . . 133
8.5.4 GET /api/v1/reference/risk . . . 134
8.5.5 GET /api/v1/reference/indexes . . . 134
8.5.6 GET /api/v1/reference/schedule . . . 135
8.6 Market data . . . 135
8.6.1 GET /api/v1/book/{symbol} . . . 135
8.6.2 GET /api/v1/book . . . 136
8.7 History . . . 137
8.7.1 GET /api/v1/history/orders . . . 137
8.7.2 GET /api/v1/history/orders/{order_id} . . . 138
8.7.3 GET /api/v1/history/fills . . . 138
8.7.4 GET /api/v1/history/trades . . . 139
8.7.5 GET /api/v1/history/daily . . . 140
8.7.6 GET /api/v1/history/price-snapshots . . . 140
8.7.7 GET /api/v1/history/index-daily . . . 141
8.7.8 GET /api/v1/history/index-snapshots . . . 142
8.7.9 GET /api/v1/history/indexes . . . 143
8.7.10 GET /api/v1/history/index-events . . . 143
8.8 Admin REST . . . 144
8.8.1 POST /api/v1/admin/session/transition . . . 144
8.8.2 GET /api/v1/admin/sim . . . 145
8.8.3 GET /api/v1/admin/gateways . . . 145
8.8.4 POST /api/v1/admin/gateways/{gateway_id}/disconnect . . . 146
8.8.5 POST /api/v1/admin/symbols/{symbol}/halt . . . 147
8.8.6 POST /api/v1/admin/symbols/{symbol}/resume . . . 148
8.8.7 GET /api/v1/admin/risk/state . . . 149
8.8.8 GET /api/v1/admin/gateways/{gateway_id}/orders . . . 149
8.8.9 GET /api/v1/admin/symbols/{symbol}/orders . . . 150
8.8.10 GET /api/v1/admin/orders . . . 151
8.8.11 GET /api/v1/admin/orders/{order_id}/audit . . . 152
8.8.12 GET /api/v1/admin/gateways/{gateway_id}/positions . . . 152
8.8.13 POST /api/v1/admin/symbols/{symbol}/cancel-orders . . . 153
8.8.14 POST /api/v1/admin/gateways/{gateway_id}/kill-switch . . . 154
8.8.15 DELETE /api/v1/admin/gateways/{gateway_id}/quotes/{symbol} . . . 155

8.8.16 `POST /api/v1/admin/market/halt` . . . 156
8.8.17 `POST /api/v1/admin/market/resume` . . . 157
8.8.18 `GET /api/v1/admin/market/volume` . . . 157
8.8.19 `POST /api/v1/admin/symbols/{symbol}/reopen` . . . 158
8.8.20 `POST /api/v1/admin/market/kill-switch` . . . 159
8.8.21 `POST /api/v1/admin/indexes/{index_id}/rebalance` . . . 160
8.8.22 `POST /api/v1/admin/reference/reload` . . . 161
8.9 Operational notes . . . 162

9 WebSocket API Reference 163
9.1 Streams . . . 163
9.2 Connecting . . . 164
9.2.1 Close codes . . . 164
9.3 The event envelope . . . 164
9.3.1 Sequencing . . . 165
9.3.2 Event types . . . 165
9.4 Flow control . . . 166
9.5 `/api/v1/ws/private` . . . 166
9.5.1 Opening frames . . . 166
9.5.2 Private event catalogue . . . 166
9.5.3 Recovery . . . 167
9.6 `/api/v1/ws/market-data` . . . 167
9.6.1 Channels . . . 168
9.6.2 Control frames . . . 168
9.6.3 Server control frames . . . 169
9.6.4 Recovery . . . 169
9.7 `/api/v1/ws/admin/monitor` . . . 170
9.7.1 Opening frames . . . 170
9.7.2 Events . . . 170
9.7.3 Recovery . . . 170
9.8 `/api/v1/ws/admin/sim` . . . 170

10 Market Data Feed (CALF) 173
10.1 Why CALF Exists . . . 173
10.2 Information Levels by Channel . . . 173
10.3 Core Message Types . . . 174
10.4 How Participants Subscribe . . . 175
10.5 Staying Up To Date Reliably . . . 176
10.6 First Connect vs Reconnect . . . 176
10.6.1 First connect . . . 176
10.6.2 Reconnect . . . 176
10.7 Why This Matters For Different Participants . . . 179
10.8 CALF and Index Dissemination . . . 179
10.9 Minimal Client Checklist . . . 180
10.10Summary . . . 180

11 Drop Copy 181
11.1 What is drop copy? . . . 181
11.2 Architecture . . . 181
11.2.1 Socket addresses . . . 182
11.3 Message format . . . 182

11.3.1 Payload fields . . . 182
11.3.2 `order.fill` event . . . 183
11.4 Sequence numbers . . . 184
11.5 In-memory buffer . . . 184
11.6 Replay . . . 184
11.6.1 Replay request (programmatic) . . . 184
11.6.2 Replay topic design . . . 185
11.6.3 Replay limitations . . . 185
11.7 Topic subscription patterns . . . 185
11.8 Startup and shutdown . . . 185
11.9 Configuration reference . . . 185
11.10 Subscribing to the drop copy feed . . . 186
11.11 Alternative: relayed through your own ALF session . . . 186
11.12 Alternative: `pm-dc-gwy` for non-ZeroMQ external clients . . . 187
11.13 How real exchanges publish drop copy . . . 187
11.14 RALF DROP_COPY channel vs. this feed . . . 188
11.15 See also . . . 188

12 Protocol Support Library Examples 189
12.1 Scope . . . 189
12.2 ALF support library and examples . . . 189
12.2.1 Source files . . . 189
12.2.2 Wire format . . . 190
12.2.3 Python API — `alf_parser.py` . . . 190
12.2.4 Python usage pattern . . . 191
12.2.5 Run Python interactive client . . . 191
12.2.6 C API — `alf_parser.h` . . . 191
12.2.7 C usage pattern . . . 192
12.2.8 Build and run C client . . . 192
12.2.9 Use ALF as a foundation for your own program . . . 192
12.2.10 Manual gateway sanity check . . . 194
12.3 CALF support library and examples . . . 194
12.3.1 Source files . . . 195
12.3.2 Wire format . . . 195
12.3.3 Python API — `calf_parser.py` . . . 195
12.3.4 Python usage pattern . . . 195
12.3.5 Run Python example . . . 196
12.3.6 C API — `calf_parser.h` . . . 196
12.3.7 C usage pattern . . . 196
12.3.8 Build and run C example . . . 196
12.3.9 Manual gateway sanity check . . . 196
12.4 RALF support library and examples . . . 197
12.4.1 Source files . . . 197
12.4.2 Wire format . . . 197
12.4.3 Python API — `ralf_parser.py` . . . 197
12.4.4 Python usage pattern . . . 198
12.4.5 Run Python example . . . 198
12.4.6 C API — `ralf_parser.h` . . . 198
12.4.7 C usage pattern . . . 199
12.4.8 Build and run C example . . . 199

12.4.9 Manual gateway sanity check . . . 199
12.5 DC support library and examples . . . 199
12.5.1 Source files . . . 199
12.5.2 Wire format . . . 200
12.5.3 Python API — `dc_parser.py` . . . 200
12.5.4 Python usage pattern . . . 200
12.5.5 Run Python example . . . 200
12.5.6 C API — `dc_parser.h` . . . 200
12.5.7 C usage pattern . . . 200
12.5.8 Build and run C example . . . 201
12.5.9 Manual gateway sanity check . . . 201
12.6 BALF support library and examples . . . 201
12.6.1 Source files . . . 201
12.6.2 Binary frame structure . . . 201
12.6.3 Supported message types . . . 201
12.6.4 Python API — `balf_parser.py` . . . 202
12.6.5 Python usage pattern . . . 203
12.6.6 C reference decoder — `balf_parser.c` . . . 203
12.6.7 Run the gateway . . . 204
12.7 REST API support library and examples . . . 204
12.7.1 Source files . . . 204
12.7.2 Authentication . . . 205
12.7.3 Python API — `api_gateway_client.py` . . . 205
12.7.4 Python usage pattern . . . 206
12.7.5 Run Python examples . . . 207
12.7.6 C API — `api_gateway_client.h` . . . 207
12.7.7 C usage pattern . . . 207
12.7.8 Build and run C example . . . 208
12.7.9 Manual API sanity check . . . 208
12.8 Integration guidance . . . 208

13 Drop-Copy Spy (`pm-dc-spy`) 209
13.1 What this tool is . . . 209
13.2 Why not the example subscriber? . . . 210
13.3 Not a protocol client — a plain ZMQ SUB . . . 210
13.4 Starting point . . . 210
13.5 Human-readable output . . . 211
13.6 JSON output . . . 212
13.7 Observing replay . . . 212
13.8 Running several instances at once . . . 212
13.9 Connection and protocol errors . . . 212
13.10 See also . . . 213

14 CALF Protocol Spy (`pm-calf-spy`) 215
14.1 What this tool is . . . 215
14.2 Why not the example subscriber? . . . 216
14.3 Starting point . . . 216
14.4 Keeping the connection alive . . . 217
14.5 Human-readable output . . . 218
14.6 JSON output . . . 218

14.7 Running several instances at once . . . 219
14.8 Connection and protocol errors . . . 219
14.9 See also . . . 219

15 RALF Protocol Spy (`pm-ralf-spy`) 221
15.1 What this tool is . . . 221
15.2 Why not `ralf_subscriber.py`? . . . 222
15.3 RALF's role model, briefly . . . 222
15.4 Starting point . . . 222
15.5 Keeping the connection alive . . . 224
15.6 Human-readable output . . . 224
15.7 JSON output . . . 224
15.8 Running several instances at once . . . 225
15.9 Connection and protocol errors . . . 225
15.10 See also . . . 225

16 Message Reference 227
16.1 Background — Messages in a Bus System . . . 227
16.1.1 What is a message? . . . 227
16.1.2 How message formats are defined in real systems . . . 228
16.1.3 What ZeroMQ requires of a message . . . 228
16.1.4 ZeroMQ without a broker . . . 228
16.2 Message structure . . . 229
16.3 Transport channels . . . 230
16.3.1 `system.gateway_connect` . . . 231
16.3.2 `system.gateway_heartbeat` . . . 232
16.3.3 `order.new` . . . 233
16.3.4 `order.cancel` . . . 236
16.3.5 `order.amend` . . . 236
16.3.6 `quote.new` . . . 237
16.3.7 `quote.ack.{GW_ID}` . . . 238
16.3.8 `quote.status.{GW_ID}` . . . 238
16.3.9 `quote.cancel` . . . 238
16.4 CALF TCP protocol (`pm-md-gwy`) . . . 239
16.5 See also . . . 240
16.6 Topic index . . . 240
16.7 Family `admin` . . . 244
16.7.1 Record types . . . 244
16.7.2 `admin.action.{gateway_id}` . . . 247
16.8 Family `auction` . . . 249
16.8.1 `auction.indicative.{symbol}` . . . 249
16.8.2 `auction.result.{symbol}` . . . 250
16.9 Family `book` . . . 252
16.9.1 Record types . . . 252
16.9.2 `book.{symbol}` . . . 253
16.9.3 `book.snapshot_request` . . . 255
16.9.4 `depth.{symbol}` . . . 255
16.10 Family `circuit_breaker` . . . 256
16.10.1 `circuit_breaker.halt.{symbol}` . . . 256
16.10.2 `circuit_breaker.extend.{symbol}` . . . 259

16.10.3 circuit_breaker.resume.{symbol} . . . 260
16.11 Family drop_copy . . . 262
16.11.1 drop_copy.event.{gateway_id} . . . 262
16.11.2 drop_copy.replay.{recipient_id} . . . 265
16.12 Family health . . . 266
16.12.1 Record types . . . 266
16.12.2 health.beacon . . . 276
16.12.3 health.snapshot . . . 278
16.12.4 health.event . . . 279
16.13 Family index . . . 280
16.13.1 Record types . . . 280
16.13.2 index.update . . . 283
16.13.3 index.history_request . . . 284
16.13.4 index.history.{gateway_id} . . . 284
16.13.5 index.corp_action . . . 285
16.13.6 index.constituent_change . . . 285
16.13.7 index.rebalance . . . 286
16.13.8 index.corp_action_ack.{gateway_id} . . . 287
16.13.9 index.constituent_change_ack.{gateway_id} . . . 288
16.13.10 index.rebalance_ack.{gateway_id} . . . 289
16.13.11 index.error.{gateway_id} . . . 290
16.14 Family log . . . 290
16.14.1 Record types . . . 290
16.14.2 log.subscribe . . . 292
16.14.3 log.renew . . . 293
16.14.4 log.unsubscribe . . . 293
16.14.5 log.backfill_request . . . 294
16.14.6 log.status_request . . . 294
16.14.7 log.subscribe_ack.{sub_id} . . . 294
16.14.8 log.renew_ack.{sub_id} . . . 295
16.14.9 log.unsubscribe_ack.{sub_id} . . . 296
16.14.10 log.status.{sub_id} . . . 296
16.14.11 log.backfill.{sub_id} . . . 296
16.14.12 log.event.{sub_id} . . . 297
16.14.13 log.notify.{sub_id} . . . 298
16.14.14 log.lease_expired.{sub_id} . . . 298
16.14.15 log.error.{sub_id} . . . 299
16.14.16 log.server_state . . . 299
16.15 Family news . . . 300
16.15.1 news.event . . . 300
16.16 Family order . . . 301
16.16.1 Record types . . . 301
16.16.2 execution_report (no bus topic) . . . 311
16.16.3 stop_triggered (no bus topic) . . . 313
16.16.4 order.ack.{gateway_id} . . . 314
16.16.5 order.fill.{gateway_id} . . . 319
16.16.6 order.cancelled.{gateway_id} . . . 325
16.16.7 order.expired.{gateway_id} . . . 327
16.16.8 order.triggered.{gateway_id} . . . 328
16.16.9 engine.trace.{symbol} . . . 330

16.16.10 order.amended.{gateway_id} . . . 334
16.16.11 order.new . . . 337
16.16.12 order.cancel . . . 340
16.16.13 order.amend . . . 341
16.16.14 order.combo_cancel . . . 342
16.16.15 order.combo . . . 342
16.16.16 order.oco . . . 343
16.16.17 order.oco_cancel . . . 344
16.16.18 order.orders_request . . . 345
16.16.19 order.orders.{gateway_id} . . . 345
16.16.20 order.price_level_orders_request . . . 346
16.16.21 order.price_level_orders.{gateway_id} . . . 347
16.17 Family quote . . . 348
16.17.1 quote.new . . . 348
16.17.2 quote.cancel . . . 349
16.17.3 quote.ack.{gateway_id} . . . 350
16.17.4 quote.status.{gateway_id} . . . 351
16.17.5 quote.cancel_ack.{gateway_id} . . . 352
16.18 Family risk . . . 353
16.18.1 risk.kill_switch . . . 353
16.18.2 risk.kill_switch_ack.{gateway_id} . . . 354
16.18.3 risk.kill_switch_gateway . . . 355
16.18.4 risk.kill_switch_gateway_ack.{gateway_id} . . . 356
16.18.5 risk.kill_switch_global . . . 358
16.18.6 risk.kill_switch_global_ack.{gateway_id} . . . 359
16.18.7 risk.band.{symbol} . . . 360
16.18.8 risk.symbol_halt . . . 361
16.18.9 risk.symbol_halt_ack.{gateway_id} . . . 362
16.18.10 risk.symbol_resume . . . 363
16.18.11 risk.symbol_resume_ack.{gateway_id} . . . 364
16.18.12 risk.cancel_symbol . . . 364
16.18.13 risk.cancel_symbol_ack.{gateway_id} . . . 365
16.18.14 risk.force_uncross . . . 366
16.18.15 risk.force_uncross_ack.{gateway_id} . . . 367
16.18.16 risk.circuit_breaker_halt_all . . . 368
16.18.17 risk.circuit_breaker_halt_all_ack.{gateway_id} . . . 369
16.18.18 risk.circuit_breaker_resume_all . . . 370
16.18.19 risk.circuit_breaker_resume_all_ack.{gateway_id} . . . 370
16.19 Family session . . . 371
16.19.1 Record types . . . 371
16.19.2 session.state . . . 372
16.19.3 session.transition . . . 374
16.19.4 session.next . . . 374
16.19.5 session.transition_ack.{gateway_id} . . . 375
16.20 Family sim . . . 375
16.20.1 Record types . . . 375
16.20.2 sim.value . . . 377
16.20.3 sim.state . . . 377
16.20.4 sim.command . . . 378
16.20.5 sim.command_ack.{gateway_id} . . . 379

16.21 Family `structure` . . . 379
16.21.1 `combo.ack.{gateway_id}` . . . 379
16.21.2 `combo.status.{gateway_id}` . . . 380
16.21.3 `oco.ack.{gateway_id}` . . . 380
16.21.4 `oco.cancelled.{gateway_id}` . . . 381
16.21.5 `oco.cancel_ack.{gateway_id}` . . . 381
16.21.6 `combo.cancel_ack.{gateway_id}` . . . 383
16.22 Family `system` . . . 384
16.22.1 Record types . . . 384
16.22.2 `system.gateway_connect` . . . 399
16.22.3 `system.gateway_auth.{gateway_id}` . . . 400
16.22.4 `system.gateway_reject` . . . 401
16.22.5 `system.gateway_disconnect` . . . 402
16.22.6 `system.gateway_heartbeat` . . . 402
16.22.7 `system.gateway_bye.{gateway_id}` . . . 403
16.22.8 `system.eod` . . . 403
16.22.9 `system.book_checkpoint.{symbol}` . . . 404
16.22.10 `system.startup` . . . 405
16.22.11 `system.startup_recovery` . . . 407
16.22.12 `system.recovery_item` . . . 408
16.22.13 `system.quote_seed_skipped` . . . 410
16.22.14 `system.diagnostic` . . . 410
16.22.15 `system.symbols_request` . . . 412
16.22.16 `system.symbols.{gateway_id}` . . . 412
16.22.17 `system.reference_request` . . . 413
16.22.18 `system.reference.{gateway_id}` . . . 413
16.22.19 `system.reference_reload` . . . 415
16.22.20 `system.reference_reload_ack.{gateway_id}` . . . 415
16.22.21 `system.session_state_request` . . . 416
16.22.22 `system.session_status.{gateway_id}` . . . 416
16.22.23 `system.session_schedule_request` . . . 417
16.22.24 `system.session_schedule.{gateway_id}` . . . 418
16.22.25 `system.halt_status_request` . . . 418
16.22.26 `system.halt_status.{gateway_id}` . . . 418
16.22.27 `system.position_request` . . . 419
16.22.28 `system.position_snapshot.{gateway_id}` . . . 419
16.22.29 `system.quote_bootstrap_request` . . . 420
16.22.30 `system.quote_bootstrap.{gateway_id}` . . . 420
16.22.31 `system.quote_legs_request` . . . 421
16.22.32 `system.quote_legs.{gateway_id}` . . . 422
16.22.33 `system.risk_state_request` . . . 423
16.22.34 `system.risk_state.{gateway_id}` . . . 423
16.22.35 `system.gateways_request` . . . 423
16.22.36 `system.gateways.{gateway_id}` . . . 424
16.22.37 `system.volume_request` . . . 424
16.22.38 `system.volume.{gateway_id}` . . . 424
16.23 Family `trade` . . . 425
16.23.1 `trade.executed` . . . 425

CHAPTER 1

How to Use This Book

Protocols and Clients is for **connecting your own program to EduMatcher**: a trading client, a market-data consumer, a reconciliation tool, a dashboard or a test harness. It describes every external protocol, how a session with each gateway behaves, every message, and how to write and test a client.

Who it is for	Developers of clients, feed handlers, dashboards and tools
What it assumes	You can run the exchange (the Quick Start Guide) and you know the trading basics — orders, fills, the order book (the Participant Guide, Part I)
What it leaves to other books	Starting and configuring the gateways (Operator's Guide, Part V); how the engine works inside (Architecture and Developer Guide); the AI traders, which are ALF clients like any other (AI Traders and Market Simulation)

1.1 How the book is organized

Part	What you find
I. Choosing and connecting	Which protocol fits which purpose, and how they compare
II. Specifications	The normative reference for ALF, BALF, CALF, RALF, LALF and the REST and WebSocket API
III. Session behavior	How the market-data and drop-copy feeds behave over time: snapshots, sequence numbers, gaps and replay
IV. Writing clients	Example client libraries in Python and C, and the spy tools that show you exactly what a gateway sends

Part	What you find
V. Message reference	Every internal message and field, generated from the message specifications

1.2 Where to start

1. Protocols Overview tells you which protocol to use. For a dashboard or a web application, the REST and WebSocket API is usually the right answer.
2. Read that protocol's specification in Part II, and its session behavior in Part III where there is one.
3. Start from the matching example in Protocol Support Library Examples, and watch the gateway with a spy tool while you develop.

The Training Guide's chapters 23–28 are exercises against each live gateway.

CHAPTER 2

Protocols Overview

Learning objectives

After reading this page you will understand:

- which external protocols EduMatcher defines and why each exists
- the runtime status of each protocol and the gateway/process that serves it
- where to find each protocol's formal specification appendix
- where to find the operational chapters that explain how each protocol is used

2.1 Why this page exists

EduMatcher exposes and documents six external protocol families, plus the internal ZeroMQ message bus that ties its own processes together:

1. ALF — order entry
2. BALF — binary order entry
3. CALF — market data
4. RALF — post-trade dissemination
5. LALF — centralized process logging
6. REST/WebSocket (`pm-api-gwy`) — HTTP access to order entry and history

A seventh, narrower wire format, DC1 (spoken by `pm-dc-gwy`), gives plain-TCP clients a text-line view of the same fill data RALF's `DROP_COPY` role and the engine's own drop-copy PUB socket already carry — see Drop Copy for how the three relate.

They serve different connectivity purposes. This page is the quick map that connects each protocol to:

- its purpose
- runtime status
- gateway/process context
- detailed chapters
- formal protocol reference appendix

2.2 Protocol map

Protocol	Primary purpose	Transport/format	Prices on the wire	Runtime status	Typical gateway/process
ALF	Human-readable order entry and gateway control	Text line protocol (`FIELD=VALUE|...`)	Decimal display price	Implemented and active	`pm-alf-console` (interactive) · `pm-alf-gwy` (TCP)
BALF	Low-latency binary order entry for programmatic clients	Binary framed protocol	`i64` fixed-point, scale 1e8	Implemented and active	`pm-balf-gwy`
CALF	External market-data dissemination (top/book/trade/state style channels)	Text line protocol over TCP (`CALF1`)	Decimal display price	Implemented and active	`pm-md-gwy`
RALF	External post-trade dissemination for clearing, drop-copy, and audit consumers	Text line protocol over TCP (`RALF1`)	Decimal display price	Implemented and active	`pm-ralf-gwy`
LALF	Centralized process logging	Text line protocol over TCP	n/a	Implemented and active	`pm-log-srv`
REST/WebSocket	HTTP/WS access to order entry and history, for browser UIs and dashboards	JSON over HTTP + WebSocket	Decimal display price	Implemented and active	`pm-api-gwy`

BALF is the one protocol here whose wire encoding of price actually differs — a scaled 64-bit integer, not a decimal string — so a client written against ALF/CALF/RALF/REST needs a different price parser for BALF.

2.3 ALF (Almost FIX)

ALF is EduMatcher's active, user-facing order-entry protocol. It is used by interactive participant gateways and is the primary way traders submit and manage orders in current deployments.

ALF is available in two runtime forms:

Process	Purpose	Transport
`pm-alf-console`	Interactive REPL for a human at a terminal — stdin/stdout, tab completion, P&L display	Local process; stdin/stdout
`pm-alf-gwy`	TCP gateway for external bots and remote clients — same ALF protocol over a plain TCP socket	TCP (default port `5565`)

Use ALF when you need:

- interactive/manual order entry → `pm-alf-console`
- educational readability of commands
- direct access to the full command set
- an external bot or remote process submitting orders → `pm-alf-gwy`

Where to read more:

- ALF TCP gateway operational guide: ALF TCP Gateway
- Interactive client behavior and operator workflow (`pm-alf-console` is a client, not a gateway): ALF Console
- Process-level role of `pm-alf-console`: Processes, Environment and Ports
- Process-level role of `pm-alf-gwy`: Processes, Environment and Ports
- Engine configuration of allowed ALF IDs/roles: The Configuration Workflow
- Formal wire syntax and semantics: Appendix - ALF Protocol

2.4 BALF (Binary ALF)

BALF is the binary order-entry protocol family. It targets programmatic clients that need compact framing and lower parsing overhead than text order-entry formats.

BALF is implemented and runs as `pm-balf-gwy`, a TCP gateway that accepts binary order-entry frames from programmatic clients.

Use BALF when you need:

- binary order-entry framing
- explicit low-latency protocol design
- a programmatic session model with binary message layouts

Where to read more:

- Operational deployment and runbook: BALF TCP Gateway
- Runtime process and architecture placement: Processes, Environment and Ports
- Protocol design details and message/frame definitions: Appendix - BALF Protocol
- Configuration context and protocol family notes: The Configuration Workflow

2.5 CALF (Channel ALF)

CALF is the external market-data protocol family. It is designed for subscription-based market-data delivery (channelized streams, snapshot + incremental patterns, and sequence-aware recovery semantics).

CALF is implemented via `pm-md-gwy` and is used for external market-data distribution with snapshot, incremental, and replay-aware reconnect semantics.

Use CALF when you need:

- external market-data subscription channels
- deterministic sequence/reconnect semantics for data consumers
- a text-based market-data feed for educational and integration scenarios
- to subscribe to the index for the exchange

Where to read more:

- Market-data concepts and channel model: Market Data Feed (CALF)
- Runtime process and architecture placement: Processes, Environment and Ports
- Operational client onboarding and examples: Market Data Feed (CALF)
- Formal wire protocol reference: Appendix - CALF Protocol

2.6 RALF (Reconciliation ALF)

RALF is EduMatcher's active post-trade dissemination protocol. It is used by `pm-ralf-gwy` to stream post-trade events to external systems such as clearing, drop-copy, and audit consumers.

Use RALF when you need:

- external post-trade event distribution
- role-based consumption (`CLEARING`, `DROP_COPY`, `AUDIT`)
- replay-aware reconnect behavior for downstream systems

Where to read more:

- Operational deployment and runbook: Post-Trade Dissemination (RALF)
- Process-level role in runtime topology: Processes, Environment and Ports
- RALF gateway configuration details: The Configuration Workflow
- Formal wire protocol reference: Appendix - RALF Protocol

2.7 LALF (Log ALF)

LALF is the protocol `pm-log-srv` speaks to collect operational log lines from every other `pm-*` process, and to distribute them live to log viewers. It carries no trading data.

Where to read more:

- Operational guide: Log Server
- Formal wire protocol reference: Appendix - LALF Protocol

2.8 REST/WebSocket (`pm-api-gwy`)

The REST API is EduMatcher's HTTP-native external interface, intended for browser UIs, dashboards, and simple bots that would rather speak JSON over HTTP than a text-line protocol. It translates requests into the same order flow and history queries the other gateways use, and exposes a WebSocket for live updates.

Use REST/WebSocket when you need:

- a browser-based UI or dashboard
- a scripting language with an easy HTTP client but no ZeroMQ bindings

- JSON request/response instead of a line protocol

Where to read more:

- Endpoint contract: REST API Reference and WebSocket API Reference
- Running the gateway: API Gateway

2.9 Drop copy and DC1 (`pm-dc-gwy`)

Drop copy is not a separate protocol family in the sense of the six above — it is a second way to reach the same fill data RALF's `DROP_COPY` role already carries. The engine publishes fills directly on its own dedicated ZMQ PUB socket (port `5557`); `pm-dc-gwy` relays that feed as a lightweight text-line protocol ("DC1") for plain-TCP clients that don't speak ZeroMQ or RALF.

Where to read more:

- Engine-side feed and wire format: Drop Copy
- The DC1 gateway and protocol: Drop-Copy Gateway
- A ready-made DC1 client for testing: dc-spy CLI

2.10 Internal ZeroMQ message bus

Everything above is the *external* surface. Internally, every EduMatcher process talks to every other one over a shared ZeroMQ bus of topic-tagged messages — the catalogue the protocols above are themselves built from.

Where to read more:

- Message framing, topics, and the full message catalogue: Message Reference

2.11 Quick selection guide

If you need to…	Protocol to start with
Enter and manage orders from participant terminals	**ALF** (`pm-alf-console`)
Submit orders from an external bot or remote process	**ALF** (`pm-alf-gwy`)
Plan binary low-latency order-entry integrations	**BALF** (`pm-balf-gwy`)
Consume market-data channels externally	**CALF** (`pm-md-gwy`)
Consume post-trade/clearing/audit streams externally	**RALF** (`pm-ralf-gwy`)
Give a plain-TCP client just the fill stream	**DC1** (`pm-dc-gwy`)
Collect operational logs centrally	**LALF** (`pm-log-srv`)
Build a browser UI, dashboard, or simple HTTP bot	**REST/WebSocket** (`pm-api-gwy`)

2.12 See also

In this book:

- Message Reference
- Drop Copy

- Appendix - ALF Protocol
- Appendix - BALF Protocol
- Appendix - CALF Protocol
- Appendix - RALF Protocol
- Appendix - LALF Protocol
- Appendix - REST API Reference

Quick Start Guide:

- What EduMatcher Is

Operator's Guide:

- ALF TCP Gateway
- BALF TCP Gateway
- Market Data Feed (CALF)
- Post-Trade Dissemination (RALF)
- Drop-Copy Gateway
- Log Server
- API Gateway

Reference Manual:

- Processes

CHAPTER 3

ALF Protocol Reference

Status: Normative. This appendix is the single source of truth for the ALF wire contract as implemented by `pm-alf-console` and `pm-alf-gwy` (`alf_gwy/`). For an operational, tutorial-style guide see ALF Console and ALF TCP Gateway; for the gateway's configuration block see Engine Config Specification §6.1. The key words MUST, MUST NOT, SHOULD, and MAY are used per RFC 2119.

3.1 What ALF is

ALF stands for **Almost FIX**.

It is EduMatcher's compact text command protocol for entering orders through `pm-alf-console`. ALF borrows the **field=value** idea from FIX, but it is **not** FIX:

- there are no numeric tags
- there is no FIX session layer
- there are no sequence numbers
- there is no checksum or body length
- there are no repeating-group rules
- there are no standard FIX message types

ALF is intentionally smaller and easier to type by hand:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.25
```

This appendix is the **normative reference** for ALF syntax and semantics. If another tool, bot, or GUI wants to generate ALF commands, this document is the correct source to follow.

3.2 Scope & conformance

ALF is the **text command language accepted by `pm-alf-console`**. It is not the internal engine wire format. The gateway converts ALF commands into the engine's ZeroMQ + JSON message payloads.

That distinction matters:

- **ALF** is what a human or external client writes
- **engine messages** are what the gateway publishes internally

This appendix describes the **ALF side**, not the internal message bus.

3.3 Wire format

3.3.1 Line structure

Every ALF command is a single line:

```
<COMMAND>|<FIELD>=<VALUE>|<FIELD>=<VALUE>|...
```

The first token is always the command verb. Everything after that is a pipe-delimited list of fields.

3.3.2 Formal grammar

This notation is **ABNF-style** (Augmented Backus-Naur Form), not classic BNF. The repetition form:

```
n*m element
```

means "repeat `element` at least `n` times and at most `m` times".

Examples:

- `1*X` means **one or more** `X`
- `*X` means **zero or more** `X`
- `2*4X` means **between 2 and 4** `X`

So in the ALF grammar:

- `command = 1*( ALPHA / "_" )` means a command is made of **one or more** letters or underscores
- `key = 1*( ALPHA / DIGIT / "_" / "." )` means a key is **one or more** letters, digits, underscores, or dots

```
alf-line      = command *( "|" segment )
command       = 1*( ALPHA / "_" )
segment       = key "=" value
key           = 1*( ALPHA / DIGIT / "_" / "." )
value         = *VCHAR
```

`VCHAR` is also ABNF terminology. It means a **visible printable character**. So:

- `value = *VCHAR` means the value may contain **zero or more visible characters**
- the leading `*` is why the value is allowed to be empty

In strict ABNF, `VCHAR` usually means printable non-space ASCII characters. For ALF, read this line as **simplified shorthand** for:

> everything after the first `=` up to the next `|`

That is closer to the real gateway parser behavior. The parser does not enforce a full RFC-style character class check; it splits on `|`, then splits each segment on the first `=`, and treats the remainder as the raw value text.

In practice:

- `|` separates segments
- the **first** `=` in a segment separates key from value
- no quoting or escaping layer exists

3.3.3 Whitespace

The gateway trims whitespace only at the very start and end of the full line. It does **not** trim spaces around individual fields. Write:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.25
```

not:

```
NEW | SYM=AAPL | SIDE=BUY
```

3.3.4 Case handling

ALF is effectively **case-insensitive** for command verbs, field names, enum values, and symbols, because the gateway normalizes parsed *inbound* values to uppercase. This is inbound-only: values the gateway sends back out (order IDs, `REASON`/`DETAIL` text, and similar) are not case-normalized and keep their original case.

The one inbound exception is `ID` on `AMEND` and `CANCEL`. Order IDs are lowercase hexadecimal, so the gateway passes `ID` through unchanged: echo the `ORDER_ID` exactly as you received it.

These are treated the same:

```
new|sym=aapl|side=buy|type=limit|qty=100|price=150.25
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.25
```

Important consequence: user-supplied labels are also normalized to uppercase.

Examples:

- `COMBO_ID=pair-01` becomes `PAIR-01`
- `OCO_ID=exit-aapl` becomes `EXIT-AAPL`
- `QUOTE_ID=mm-1` becomes `MM-1`

If you generate ALF programmatically, assume that **labels and symbols are case-folded to uppercase** by the gateway.

3.3.5 Field order

After the command word, field order is generally **not significant**. The gateway parses fields into a key/value map and then validates what is present.

These are equivalent:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.25
NEW|TYPE=LIMIT|PRICE=150.25|QTY=100|SIDE=BUY|SYM=AAPL
```

Exceptions:

- the command verb must be the first token
- combo leg numbering must be consistent with `LEG_COUNT`

3.3.6 Duplicate and unknown fields

If the same field appears more than once, the **last occurrence wins**.

Example:

```
NEW|SYM=AAPL|SYM=MSFT|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.25
```

This is parsed as `SYM=MSFT`.

Unknown fields are ignored by the gateway unless they are required for the specific command being validated.

The price fields are the exception: `PRICE`, `STOP`, `VISIBLE` and `TRAIL` (and their `LEGn_` / `LEGi.` forms) follow one per-type rule, the same one REST, BALF and the engine apply. A field the `TYPE` needs

and lacks is `MISSING_FIELD` (`"STOP_LIMIT requires PRICE"`). A field the `TYPE` does not use is refused, not ignored: `UNSUPPORTED_FIELD` (`"LIMIT forbids TRAIL"`). A leg type the leg has no field for (an `ICEBERG` OCO or combo leg) is `INVALID_VALUE`. The same order gets the same `REJECT_CODE` on every transport.

3.3.7 Numeric formatting

ALF uses human-readable decimal prices:

- `PRICE=150.25`
- `STOP=149.90`
- `TRAIL=1.50`

Quantities are integers:

- `QTY=100`
- `VISIBLE=25`

The gateway converts decimal prices to the symbol's internal tick representation before sending them to the engine.

3.3.8 No escaping

ALF has **no escaping mechanism** for `|`. Do not place `|` inside values.

The first `=` in a segment splits key from value. Everything after that first `=` is considered part of the value, but values containing additional `=` are not part of the documented protocol and should be avoided.

3.4 Common field vocabulary

Field	Meaning
`SYM`	Symbol, for example `AAPL`
`SIDE`	`BUY` or `SELL`
`TYPE`	Order type or high-level order family such as `COMBO` or `OCO`
`QTY`	Integer quantity
`PRICE`	Limit price in display units
`STOP`	Stop trigger price in display units
`TRAIL`	Trailing-stop offset in display units
`TIF`	Time in force: `DAY`, `GTC`, `ATO`, `ATC`
`VISIBLE`	Displayed peak size for iceberg orders
`SMP`	Self-match-prevention action
`ID`	Engine-generated single-order identifier
`COMBO_ID`	User-supplied combo label
`OCO_ID`	User-supplied OCO label
`QUOTE_ID`	User-supplied quote label

3.4.1 Supported `TIF` values

Value	Meaning
`DAY`	Valid for the trading day only
`GTC`	Good till cancelled
`ATO`	At the open; only meaningful during the opening auction
`ATC`	At the close; only meaningful during the closing auction

3.4.2 Supported `SMP` values

Value	Meaning
`NONE`	Allow self-trades
`CANCEL_AGGRESSOR`	Cancel the incoming order if it would self-match
`CANCEL_RESTING`	Cancel the resting order and continue matching
`CANCEL_BOTH`	Cancel both the incoming and resting order

If `SMP` is omitted entirely, the engine falls back to the submitting gateway's configured `participants[].smp_actio` (or `NONE` if the gateway has none configured) — see Configuration — Participant Fields. This is different from sending `SMP=NONE` explicitly, which is always honoured as-is even if the gateway has a non-`NONE` default configured.

3.5 Session establishment — `HELLO` (`pm-alf-gwy` only)

`pm-alf-console` already knows its own gateway identity from its `--id` command-line flag, so its one session is authenticated before the first command is read — there is no handshake to perform. `pm-alf-gwy` is a shared, multi-client TCP server: every connection **must** send `HELLO` as the first line, before any other command, or the gateway rejects it.

3.5.1 Formal syntax

```
HELLO|CLIENT=<client-name>|PROTO=ALF1|ID=<gateway-id>[|FEED=<ALL|ORDERS>]
```

Field	Required	Meaning
`CLIENT`	Yes	Free-text client name, max 32 characters
`PROTO`	Yes	Must be exactly `ALF1`
`ID`	Yes	Gateway ID to authenticate as (the same `participants[].id` ALF and BALF share): a letter, then up to 31 letters, digits or `_`
`FEED`	No	`ALL` (default): also receive this gateway's public/market-data-style broadcasts; `ORDERS`: only this session's own order/quote traffic

3.5.2 Successful handshake

`HELLO` triggers an engine allowlist check. On success the gateway replies `WELCOME` and then sends the current symbol list:

```
> HELLO|CLIENT=my-bot|PROTO=ALF1|ID=TRADER01
WELCOME|PROTO=ALF1|GW=<gateway-name>|ID=TRADER01|HBINT=<heartbeat_interval_sec>|IDLE=<idle_timeout_sec>
```

`HBINT`/`IDLE` echo the gateway's configured `heartbeat_interval_sec` / `idle_timeout_sec`. See ALF TCP Gateway → Session lifecycle for the full sequence diagram and ALF TCP Gateway → Configuration for their real defaults (`idle_timeout_sec`'s default depends on whether the config even has an `alf_gateway:` section).

3.5.3 Pre-authentication error codes

Only `HELLO` is accepted before authentication completes; every other command gets `AUTH_REQUIRED`. Most of these close the connection:

CODE	Meaning	Closes connection?
`MISSING_FIELD`	`CLIENT` or `ID` absent	Yes
`INVALID_VALUE`	`CLIENT` longer than 32 characters, or `ID` not a letter followed by up to 31 letters, digits or `_`	Yes
`PROTO_MISMATCH`	`PROTO` is not `ALF1`	Yes
`GATEWAY_ALREADY_CONNECTED`	This `ID` already has an active session	Yes
`HELLO_ALREADY_PENDING`	A second `HELLO` arrived while the first is still awaiting the engine's auth reply	No
`ENGINE_UNAVAILABLE`	The gateway could not forward the connect request to the engine	No
`AUTH_FAILED`	The engine's allowlist rejected this `ID`	Yes
`AUTH_REQUIRED`	Any command other than `HELLO` arrived before authentication completed	Yes
`AUTH_TIMEOUT`	No `HELLO` at all within `handshake_timeout_sec` of connecting	Yes
`RATE_LIMITED`	Too many commands per second before authentication	No
`BAD_MESSAGE`	The line could not be parsed as `COMMAND|KEY=VALUE|...`	No

3.5.4 `PING` / `PONG`

Once authenticated, a client may send `PING` at any time; the gateway replies `PONG|TS=<iso-8601-timestamp>`. This is a session-local liveness check — distinct from the engine-level heartbeat the gateway itself sends the engine — and counts as inbound traffic against `idle_timeout_sec`.

3.6 Command families

ALF commands fall into six groups:

Group	Commands
Session establishment (`pm-alf-gwy` only)	`HELLO`, `PING`

Group	Commands
Trading commands forwarded to the engine	`NEW`, `AMEND`, `CANCEL`, `QUOTE`, `QUOTE_CANCEL`, `KILL`, `SYMBOLS`, `ORDERS`, `QBOOT`, `SESSION`
Client-local informational commands	`STATUS`, `HELP`
Divergent between clients (see note below)	`QLEGS`, `POS`
Drop-copy relay control (gateway-local)	`DC`
Session-control commands for the CLI process	`EXIT`, `QUIT`

If you are writing another interface that wants to submit orders, the most important commands are the **trading commands**.

Rejected order acknowledgements carry two diagnostics: `REJECT_CODE=<code>` is the canonical machine-readable code, and `REASON=<text>` is the human-readable explanation. Use `REJECT_CODE` for branching, metrics, and tests; log `REASON` for operators.

On `pm-alf-gwy`, `FILL` also carries the order's `SYMBOL` and `SIDE`.

`FILL` carries `TRADE_IDS=<id[,id...]>`, the comma-separated durable public trade IDs that composed the private fill. A one-level fill has one ID; a coalesced sweep has one ID per execution in match order. The field is present as `TRADE_IDS=` when no ID is available.

`FILL` also carries `LIQUIDITY=<MAKER|TAKER>`: whether *this* order supplied liquidity (rested on the book and was matched into) or removed it (crossed the spread and matched immediately). It is derived from the trade's aggressor side, the same rule the drop-copy `DC_FILL` line's `LIQUIDITY` field already used — the two now always agree for the same fill. A resting order that gets matched reports `MAKER`; the order that crossed to match it reports `TAKER`. One coalesced `FILL` covering several trades still reports a single, consistent value, because one order cannot be its own counterparty's maker on one trade and taker on another.

Gateway-local errors use the same canonical field: `ERR|CODE=<legacy-code>|REJECT_CODE=<canonical-code>|DET` `CODE` names the ALF gateway validation condition; `REJECT_CODE` is the transport-independent order rejection class used for ALF/REST comparison.

QLEGS behaves differently in each client

Every other command in the "trading commands" row above is a genuine engine round trip in **both** `pm-alf-console` and `pm-alf-gwy`. `QLEGS` is the one exception: in `pm-alf-console` it renders entirely from a local, session-scoped cache built by observing this session's own quote/fill/cancel events and never contacts the engine, while `pm-alf-gwy` forwards it to the engine's `system.quote_legs_request` and renders the engine's authoritative reply. The command syntax and output columns are identical either way — see the `QLEGS` section below and ALF TCP Gateway → QLEGS for the `pm-alf-gwy` wire format.

`STATUS` is `pm-alf-console`-only: `pm-alf-gwy` rejects it with `ERR|CODE=UNKNOWN_COMMAND` since it has no interactive terminal to print a summary to.

`POS` also behaves differently in each client, but in the other direction from `QLEGS`: bare `POS` (no `GW=`) in `pm-alf-console` prints that session's own locally-tracked fills and P&L, and `pm-alf-gwy` still rejects a bare `POS` — it keeps no such local ledger. `POS|GW=<gateway_id>` is the form that **is** a genuine engine round trip in both clients: it asks the engine's `system.position_request` /

`system. position_snapshot.{GW_ID}` for *any* gateway's position (an MM bot's, typically), not just the caller's own. See ALF TCP Gateway → POS for the `pm-alf-gwy` wire format.

3.7 Single-leg `NEW` orders

3.7.1 Generic form

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=<order-type>|QTY=<quantity>[|...]
```

`TAG=<order-tag>` is optional on every `NEW` form. It identifies the order and is echoed on `ACK`, `FILL`, `AMENDED`, `CANCELLED`, and `EXPIRED` when known. Do not use `TAG` on `AMEND` or `CANCEL`; those request-scoped commands use `RTAG` instead.

3.7.2 Formal grammar

```
new-single =
    "NEW"
    "|SYM=" symbol
    "|SIDE=" side
    "|TYPE=" order-type
    "|QTY=" quantity
    *( "|" option )
    [ "|TAG=" tag ]
```

Where `option` depends on the selected order type.

3.7.3 Supported single-leg order types

Type	Required fields	Optional fields	Notes
`MARKET`	`SYM`, `SIDE`, `TYPE=MARKET`, `QTY`	`TIF`, `SMP`	`TIF` is accepted syntactically but market-order behavior still applies
`LIMIT`	`SYM`, `SIDE`, `TYPE=LIMIT`, `QTY`, `PRICE`	`TIF`, `SMP`	Standard resting limit order
`IOC`	`SYM`, `SIDE`, `TYPE=IOC`, `QTY`, `PRICE`	`TIF`, `SMP`	Immediate-or-cancel behavior still applies even if `TIF` is supplied
`FOK`	`SYM`, `SIDE`, `TYPE=FOK`, `QTY`, `PRICE`	`TIF`, `SMP`	Fill-or-kill behavior still applies even if `TIF` is supplied
`STOP`	`SYM`, `SIDE`, `TYPE=STOP`, `QTY`, `STOP`	`TIF`, `SMP`	Becomes a market order when triggered
`STOP_LIMIT`	`SYM`, `SIDE`, `TYPE=STOP_LIMIT`, `QTY`, `STOP`, `PRICE`	`TIF`, `SMP`	Becomes a limit order when triggered
`ICEBERG`	`SYM`, `SIDE`, `TYPE=ICEBERG`, `QTY`, `PRICE`, `VISIBLE`	`TIF`, `SMP`	`VISIBLE` must be strictly less than `QTY`

Type	Required fields	Optional fields	Notes
`TRAILING_STOP`	`SYM`, `SIDE`, `TYPE=TRAILING_STOP`, `QTY`, `TRAIL`	`STOP`, `TIF`, `SMP`	`STOP` is optional; if omitted the engine derives the initial stop from the last trade price

3.7.4 Formal syntax by order type

3.7.5 MARKET

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=MARKET|QTY=<quantity>[|TIF=<DAY|GTC|ATO|ATC>][|SMP=<...>]
```

Example:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=MARKET|QTY=100
```

3.7.6 LIMIT

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=LIMIT|QTY=<quantity>|PRICE=<price>[|TIF=<DAY|GTC|ATO|ATC>][|SMP=<...>]
```

Examples:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=152.00
NEW|SYM=MSFT|SIDE=BUY|TYPE=LIMIT|QTY=200|PRICE=310.00|TIF=GTC
```

3.7.7 IOC

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=IOC|QTY=<quantity>|PRICE=<price>[|TIF=<DAY|GTC|ATO|ATC>][|SMP=<...>]
```

Example:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=IOC|QTY=250|PRICE=150.20
```

3.7.8 FOK

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=FOK|QTY=<quantity>|PRICE=<price>[|TIF=<DAY|GTC|ATO|ATC>][|SMP=<...>]
```

Example:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=FOK|QTY=100|PRICE=150.00
```

3.7.9 STOP

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=STOP|QTY=<quantity>|STOP=<stop-price>[|TIF=<DAY|GTC|ATO|ATC>][|SMP=<...
```

Example:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=STOP|QTY=100|STOP=148.00
```

3.7.10 STOP_LIMIT

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=STOP_LIMIT|QTY=<quantity>|STOP=<stop-price>|PRICE=<limit-price>[|TIF=<D
```

Example:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=STOP_LIMIT|QTY=100|STOP=148.00|PRICE=147.50
```

3.7.11 ICEBERG

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=ICEBERG|QTY=<quantity>|PRICE=<price>|VISIBLE=<visible-quantity>[|TIF=<DAY|GTC|A
```

Example:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=ICEBERG|QTY=1000|PRICE=150.00|VISIBLE=100
```

3.7.12 TRAILING_STOP

```
NEW|SYM=<symbol>|SIDE=<BUY|SELL>|TYPE=TRAILING_STOP|QTY=<quantity>|TRAIL=<offset>[|STOP=<initial-stop>][|TIF=<DAY|GTC
```

Examples:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=TRAILING_STOP|QTY=100|TRAIL=1.50|STOP=148.00
NEW|SYM=AAPL|SIDE=SELL|TYPE=TRAILING_STOP|QTY=100|TRAIL=1.50
```

Semantics:

- `TRAIL` is mandatory
- `STOP` is optional
- if `STOP` is omitted, the engine derives the initial stop from the last trade price
- if `STOP` is omitted and no prior trade exists, the engine rejects the order

When a `STOP`, `STOP_LIMIT` or `TRAILING_STOP` fires, `pm-alf-gwy` sends `TRIGGERED` before the order's own `FILL`s:

```
TRIGGERED|ORDER_ID=<id>|SYMBOL=<symbol>|TYPE=<STOP|STOP_LIMIT|TRAILING_STOP>|STOP=<price>|BECOMES=<MARKET|LIMIT>[|TRA
```

`STOP` is the trigger it fired at (a trailing stop's ratcheted one); `TRADE_ID`/`TRADE_PRICE` name the trade that reached it and are absent when the last price came from an engine restart. `BECOMES` is `MARKET` for `STOP` and `TRAILING_STOP`, `LIMIT` for `STOP_LIMIT`.

3.7.13 Auction-only `TIF` values

`ATO` and `ATC` are accepted by the single-leg parser even though they are not prominent in tab completion.

Examples:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=ATO
NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=151.00|TIF=ATC
```

These are only meaningful during the opening and closing auction phases.

3.7.14 Practical parsing notes

- `PRICE`, `STOP`, and `TRAIL` are decimal display prices, not ticks
- `QTY` and `VISIBLE` must parse as integers
- `VISIBLE` must be strictly less than total `QTY`
- the gateway generates the final engine order ID; you do not supply it in `NEW`

3.8 `AMEND` - order amendments

`AMEND` applies only to **existing single-leg orders** identified by full engine-generated order ID.

3.8.1 Formal syntax

```
AMEND|ID=<order-id>[|PRICE=<new-price>][|QTY=<new-total-quantity>][|RTAG=<request-tag>]
```

At least one of `PRICE` or `QTY` must be present. `RTAG` is optional and is echoed on the corresponding `AMENDED` event or rejected `ACK`.

3.8.2 Supported amendment forms

Form	Meaning
`AMEND|ID=...|PRICE=...[|RTAG=...]`	Price-only amendment
`AMEND|ID=...|QTY=...[|RTAG=...]`	Quantity-only amendment
`AMEND|ID=...|PRICE=...|QTY=...[|RTAG=...]`	Change both price and quantity

3.8.3 Examples

Price only:

```
AMEND|ID=ORD-12345678|PRICE=151.00|RTAG=AMD-001
```

Quantity only, lower quantity:

```
AMEND|ID=ORD-12345678|QTY=80|RTAG=AMD-002
```

Quantity only, higher quantity:

```
AMEND|ID=ORD-12345678|QTY=200|RTAG=AMD-003
```

Change both:

```
AMEND|ID=ORD-12345678|PRICE=151.00|QTY=200|RTAG=AMD-004
```

3.8.4 Amendment semantics

As implemented today:

- only **resting `LIMIT` and `ICEBERG` orders** can be amended
- `MARKET`, `IOC`, `FOK`, `STOP`, `STOP_LIMIT`, and `TRAILING_STOP` orders are not amendable
- `QTY` is the **new total order quantity**, not a delta
- `QTY` must stay positive
- the new total quantity must be **greater than the already-filled quantity**

3.8.5 Priority rules

The engine follows exchange-style amendment priority rules:

- **quantity decrease only, same price** -> priority is preserved
- **price change** -> priority is reset
- **quantity increase** -> priority is reset

What actually determines priority: arrival_seq, not timestamp

"Priority" here means queue position under price-time priority, and it is decided by **`arrival_seq`** — a monotonic counter the *engine* assigns each time it admits an order (or a re-priced/re-queued amendment) into the book. It is **not** the order's `timestamp` field, which is supplied by the *client* at submission time and is never used by the book to order the queue (a back-dated `timestamp` cannot jump the line).

Do not confuse `arrival_seq` with the "no sequence numbers" bullet earlier on this page — that refers to ALF having no FIX-style `MsgSeqNum`/session-layer sequencing on the wire. `arrival_seq` is an unrelated, internal engine field that is assigned after ALF parses the command and is visible on

outbound `order.orders.{GW_ID}` snapshots and `order.amended.{GW}` events. See the Order Types — Priority Rules and the message reference for the field-level detail.
Example:

1. `NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00`
2. `AMEND|ID=...|QTY=80|RTAG=AMD-KEEP` keeps queue priority (`arrival_seq` unchanged) if the price stays `150.00`
3. `AMEND|ID=...|PRICE=150.10|RTAG=AMD-PRICE` loses queue priority (new `arrival_seq` assigned)
4. `AMEND|ID=...|QTY=120|RTAG=AMD-UP` also loses queue priority (new `arrival_seq` assigned)

3.8.6 Important identifier rule

Use the **full order ID** from the gateway's order table or acknowledgement stream. Inline gateway messages often abbreviate IDs visually, but `AMEND` requires the full internal ID.

3.9 `CANCEL` - order and group cancellation

3.9.1 Single order cancel

```
CANCEL|ID=<order-id>[|RTAG=<request-tag>]
```

Example:

```
CANCEL|ID=ORD-12345678|RTAG=CXL-001
```

`RTAG` is optional on single-order cancels and is echoed on the resulting `CANCELLED` event or rejected `ACK`. Engine-initiated cancels have no client request behind them and therefore omit `RTAG`.

3.9.2 Combo cancel

```
CANCEL|COMBO_ID=<combo-label>
```

Example:

```
CANCEL|COMBO_ID=PAIR-001
```

3.9.3 OCO cancel

```
CANCEL|OCO_ID=<oco-label>
```

Example:

```
CANCEL|OCO_ID=EXIT-AAPL
```

3.9.4 Cancellation semantics

- `ID` cancels one single-leg order
- `COMBO_ID` cancels a combo and its still-resting child legs
- `OCO_ID` cancels the whole OCO pair
- fills that already happened are never reversed by cancel

3.10 `NEW|TYPE=OCO` - one-cancels-other pairs

OCO syntax creates **two linked orders on the same symbol and with the same quantity**. When one leg fills or is otherwise actioned, the sibling is cancelled.

3.10.1 Formal syntax

```
NEW
|TYPE=OCO
|OCO_ID=<label>
|SYM=<symbol>
|QTY=<quantity>
[|TIF=<DAY|GTC|ATO|ATC>]
|LEG1_SIDE=<BUY|SELL>
|LEG1_TYPE=<order-type>
[|LEG1_PRICE=<price>]
[|LEG1_STOP=<stop-price>]
[|LEG1_TRAIL=<trail-offset>]
|LEG2_SIDE=<BUY|SELL>
|LEG2_TYPE=<order-type>
[|LEG2_PRICE=<price>]
[|LEG2_STOP=<stop-price>]
[|LEG2_TRAIL=<trail-offset>]
```

3.10.2 OCO field semantics

Field	Meaning
`OCO_ID`	User label for the pair
`SYM`	Shared symbol for both legs
`QTY`	Shared quantity for both legs
`TIF`	Shared TIF for both legs
`LEG1_*`, `LEG2_*`	Per-leg side, type, and trigger/limit fields

If `TIF` is omitted, the gateway defaults it to `DAY`.

3.10.3 OCO leg expressibility

ALF OCO syntax can express these leg types cleanly:

Leg type	Supported in ALF OCO syntax	Required extra fields
`MARKET`	Yes	none
`LIMIT`	Yes	`LEGn_PRICE`
`IOC`	Yes	`LEGn_PRICE`
`FOK`	Yes	`LEGn_PRICE`
`STOP`	Yes	`LEGn_STOP`
`STOP_LIMIT`	Yes	`LEGn_STOP` and `LEGn_PRICE`
`TRAILING_STOP`	Yes	`LEGn_TRAIL`, optional `LEGn_STOP`
`ICEBERG`	No, refused (`INVALID_VALUE`)	An OCO leg has no visible quantity

Any other price field on a leg is refused (`UNSUPPORTED_FIELD`), as on a single order.

3.10.4 Examples

Classic take-profit / stop-loss exit:

```
NEW|TYPE=OCO|OCO_ID=EXIT-AAPL|SYM=AAPL|QTY=100|TIF=GTC|LEG1_SIDE=SELL|LEG1_TYPE=LIMIT|LEG1_PRICE=155.00|LEG2_SIDE=SE
```

Limit exit plus trailing-stop protection:

```
NEW|TYPE=OCO|OCO_ID=EXIT-TRAIL|SYM=AAPL|QTY=100|TIF=GTC|LEG1_SIDE=SELL|LEG1_TYPE=LIMIT|LEG1_PRICE=155.00|LEG2_SIDE=S
```

3.10.5 OCO semantics

- both legs share the same `SYM`, `QTY`, `TIF`, and gateway identity
- each leg receives its own engine-generated order ID
- the gateway first receives an `OCO_ACK`
- each leg is then posted and acknowledged as a normal order
- when one leg fills or is cancelled in a way that completes the OCO action, the sibling is auto-cancelled

3.11 `NEW|TYPE=COMBO` - multi-leg combo orders

Combo syntax creates a parent combo plus child orders across multiple symbols.

3.11.1 Formal syntax

```
NEW
|TYPE=COMBO
|COMBO_ID=<label>
[|COMBO_TYPE=AON]
[|TIF=<DAY|GTC|ATO|ATC>]
[|SMP=<NONE|CANCEL_AGGRESSOR|CANCEL_RESTING|CANCEL_BOTH>]
|LEG_COUNT=<n>
|LEG0.SYM=<symbol>
|LEG0.SIDE=<BUY|SELL>
|LEG0.QTY=<quantity>
[|LEG0.TYPE=<order-type>]
[|LEG0.PRICE=<price>]
...
|LEG<n-1>.SYM=<symbol>
|LEG<n-1>.SIDE=<BUY|SELL>
|LEG<n-1>.QTY=<quantity>
[|LEG<n-1>.TYPE=<order-type>]
[|LEG<n-1>.PRICE=<price>]
[|LEG<n-1>.STOP=<price>]
```

3.11.2 Combo field semantics

Field	Meaning
`TYPE=COMBO`	Selects combo mode
`COMBO_ID`	User label used later by `CANCEL|COMBO_ID=...`
`COMBO_TYPE=AON`	Optional today; defaults to `AON`
`TIF`	Shared TIF for all child legs

Field	Meaning
SMP	Shared SMP setting applied to all child legs
LEG_COUNT	Number of legs, from 2 to 10
LEGi.*	Per-leg fields, using zero-based numbering

If COMBO_TYPE is omitted, the gateway defaults it to AON. If TIF is omitted, it defaults to DAY. If SMP is omitted, every leg falls back to the submitting gateway's configured participants[].smp_action (see Supported SMP values above) rather than a fixed NONE.

3.11.3 Combo leg expressibility

A combo leg carries LEGi.SYM, LEGi.SIDE, LEGi.QTY, LEGi.TYPE, LEGi.PRICE and LEGi.STOP. It has no LEGi.TRAIL or LEGi.VISIBLE:

Leg type	ALF combo support	Fields
LIMIT	Yes	Default leg type; LEGi.PRICE
MARKET	Yes	none
IOC	Yes	LEGi.PRICE
FOK	Yes	LEGi.PRICE
STOP	Yes	LEGi.STOP
STOP_LIMIT	Yes	LEGi.STOP and LEGi.PRICE
TRAILING_STOP	No, refused (INVALID_VALUE)	A combo leg has no trail
ICEBERG	No, refused (INVALID_VALUE)	A combo leg has no visible quantity

A missing field is MISSING_FIELD; a field the leg's type does not use is UNSUPPORTED_FIELD.

3.11.4 Examples

Two-leg pairs trade:

```
NEW|TYPE=COMBO|COMBO_ID=PAIR-001|COMBO_TYPE=AON|TIF=GTC|LEG_COUNT=2|LEG0.SYM=MSFT|LEG0.SIDE=BUY|LEG0.QTY=100|
```

Three-leg arbitrage-style order:

```
NEW|TYPE=COMBO|COMBO_ID=ARB-01|COMBO_TYPE=AON|TIF=DAY|LEG_COUNT=3|LEG0.SYM=AAPL|LEG0.SIDE=BUY|LEG0.QTY=200|LE
```

3.11.5 Combo semantics

- LEG_COUNT must be between 2 and 10
- leg numbering is zero-based: LEG0, LEG1, LEG2, …
- duplicate symbols are rejected by the engine
- the engine posts child orders to the relevant per-symbol books
- if a child leg is cancelled or expires, the remaining legs are cascade-cancelled
- combo state is tracked separately from the ordinary ORDERS table

3.12 `QUOTE` and `QUOTE_CANCEL`

Quotes are a two-sided market-maker convenience command. One `QUOTE` submits or replaces both a bid and an ask for the same symbol.

3.12.1 Formal syntax

```
QUOTE|SYM=<symbol>|BID=<bid-price>|ASK=<ask-price>|BID_QTY=<quantity>|ASK_QTY=<quantity>[|TIF=<DAY|GTC|ATO|ATC>][|QU
```

Cancel syntax:

```
QUOTE_CANCEL|SYM=<symbol>
```

3.12.2 Example

```
QUOTE|SYM=AAPL|BID=149.90|ASK=150.10|BID_QTY=500|ASK_QTY=500|TIF=DAY|QUOTE_ID=MM-AAPL-1
QUOTE_CANCEL|SYM=AAPL
```

3.12.3 Quote semantics

- only gateways configured with the `MARKET_MAKER` role may submit quotes
- `BID` must be strictly less than `ASK`
- both quantities must be positive
- a new quote for the same gateway and symbol replaces the previous active quote
- a quote creates two internal `LIMIT` orders, one bid and one ask

3.13 `KILL`

`KILL` triggers the kill switch for the current gateway.

3.13.1 Syntax

```
KILL
KILL|SYM=<symbol>
```

3.13.2 Examples

Cancel all eligible exposure for the gateway:

```
KILL
```

Cancel only for one symbol:

```
KILL|SYM=AAPL
```

3.13.3 Semantics

`KILL` applies to the **current authenticated gateway only**. It does not cancel other gateways' orders.

The engine performs these steps:

1. Validate that the gateway is currently allowed to submit risk actions
2. Cancel the gateway's active quote entry or entries
3. Cancel the gateway's remaining **resting non-quote orders**
4. Return a kill-switch acknowledgement with summary counts

More precisely:

- if no `SYM` is supplied, the engine removes **all active quotes** for that gateway from the internal quote index
- if `SYM` is supplied, it removes only the active quote for that `gateway + symbol` pair
- each removed quote entry then cancels its **bid leg** and **ask leg**
- after quote cancellation, the engine walks every order book and cancels every **resting** order where:
  - `order.gateway_id` matches the current gateway
  - the order is **not** a quote-origin order
  - the symbol matches `SYM` if a symbol filter was supplied

This means `KILL` cancels:

- active market-maker quote legs for the current gateway
- ordinary resting limit-style exposure for the current gateway
- any other gateway-owned resting child orders that are sitting in a book and are not quote-origin orders

This means `KILL` does **not**:

- cancel other gateways' orders
- reverse fills that already happened
- reach into already-completed orders that are no longer resting in a book
- count quotes as one unit; the acknowledgement counts **cancelled quote legs** separately from **cancelled non-quote orders**

If the gateway is not allowed to perform the action, the engine sends a reject acknowledgement instead of cancelling anything.

3.13.4 Acknowledgement behavior

The gateway receives a `KILL_ACK` message.

On success, the acknowledgement includes:

- `ORDERS` - number of cancelled non-quote resting orders
- `QUOTES` - number of cancelled quote legs

(`pm-alf-console` labels these `cancelled_orders`/`cancelled_quotes` in its own on-screen rendering; those are display labels, not wire field names.)

Important detail: `cancelled_quotes` counts **legs**, not quote records. One two-sided quote typically contributes up to **2** to this count.

3.13.5 Examples

If gateway `MM01` currently has:

- one active two-sided quote in `AAPL`
- one active two-sided quote in `MSFT`
- three resting non-quote orders in `AAPL`
- one resting non-quote order in `GOOG`

then:

```
KILL|SYM=AAPL
```

will cancel:

- the AAPL quote's bid and ask legs
- the three resting non-quote AAPL orders

but it will leave:

- the MSFT quote untouched
- the GOOG resting order untouched

while:

```
KILL
```

will cancel all of the above gateway-owned resting exposure across all symbols.

3.14 DC — drop-copy relay control

DC toggles whether this session relays the engine's drop-copy feed (DropCopyPublisher, ZMQ PUB on a separate socket from the main event bus) scoped to the session's own gateway ID.

3.14.1 Syntax

```
DC|STATE=ON
DC|STATE=OFF
```

STATE is required and must be exactly ON or OFF; any other value is rejected with ERR|CODE=INVALID_VALUE.

3.14.2 Semantics

Unlike the trading commands above, DC is **not forwarded to the engine as an order-entry message**. It only changes a local subscription on the gateway process (pm-alf-console or pm-alf-gwy): STATE=ON subscribes this session to drop-copy fill events for its own gateway_id; STATE=OFF unsubscribes. Disabled by default — each session opts in independently, and there is no config flag to force it on for every session.

Once enabled, fills for this gateway arrive **asynchronously** as unsolicited DC_FILL lines, in addition to (not instead of) the ordinary FILL message that every session already receives. DC_FILL is not itself a command a client sends — it is only ever sent by the gateway, so it is not part of the command grammar in this appendix; see ALF TCP Gateway → DC for its full field format.

3.14.3 Acknowledgement behavior

The gateway replies with DC_ACK|STATE=ON or DC_ACK|STATE=OFF confirming the new subscription state.

3.14.4 Examples

```
> DC|STATE=ON
DC_ACK|STATE=ON
...
DC_FILL|SEQ=42|ORDER_ID=ORD-001|SYMBOL=AAPL|FILL_QTY=100|FILL_PRICE=150.05|LIQUIDITY=TAKER
> DC|STATE=OFF
DC_ACK|STATE=OFF
```

Not FIX drop-copy, not a separate session

This is a same-connection relay, not a second FIX-style drop-copy session. See Drop Copy for the underlying feed this relays, and Drop-Copy TCP Gateway for a standalone external gateway (pm-dc-gwy) that exposes the same feed to clients that are not ALF sessions at all.

3.15 Informational gateway commands

These commands are part of the ALF command language accepted by `pm-alf-console`, but they are not order-entry messages in the strict sense.

3.15.1 STATUS

Purpose: quick local gateway/session summary.

```
STATUS
```

`STATUS` reports the current gateway ID, known symbols, cached order counts by lifecycle state, cached quote-leg counts, and position symbols. It is useful as a fast health check after connecting or after a burst of activity.

`STATUS` is not the detailed order-inspection command. Use `ORDERS` when you need order IDs, prices, quantities, remaining quantity, TIF, or per-order lifecycle state.

3.15.2 ORDERS

```
ORDERS
```

Purpose: detailed order inspection for the current gateway session.

`ORDERS` prints the gateway's order table, including order IDs, symbol, side, type, TIF, quantity, remaining quantity, price, current lifecycle status, and last update time.

3.15.3 POS

```
POS
POS|GW=<gateway_id>
```

Bare `POS` prints positions and P&L computed from fills seen by this gateway session's own local ledger — `pm-alf-console` only; `pm-alf-gwy` rejects it with `ERR|CODE=MISSING_FIELD|DETAIL=POS requires GW=<gateway_id>` (see the note above).

`POS|GW=<gateway_id>` asks the engine what *any* gateway is holding — typically a running `pm-mm-bot` instance, but any connected gateway_id works, including your own. Both clients send `system.position_request` for that `gateway_id` and render the engine's `system.position_snapshot. {gateway_id}` reply: net quantity and average cost per symbol, for every symbol that gateway has a non-zero position in. This is always a genuine engine round trip, in both clients — it does not use either client's local fill ledger. See ALF TCP Gateway → POS for the wire-level request/reply shape.

3.15.4 SYMBOLS

```
SYMBOLS
```

Requests the currently active symbols from the engine.

3.15.5 SESSION

```
SESSION
```

Requests the engine's current trading session state (`PRE_OPEN`, `OPENING_AUCTION`, `CONTINUOUS`, `CLOSING_AUCTION`, `CLOSED`) without waiting for the next unsolicited session-phase-change broadcast. Useful right after connecting, so a client can discover the current phase instead of only learning about it on the *next* transition.

The reply reuses the same message shape as the unsolicited session-state broadcast, with `PREV_STATE` left empty (a query has no previous state to report) and an added `SESSIONS_ENABLED` field:

```
SESSION|STATE=CONTINUOUS|PREV_STATE=|SESSIONS_ENABLED=TRUE
```

See `system.session_state_request` for the underlying engine message.

3.15.6 QBOOT

```
QBOOT
QBOOT|SYM=<symbol>
```

Requests active quote bootstrap state for the current gateway from the engine.

This is intended for MM startup/reconnect flows where quote legs may have been seeded by configuration before the gateway connected. The gateway prints a table of active quote bootstrap entries including quote id, symbol, state, bid/ask prices, and remaining quantities.

Use `QBOOT|SYM=<symbol>` when you only need one symbol.

3.15.7 QLEGS

```
QLEGS
QLEGS|SYM=<symbol>
QLEGS|SHOW=<ACTIVE|RECENT|ALL>
QLEGS|SYM=<symbol>|SHOW=<ACTIVE|RECENT|ALL>
```

Shows market-maker quote legs with explicit fill state, remaining size, and latest lifecycle status in one place. The command syntax, filtering, and output columns are identical across both ALF clients, but **the two clients answer it differently**:

- **`pm-alf-console`**: read-only, purely local. It renders from this session's own in-memory quote-leg projection, built by observing the `QUOTE_ACK`/`FILL`/`ORDER`/quote-status events this session has already received — it never sends a request to the engine for this command.
- **`pm-alf-gwy`**: forwards the command as `system.quote_legs_request` to the engine and renders the engine's authoritative reply — see ALF TCP Gateway → QLEGS for the wire-level request/response shape.

Neither variant sends modify or cancel actions to the engine.

Field behavior (same in both clients):

- `SYM` (optional): limit output to one symbol.
- `SHOW` (optional): controls filtering mode; defaults to `ACTIVE`.
  - `ACTIVE`: quote legs that are currently live or still have remaining size.
  - `RECENT`: legs belonging to a quote that has been inactivated (removed from the active index) — by cancel, kill switch, disconnect, halt, or replacement. A recent leg can still carry a non-zero remaining size; it is the quote, not the leg's size, that determines `RECENT` vs `ACTIVE`.
  - `ALL`: union of active and recent rows.

Typical output columns:

- `Symbol`, `Quote`, `Leg`, `Order`
- `Qty`, `Rem`, `Filled`, `Filled?`
- `Leg status`, `Quote status`, `Time`

Operational usage pattern:

1. Submit a quote.
2. Run `QLEGS|SYM=<symbol>` to confirm both legs are active.
3. After a fill/status event, run `QLEGS|SYM=<symbol>|SHOW=ALL`.
4. Determine which leg traded and whether the quote is still active before re-quoting.

Examples:

```
[MM_AAPL_01]> QLEGS
# show currently active quote legs across all symbols

[MM_AAPL_01]> QLEGS|SYM=AAPL
# show currently active quote legs for AAPL only

[MM_AAPL_01]> QLEGS|SYM=AAPL|SHOW=ALL
# show both active and recently completed/cancelled AAPL quote legs

[MM_AAPL_01]> QLEGS|SHOW=RECENT
# show recent completed quote legs across all symbols
```

Important notes:

- On `pm-alf-console`, the local view is rebuilt on reconnect from current order snapshots and new events, but only reflects what this session has observed since (re)connecting.
- On `pm-alf-gwy`, `QLEGS` always reflects the engine's current authoritative state, including a bounded, in-memory, per-gateway history of recently-inactivated quotes for the `RECENT`/`ALL` rows.
- Use `QBOOT` on either client when you need protocol-level startup bootstrap state for active quote slots owned by this gateway (unlike `QLEGS` on `pm-alf-console`, `QBOOT` always asks the engine).

3.15.8 `HELP`

```
HELP
```

Prints the gateway help text.

3.15.9 `EXIT` / `QUIT`

```
EXIT
QUIT
```

Closes this session's connection. `pm-alf-gwy` is a multi-client server: this only ends the current TCP session, it does not stop the gateway process itself. (For the single-session `pm-alf-console`, ending the one session is equivalent to exiting the program.)

3.16 Configuration reference

ALF gateway authorization and behavior are configured in the main engine configuration file (`engine_config.ya` under the `participants` list.

Path location:

- `engine_config.yaml` -> `participants` -> list of participant entries

All supported ALF gateway configuration fields are listed below.

Field	Type / allowed range	Default	Description
`participants[].id`	Non-empty string	None (required)	Gateway identity used by `pm-alf-console --id ...` and engine allowlist checks.
`participants[].description`	String	Empty string	Human-readable gateway description.
`participants[].role`	Enum: `TRADER`, `MARKET_MAKER`, `ADMIN`	`TRADER`	Participant role used for authorization/policy checks.
`participants[].disconnect_behaviour`	Enum: `CANCEL_QUOTES_ONLY`, `CANCEL_ALL`, `LEAVE_ALL`	`CANCEL_QUOTES_ONLY`	Engine behavior applied when that gateway disconnects.
`participants[].quote_refresh_policy`	Enum: `INACTIVATE_ON_ANY_FILL`, `INACTIVATE_ON_FULL_FILL`, `NEVER_INACTIVATE`	`INACTIVATE_ON_ANY_FILL`	Controls market-maker quote life-cycle after fills.
`participants[].enforce_mm_obligation`	Boolean (`true`/`false`)	`false`	Enables per-gateway market-maker obligation checks.
`participants[].mm_max_spread_ticks`	Integer, `> 0`	`20`	Maximum allowed quote spread for MM obligation checks.
`participants[].mm_min_qty`	Integer, `> 0`	`100`	Minimum quote size for MM obligation checks.
`participants[].mm_obligations`	Mapping of symbol -> obligation object	Empty mapping	Optional per-symbol MM overrides when configured.

Example:

```
participants:
  - id: TRADER01
    description: Human trader workstation
    role: TRADER
    disconnect_behaviour: CANCEL_ALL
    quote_refresh_policy: INACTIVATE_ON_ANY_FILL
    enforce_mm_obligation: false
    mm_max_spread_ticks: 20
    mm_min_qty: 100
```

3.17 What to watch out for during implementation

- Normalize command verbs, field names, symbols, and enum values to uppercase before validation; mixed-case input is common in real clients.

- Parse segments by splitting only on the first `=` in each `KEY=VALUE` token; extra `=` characters in values should not break parsing.
- Apply last-value-wins behavior for duplicate keys; do not reject a command only because a key appears twice.
- Do not trim field-level whitespace implicitly; only trim line boundaries, otherwise behavior diverges from gateway semantics.
- Treat `QTY` in `AMEND` as new total quantity (not delta) and reject values below already-filled quantity.
- Enforce OCO and combo field-shape constraints early (leg numbering, required leg fields) to avoid partial acceptance of malformed multi-leg commands.
- Keep a strict separation between user-facing ALF parsing and internal engine payload format; do not expose internal message details in ALF syntax.

3.18 Conformance notes

If you are writing another ALF producer, the most important exact behaviors are:

1. ALF is **not** FIX. It is a simpler pipe-delimited command language.
2. The command verb must be first; after that, field order is mostly free.
3. Parsed values are normalized to **uppercase**, including user labels.
4. Repeated fields use the **last value** seen.
5. Single-leg orders support all documented order types, including `TRAILING_STOP`.
6. `AMEND` applies only to resting `LIMIT` and `ICEBERG` orders.
7. OCO syntax can express stop and trailing-stop legs, but not iceberg legs.
8. Combo ALF syntax does **not** expose per-leg `STOP`, `TRAIL`, or `VISIBLE` fields, so some engine-level leg types are not practically expressible in ALF combo form.
9. `QBOOT` is the protocol-level way to discover active quote bootstrap state for the current gateway (optionally filtered by symbol).
10. `QLEGS` is a monitoring view for quote legs; use `SHOW=ALL` after fills to inspect both live and recently completed legs. It is the one command whose mechanics differ by client — local cache on `pm-alf-console`, engine round trip on `pm-alf-gwy` — see the QLEGS section.
11. `DC` is handled entirely gateway-side and is never forwarded to the engine as an order-entry message; `DC_FILL` is sent only by the gateway and is not part of the client-side command grammar.
12. `SESSION` is an on-demand engine round trip in both clients; its reply reuses the same message shape as the unsolicited session-state broadcast, with `PREV_STATE` empty and an added `SESSIONS_ENABLED` field.

3.19 See also

In this book:

- Messages — the ZeroMQ messages that the gateway emits after parsing ALF commands
- Drop Copy — the underlying feed `DC|STATE=ON` relays
- BALF Protocol Reference — binary encoding of the same order model
- CALF Protocol Reference — market-data feed protocol

Participant Guide:

- ALF Console — the ALF client that implements this protocol
- Order Types — semantic descriptions of every order type referenced here

Operator's Guide:

- Drop-Copy TCP Gateway — standalone gateway for external, non-ALF drop-copy clients

CHAPTER 4

BALF Protocol Reference

> **Status: Normative.** This appendix is the single source of truth for the BALF `1.0.0` wire contract as implemented by `pm-balf-gwy` (`balf_gwy/`). For an operational, tutorial-style guide see BALF TCP Gateway; for the gateway's configuration block see Engine Config Specification §6.2. The key words MUST, MUST NOT, SHOULD, and MAY are used per RFC 2119.

4.1 What BALF is

BALF stands for **Binary ALF**.

It is EduMatcher's fixed-width binary protocol for programmatic order entry through `pm-balf-gwy`. BALF carries the same trading intent as ALF, but it replaces human-readable text parsing with deterministic byte offsets.

BALF is designed for low-latency clients that want:

- fixed frame sizes
- no delimiter scanning
- no decimal string parsing on the hot path
- explicit message sequencing
- simple exact-byte reads

BALF is intentionally smaller and more predictable than text protocols:

```
NEW_ORDER is always 60 bytes on the wire
CANCEL_ORDER is always 24 bytes on the wire
EXECUTION_REPORT is always 64 bytes on the wire
```

This appendix is the **normative reference** for BALF `1.0.0`. If another tool, bot, or GUI wants to speak BALF, this document is the correct source to follow.

4.2 Scope & conformance

BALF is the **binary client protocol accepted by `pm-balf-gwy`**. The gateway translates BALF frames into the engine's internal ZeroMQ + JSON messages.

That distinction matters:

- **BALF** is what a client writes on the wire
- **engine messages** are what the gateway publishes internally

This appendix describes the BALF side, not the internal message bus.

4.2.1 What BALF does not change

- The matching engine itself.
- The set of supported instruments and single-leg order types.
- Gateway authentication rules. The same allowlist model is used.

4.2.2 Non-goals for BALF `1.0.0`

- Encryption at the protocol layer. Use TLS or a tunnel if needed.
- RDMA or kernel-bypass transport. BALF is defined over standard TCP.
- Combo and OCO wire encodings. These are future work, not part of `1.0.0`.
- Sequence recovery / resend request design. Gap detection exists, but recovery negotiation is not defined in `1.0.0`.
- Multicast market data. A future BDMD protocol may cover that separately.

4.3 Wire format

4.3.1 Connection model

BALF uses one TCP connection per gateway session.

A session starts with `LOGON` and ends with `LOGOUT` or a disconnect. Each TCP connection is logically bound to one gateway ID.

4.3.2 Byte order

All multi-byte numeric fields are **little-endian**.

This is an explicit BALF `1.0.0` convention and intentionally differs from big-endian exchange protocols such as ITCH / OUCH.

4.3.3 Frame structure

Every BALF frame has:

- an 8-byte header
- a fixed-size body determined only by `msg_type`

There is no length prefix. The receiver reads the header first, looks up the expected frame size from the message type, then reads exactly the remaining bytes.

If `msg_type` is unknown, the receiver treats the frame as a protocol error and should close the connection.

4.3.4 Flag semantics

In BALF `1.0.0`, all header flag bits must be zero.

The `RETRANSMIT` bit is reserved for a future recovery extension and is not active in `1.0.0`.

Reserved fields in all message bodies must also be zero when sent and should be validated as zero when received.

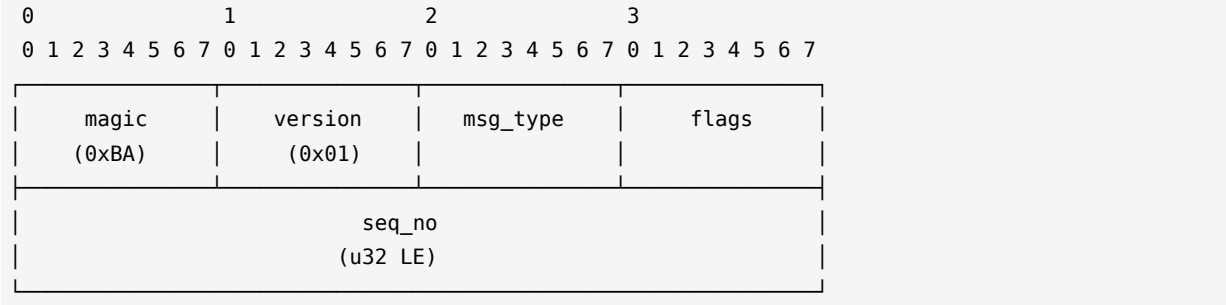
4.3.5 Identifier rules

- `gateway_id` is a 16-byte ASCII field, zero-padded on the right. The gateway uppercases both `gateway_id` (at `LOGON`) and `symbol` (at `NEW_ORDER`) after decoding, so a lowercase value is silently normalized rather than rejected or matched case-sensitively.
- `client_order_id` is a client-assigned `u64`.
- `order_id` is a gateway-assigned session-scoped `u64`.
- `0` is reserved and must not be used as a valid accepted `order_id`.

4.4 Frame format

4.4.1 Header layout

Every BALF message begins with the same 8-byte header:



4.4.2 Header fields

Offset	Field	Type	Description
0	`magic`	`u8`	Always `0xBA`. Reject frames where this byte is wrong.
1	`version`	`u8`	Protocol version. BALF `1.0.0` uses `1`.
2	`msg_type`	`u8`	Message type code, see the tables below.
3	`flags`	`u8`	Must be `0` in BALF `1.0.0`. Reserved for future use.
4-7	`seq_no`	`u32 LE`	Monotonically increasing sequence number.

4.4.3 Sequence numbering rules

- Sequence numbers are tracked separately in each direction.
- `LOGON` and `LOGON_ACK` carry `seq_no = 0`.
- The first non-LOGON message in each direction starts at `1`.
- Sequence numbers increment by `1` per message.
- Sequence numbers wrap to `1` at `UINT32_MAX`.
- A gap in the inbound sequence number is a protocol error.
- BALF `1.0.0` does not define a resend request or replay recovery protocol.

4.5 Scalar encodings

4.5.1 Price encoding

All prices and price-like offsets are encoded as signed 64-bit integers scaled by `10^8`.

```
wire_value = round(display_price * 100_000_000)
```

Examples:

Display value	Wire value
`150.25`	`15_025_000_000`
`0.0001`	`10_000`
`-2.00`	`-200_000_000`

A value of `0` means the field is not used for that order type.

4.5.2 Symbol encoding

Symbols are stored as 8 bytes, zero-padded ASCII, left-aligned.

Example:

```
"AAPL" -> 41 50 50 4C 00 00 00 00
```

Symbols longer than 8 ASCII characters are not supported in BALF `1.0.0`.

4.5.3 Gateway ID encoding

Gateway IDs are stored as 16 bytes, zero-padded ASCII, left-aligned.

Example:

```
"TRADER01" -> 54 52 41 44 45 52 30 31 00 00 00 00 00 00 00 00
```

4.5.4 Order ID encoding

BALF uses a compact `u64` session-scoped order ID on the wire.

- the gateway allocates the ID when the order is accepted
- the gateway maps that ID to the engine's internal UUID or matching-engine key
- the `order_id` is echoed in cancel, amend, and execution messages
- `0` is reserved and invalid

4.6 Message definitions

4.6.1 Summary table

Code	Name	Direction	Body bytes	Total frame bytes
`0x01`	`LOGON`	Client -> Server	24	32
`0x02`	`LOGON_ACK`	Server -> Client	84	92
`0x10`	`NEW_ORDER`	Client -> Server	52	60

Code	Name	Direction	Body bytes	Total frame bytes
0x11	ORDER_ACK	Server -> Client	52	60
0x12	CANCEL_ORDER	Client -> Server	16	24
0x13	CANCEL_ACK	Server -> Client	24	32
0x14	AMEND_ORDER	Client -> Server	36	44
0x15	AMEND_ACK	Server -> Client	40	48
0x20	EXECUTION_REPORT	Server -> Client	56	64
0x21	STOP_TRIGGERED	Server -> Client	48	56
0x30	HEARTBEAT	Bidirectional	8	16
0x31	HEARTBEAT_ACK	Bidirectional	8	16
0x40	LOGOUT	Client -> Server	0	8

All sizes include the 8-byte header.

4.6.2 LOGON (0x01) - Client -> Server

Sent immediately after the TCP connection is established, before any orders. seq_no is 0.

Body (24 bytes):

```
Offset  0  |  gateway_id     |  u8[16]  |  Zero-padded ASCII gateway ID
Offset 16  |  proto_version   |  u8      |  Must be 1
Offset 17  |  _reserved       |  u8[7]   |  Must be zero
```

4.6.3 LOGON_ACK (0x02) - Server -> Client

Sent in response to LOGON. seq_no is 0 on this message; normal sequencing begins from 1 on subsequent messages.

Body (84 bytes):

```
Offset  0  |  gateway_id     |  u8[16]  |  Echoed gateway ID
Offset 16  |  accepted       |  u8      |  1 = accepted, 0 = rejected
Offset 17  |  reject_code    |  u8      |  See reject codes below; 0 if accepted
Offset 18  |  msg_len        |  u8      |  Byte length of the meaningful part of msg[]
Offset 19  |  _pad           |  u8      |  Reserved, zero
Offset 20  |  msg            |  u8[64]  |  Human-readable description or rejection reason
```

Reject codes:

Code	Meaning
0x00	Accepted
0x01	Gateway ID not configured in engine
0x02	Gateway ID already connected
0x03	Protocol version mismatch
0xFF	Other (see msg field)

4.6.4 `NEW_ORDER` (0x10) - Client -> Server

Body (52 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  Client-assigned reference, echoed in all responses
Offset  8  |  symbol           |  u8[8]   |  Zero-padded ASCII symbol
Offset 16  |  price            |  i64 LE  |  Limit price x 10^8; 0 for MARKET/STOP orders
Offset 24  |  stop_price       |  i64 LE  |  Stop trigger x 10^8; 0 if unused
Offset 32  |  trail_offset     |  i64 LE  |  Trailing offset x 10^8; 0 if unused
Offset 40  |  quantity         |  u32 LE  |  Order quantity
Offset 44  |  visible_qty      |  u32 LE  |  ICEBERG peak size; 0 for all other types
Offset 48  |  side             |  u8      |  1 = BUY, 2 = SELL
Offset 49  |  order_type       |  u8      |  See order type codes below
Offset 50  |  tif              |  u8      |  See TIF codes below
Offset 51  |  smp              |  u8      |  See SMP codes below
```

Order type codes:

Code	ALF equivalent	Required price fields
`0x01`	`MARKET`	none (price = 0)
`0x02`	`LIMIT`	`price`
`0x03`	`IOC`	`price`
`0x04`	`FOK`	`price`
`0x05`	`STOP`	`stop_price`
`0x06`	`STOP_LIMIT`	`stop_price` and `price`
`0x07`	`ICEBERG`	`price` and `visible_qty`
`0x08`	`TRAILING_STOP`	`trail_offset`; optionally `stop_price`

A price field is "absent" when it is zero. Every other price field must be zero: a non-zero field the type does not use (a `trail_offset` on a `LIMIT`, say) is refused with `RC_INVALID_FIELD` rather than ignored. This is the same per-type rule REST, ALF and the engine apply.

TIF codes:

Code	ALF equivalent
`0x01`	`DAY`
`0x02`	`GTC`
`0x03`	`ATO`
`0x04`	`ATC`

SMP codes:

Code	ALF equivalent
`0x00`	`NONE`
`0x01`	`CANCEL_AGGRESSOR`
`0x02`	`CANCEL_RESTING`
`0x03`	`CANCEL_BOTH`

4.6.5 `ORDER_ACK` (0x11) - Server -> Client

Sent for every `NEW_ORDER`. Arrives before any `EXECUTION_REPORT` for the same order.

Body (52 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  Echoed from NEW_ORDER
Offset  8  |  order_id         |  u64 LE  |  Session-scoped BALF order ID; 0 if rejected
Offset 16  |  timestamp_ns     |  u64 LE  |  Nanoseconds since Unix epoch (engine receive time)
Offset 24  |  accepted         |  u8      |  1 = accepted, 0 = rejected
Offset 25  |  reject_code      |  u8      |  See reject codes below; 0 if accepted
Offset 26  |  reason_len       |  u8      |  Length of meaningful bytes in reason[]
Offset 27  |  reason           |  u8[25]  |  Rejection reason string (ASCII); zeros if accepted
```

Reject codes:

Code	Constant	Meaning
`0x00`	`RC_ACCEPTED`	Accepted
`0x01`	`RC_GW_NOT_CONFIGURED`	Gateway ID not configured in the engine
`0x02`	`RC_GW_NOT_CONNECTED`	Gateway not connected
`0x03`	`RC_SYMBOL_NOT_CONFIGURED`	Symbol not configured
`0x04`	`RC_MARKET_CLOSED`	Market closed
`0x05`	`RC_ATO_OUTSIDE_OPENING`	ATO order outside the opening auction
`0x06`	`RC_ATC_OUTSIDE_CLOSING`	ATC order outside the closing auction
`0x07`	`RC_HALT_REJECTION`	Symbol halted, by a circuit breaker or by an administrator — BALF has one code for both
`0x08`	`RC_PHASE_REJECTION`	Rejected by the current session phase
`0x09`	`RC_TRAILING_STOP_NO_PRICE`	TRAILING_STOP with no prior trade price
`0x0A`	`RC_INSUFFICIENT_LIQUIDITY`	FOK could not be filled in full
`0x0B`	`RC_PRICE_COLLAR`	Rejected by a price collar
`0x0C`	`RC_INVALID_FIELD`	Invalid field — bad quantity, a price field the order type needs and lacks (`"LIMIT requires price"`) or does not use (`"LIMIT forbids trail_offset"`), or a price that is not on the instrument's tick grid (`reason` = `"price off tick grid"`)
`0xFF`	`RC_OTHER`	Other; inspect the `reason` string

Prices must land on the tick grid

BALF carries prices as `i64` fixed point at scale 1e8, so a client can express values far finer than any instrument's tick size. A price that is not an exact multiple of the symbol's tick is rejected with `RC_INVALID_FIELD` rather than rounded — rounding would fill you at a price you never sent. `tick_decimals` per symbol comes from the reference data.

4.6.6 `CANCEL_ORDER` (0x12) - Client -> Server

Body (16 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  New client ref for this cancel request
Offset  8  |  order_id         |  u64 LE  |  Session-scoped BALF order ID to cancel
```

4.6.7 `CANCEL_ACK` (0x13) - Server -> Client

Body (24 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  Echoed from CANCEL_ORDER
Offset  8  |  order_id         |  u64 LE  |  Order being cancelled
Offset 16  |  accepted         |  u8      |  1 = cancelled, 0 = rejected
Offset 17  |  cancel_reason    |  u8      |  0 = client request, 255 = system-originated (SMP, session-end, expiry)
Offset 18  |  _reserved        |  u8[6]   |  Must be zero
```

4.6.8 `AMEND_ORDER` (0x14) - Client -> Server

Amends price and/or quantity of a resting LIMIT or ICEBERG order. At least one of the `amend_flags` bits must be set.

Body (36 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  New client ref for this amend request
Offset  8  |  order_id         |  u64 LE  |  Session-scoped BALF order ID to amend
Offset 16  |  new_price        |  i64 LE  |  New limit price x 10^8; ignored if bit 0 of amend_flags is clear
Offset 24  |  new_quantity     |  u32 LE  |  New total quantity; ignored if bit 1 of amend_flags is clear
Offset 28  |  amend_flags      |  u8      |  Bit 0 = price changed, bit 1 = quantity changed
Offset 29  |  _reserved        |  u8[7]   |  Must be zero
```

4.6.9 `AMEND_ACK` (0x15) - Server -> Client

Body (40 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  Echoed from AMEND_ORDER
Offset  8  |  order_id         |  u64 LE  |  Amended order
Offset 16  |  new_price        |  i64 LE  |  Price after amendment x 10^8
Offset 24  |  new_quantity     |  u32 LE  |  Total quantity after amendment
Offset 28  |  remaining_qty    |  u32 LE  |  Unfilled quantity after amendment
Offset 32  |  accepted         |  u8      |  1 = accepted, 0 = rejected
Offset 33  |  priority_reset   |  u8      |  1 = order lost time priority; 0 = priority preserved
Offset 34  |  _reserved        |  u8[6]   |  Must be zero
```

4.6.10 `EXECUTION_REPORT` (0x20) - Server -> Client

Sent for every partial or full fill. Both sides of a match (aggressor and resting order) receive their own `EXECUTION_REPORT`.

Body (56 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  Echoed from the original NEW_ORDER
Offset  8  |  order_id         |  u64 LE  |  Filled order ID
Offset 16  |  fill_price       |  i64 LE  |  Execution price x 10^8
Offset 24  |  fill_qty         |  u32 LE  |  Quantity matched in this event
Offset 28  |  remaining_qty    |  u32 LE  |  Unfilled quantity after this fill
Offset 32  |  timestamp_ns     |  u64 LE  |  Trade timestamp - nanoseconds since Unix epoch
Offset 40  |  symbol           |  u8[8]   |  Symbol (for convenience; matches original order)
Offset 48  |  side             |  u8      |  1 = BUY, 2 = SELL
Offset 49  |  status           |  u8      |  1 = PARTIAL, 2 = FILLED
Offset 50  |  _reserved        |  u8[6]   |  Must be zero
```

4.6.11 `STOP_TRIGGERED` (0x21) - Server -> Client

A stop, stop-limit or trailing stop of this session has fired. Sent before the order's own `EXECUTION_REPORT`s.

Body (48 bytes):

```
Offset  0  |  client_order_id  |  u64 LE  |  Echoed from the original NEW_ORDER
Offset  8  |  order_id         |  u64 LE  |  The stop's order ID
Offset 16  |  stop_price       |  i64 LE  |  Trigger price it fired at x 10^8 (a trailing stop's ratcheted on
Offset 24  |  trigger_price    |  i64 LE  |  Price of the trade that reached it x 10^8; 0 after an engine res
Offset 32  |  symbol           |  u8[8]   |  Symbol
Offset 40  |  order_type       |  u8      |  5 = STOP, 6 = STOP_LIMIT, 8 = TRAILING_STOP (NEW_ORDER's codes)
Offset 41  |  becomes          |  u8      |  1 = MARKET, 2 = LIMIT
Offset 42  |  _reserved        |  u8[6]   |  Must be zero
```

4.6.12 `HEARTBEAT` (0x30) - Bidirectional

Either side may send a heartbeat at any time. The recipient must respond with `HEARTBEAT_ACK`. A session is considered dead if no traffic (including heartbeats) arrives within 5 seconds (configurable). The default send interval is 1 second.

If the timeout is exceeded, the connection is considered stale and should be closed.

Body (8 bytes):

```
Offset  0  |  send_time_ns  |  u64 LE  |  Sender's wall-clock time in nanoseconds since Unix epoch
```

4.6.13 `HEARTBEAT_ACK` (0x31) - Bidirectional

Body (8 bytes):

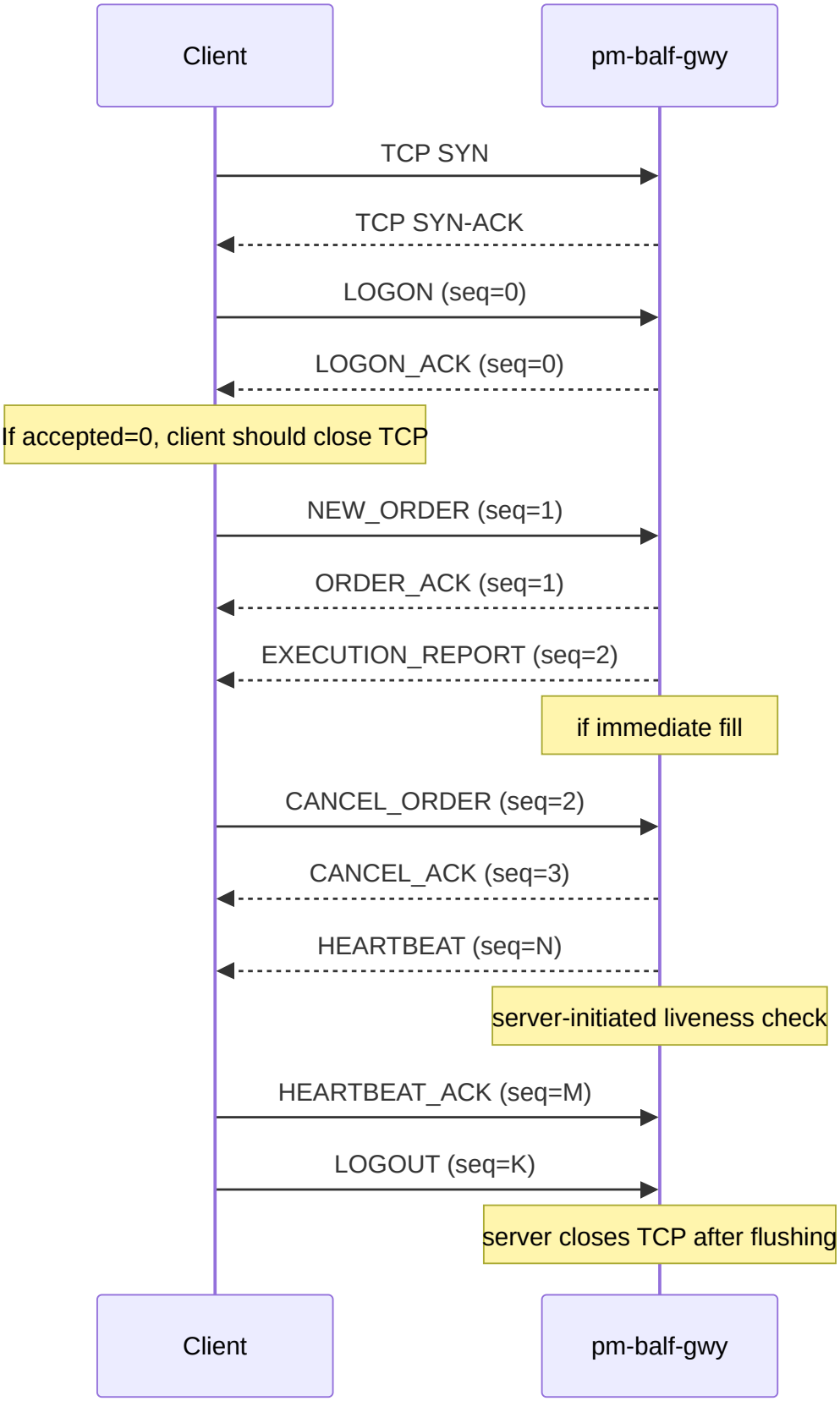
```
Offset  0  |  orig_send_time_ns  |  u64 LE  |  Echo of the send_time_ns from the HEARTBEAT
```

4.6.14 `LOGOUT` (0x40) - Client -> Server

Graceful disconnect. No body. After sending `LOGOUT`, the client must not send any further messages and should close the TCP connection. The server will flush any pending outbound messages and then close the connection on its side.

Body: none (total frame = 8 bytes, header only).

4.7 Session lifecycle



4.7.1 Sequence number rules

- Separate, independent sequence counters for each direction.

- Both counters start at `1` on the first non-LOGON message.
- `LOGON` and `LOGON_ACK` carry seq_no `0` by convention and are not part of the numbered sequence.
- Sequence numbers increment by `1` per message. They wrap to `1` (not `0`) at `UINT32_MAX`.
- A gap in the inbound sequence number is a protocol error; the receiver should log it and may close the connection.
- BALF `1.0.0` does not define a resend/recovery request. On detected gaps, receivers should treat the session as out-of-sync and reconnect.
- The `RETRANSMIT` flag is reserved for a future recovery extension and must be `0` in BALF `1.0.0`.

4.8 Configuration reference

BALF settings are part of the main engine configuration file (`engine_config.yaml`). BALF has no config block of its own for gateway identity — it reuses `participants` (the same allowlist ALF uses, see ALF TCP Gateway) for identity, role, and `disconnect_behaviour`, and adds only its own TCP listener/runtime parameters under `balf_gateway`.

Path locations:

- `engine_config.yaml -> participants` (shared with ALF)
- `engine_config.yaml -> balf_gateway`

4.8.1 `balf_gateway` fields

Field	Type / allowed range	Default	Description
`balf_gateway.bind_address`	IP/host bind string	`0.0.0.0` (common)	Local interface address for BALF TCP listening.
`balf_gateway.port`	Integer, `1..65535`	`5560`	BALF listen port.
`balf_gateway.heartbeat_interval_sec`	Integer, `>0`	`1`	Default heartbeat send interval.
`balf_gateway.heartbeat_timeout_sec`	Integer, `>0`	`5`	Liveness timeout before session is considered dead and closed.

Example:

```
balf_gateway:
  bind_address: "0.0.0.0"
  port: 5560
  heartbeat_interval_sec: 1
  heartbeat_timeout_sec: 5
```

The gateway ID allowlist is still enforced by the engine. BALF IDs do not need to match ALF IDs, but both use the same allowlist policy.

4.9 What to watch out for during implementation

- Read exactly 8 bytes for the header first, then read the exact remaining bytes for that `msg_type`; partial reads are normal on TCP.

- Reject unknown `msg_type`, bad `magic`, wrong `version`, and non-zero reserved fields deterministically; silent tolerance creates interoperability bugs.
- Keep separate inbound and outbound sequence counters; never mix directions.
- Treat `LOGON` and `LOGON_ACK` as `seq_no = 0` special cases and start normal sequencing at `1` afterward.
- Enforce little-endian decoding/encoding for all multi-byte numeric fields; endianness mistakes are the most common BALF parser defect.
- Respect fixed-width ASCII rules for `symbol` and `gateway_id` with zero padding; trim only trailing zero bytes on decode.
- Treat `order_id = 0` as invalid/reserved and do not allow it as an accepted live order identifier.
- On detected sequence gaps, treat the session as out-of-sync and reconnect; BALF `1.0.0` does not define resend negotiation.

4.10 Practical parsing notes

If you are writing another BALF client, the most important exact behaviors are:

1. BALF is **not** ALF. It is a binary wire protocol with fixed-width frames.
2. The header is always 8 bytes.
3. All multi-byte numeric values are little-endian.
4. `flags` must be zero in BALF `1.0.0`.
5. `order_id` is a compact session-scoped `u64`.
6. `LOGON` and `LOGON_ACK` use `seq_no = 0`.
7. Sequence gaps are protocol errors.
8. Combo, OCO, and recovery wire features are not part of `1.0.0`.
9. Heartbeats are bidirectional and use their own fixed 8-byte body.
10. The gateway generates the final engine mapping; clients do not send UUIDs.
11. The protocol uses little-endian byte order for all multi-byte numeric fields.

4.11 See also

In this book:

- External Protocols Overview — protocol comparison and selection guide

Operator's Guide:

- BALF Gateway — Binary Order Entry — setup, configuration, session lifecycle, Python client example, and troubleshooting
- ALF Gateway — Order Entry — text-protocol alternative when binary parsing is not required

Reference Manual:

- Configuring `pm-balf-gwy` — `balf_gateway:` section reference

CHAPTER 5

CALF Protocol Reference

Status: Normative. This appendix is the single source of truth for the CALF `1.0.0` wire contract as implemented by `pm-md-gwy` (`md_gateway/`). For an operational, tutorial-style guide see Market Data Feed (CALF); for the gateway's configuration block see Engine Config Specification §6.3. The key words MUST, MUST NOT, SHOULD, and MAY are used per RFC 2119.

5.1 What CALF is

CALF stands for **Channel ALF**.

CALF is EduMatcher's text market-data protocol. It is designed for educational clarity and bot usability: human-readable on the wire, easy to debug in a terminal, and strict enough to support deterministic clients.

CALF complements the other application protocols:

Protocol	Purpose
ALF	Text order entry (interactive)
BALF	Binary order entry (low-latency programmatic)
CALF	Channelized text market data

This appendix is the **normative reference** for CALF `1.0.0` semantics.

5.2 Scope & conformance

CALF is the external market-data protocol exposed by `pm-md-gwy`. The gateway subscribes to internal engine PUB topics and translates them into CALF lines for TCP clients.

This appendix specifies the **client-visible CALF protocol**. It does not specify internal engine message schemas beyond what is needed to explain CALF behavior.

5.2.1 Supported in CALF `1.0.0`

- top-of-book updates (`MD`) by symbol
- trade prints (`TRADE`) by symbol
- state transitions (`STATE`) for session-wide and symbol-level changes

- index level updates (`IDX`) from `pm-index`
- aggregated multi-level order book updates (`DEPTH`) — Level 2, not order-by-order
- auction uncross results (`AUCTION`) — equilibrium price, matched quantity, and imbalance for open/close auctions and circuit-breaker reopening auctions
- full circuit-breaker halt/resume detail (`CB`) — trigger price, reference price, ladder level, auto-resume time, and halt cause, alongside the coarse `STATE` transition
- point-in-time stream baselines (`SNAP`) for `TOP`, `STATE`, `INDEX`, `DEPTH`, and `CB`
- per-stream sequence numbers on `(CH, SYM)`
- `SYM=*` wildcard subscriptions for `STATE`, `TOP`, `TRADE`, and `AUCTION`
- bounded replay on reconnect (`RESUME` + `LASTSEQ`)
- heartbeat and liveness signaling
- gateway capability advertisement via `WELCOME|CH_SUPPORTED=`

5.2.2 Out of scope in CALF `1.0.0`

- full order-by-order (Level 3) market data — `DEPTH` is Level 2 (aggregated per price level), never per-order
- `SYM=*` for `INDEX`, `DEPTH`, or `CB`
- multicast / UDP transport
- entitlement matrix per field
- durable historical replay from disk
- protocol-layer authentication token

5.3 Transport and session model

Property	Value
Transport	TCP
Default port	`5570`
Encoding	UTF-8 line protocol
Delimiter	`\n`
Max line length	4096 bytes including newline

A CALF client connection is long-lived.

- Client must send `HELLO` within 5 seconds of TCP connect.
- Gateway replies with `WELCOME` on success.
- Client may then send `SUB`, `RESUME`, `UNSUB`, `SYMBOLS`, `PING`, and `EXIT`.
- Gateway streams `SNAP`, `MD`, `TRADE`, `STATE`, `IDX`, `DEPTH`, `AUCTION`, `CB`, `HB`, and `ERR`.

If no `HELLO` is received within 5 seconds, the gateway sends `EXIT|REASON=auth_timeout` and closes the socket. Once the gateway has sent the line that ends a session (`ERR|CODE=AUTH_REQUIRED`, for example), it ignores anything more the client sends — a `HELLO` pipelined behind a refused first line is not answered.

5.4 Wire format

5.4.1 Line structure

Every CALF message is one line:

```
<MSGTYPE>|KEY=VALUE|KEY=VALUE|...\n
```

`MSGTYPE` is the first token and is always uppercase ASCII.

Examples:

```
HELLO|CLIENT=bot01|PROTO=CALF1
TRADE|CH=TRADE|SYM=AAPL|SEQ=809|TS=2026-06-07T10:16:00.141Z|TRADE_ID=000042-000000123|RUN_SEQ=42|PX=150.12|QT
```

5.4.2 Parsing behavior

- Messages are delimited by newline (`\n`).
- `\r\n` from clients is accepted for robustness.
- Field order after `MSGTYPE` is not significant.
- Unknown keys are ignored unless needed for validating a specific message.
- Duplicate keys: last occurrence wins.
- Empty lines are invalid and may result in `ERR|CODE=BAD_MESSAGE`.

5.4.3 TCP stream requirement

TCP is a byte stream, not a message queue.

A receiver must buffer bytes and split by newline. A single `recv()` may contain half a line, one full line, or many lines.

5.5 Field conventions

5.5.1 Reserved keys

Key	Meaning
`CH`	Logical channel (`TOP`, `TRADE`, `STATE`, `INDEX`, `DEPTH`, `AUCTION`, `CB`)
`SYM`	Symbol, index id, or `*` where allowed
`SEQ`	Sequence number for one `(CH, SYM)` stream
`TS`	UTC ISO-8601 timestamp with milliseconds

5.5.2 Wire value types

Type	Wire representation	Example
Decimal price	Text decimal	`150.25`
Integer	Base-10 text integer	`1200`
Boolean flag	`0` or `1`	`EXC=1`
Timestamp	UTC ISO-8601 ms	`2026-06-07T10:15:23.411Z`

Optional fields are omitted when not present. Empty required values are invalid. One optional field pair carries meaning when empty rather than being malformed: `MD`'s `BID/ASK`, where an empty value marks that book side as withdrawn — see **Withdrawal of a book side** under `MD`.

5.6 Channel model

CALF groups market data into logical channels.

Channel	Description	SYM=* allowed?
TOP	Best bid/ask updates and snapshots	Yes
TRADE	Trade prints	Yes
STATE	Session or symbol state transitions	Yes
INDEX	Index level updates	No
DEPTH	Aggregated multi-level order book (Level 2)	No
AUCTION	Auction uncross results (equilibrium price, imbalance)	Yes
CB	Circuit-breaker halt/resume detail (trigger price, level, …)	No

SNAP is a message type, not a channel.

A client does not subscribe to SNAP directly. The gateway auto-sends SNAP for new SUB requests on TOP, STATE, INDEX, DEPTH, and CB. TRADE and AUCTION have no baseline SNAP — only future events are delivered for those two channels, since neither has a persistent "current value" to snapshot.

SYM=* for TOP does not produce a single SNAP|SYM=*. Top-of-book has no meaningful "wildcard" value, so the gateway instead sends one real SNAP per currently known symbol, then live MD for any symbol — including ones that become known only after the SUB — via the same wildcard subscription entry.

5.6.1 Subscription rules

- SUB may include multiple channels and symbols separated by commas.
- A multi-value SUB applies to the Cartesian product of channels and symbols.
- SYM=* is valid when the channel set is a subset of {STATE, TOP, TRADE, AUCTION}, in any combination. SYM=* combined with INDEX, DEPTH, or CB — alone or mixed with any other channel in the same SUB — is invalid.
- A wildcard subscription (SYM=*) counts as exactly one entry toward max_symbols_per_client, not one entry per known symbol.
- Re-subscribing an already active pair is idempotent.
- Maximum symbols per client are enforced by gateway config.
- If any requested (CH,SYM) pair is invalid, the gateway rejects the SUB request with ERR and leaves existing subscriptions unchanged.
- For non-INDEX channels, a non-wildcard SYM is validated against the gateway's known instrument list (populated from the engine's configured symbols at startup); an unrecognized symbol returns ERR|CODE=INVALID_SYMBOL. INDEX is exempt from this check — index ids live in a separate namespace from tradable instrument symbols, so any non-empty id is accepted at SUB time regardless of whether a matching index is actually configured. AUCTION and CB are not exempt: a non-wildcard symbol for either is checked against the same known-instrument list as TOP/TRADE/STATE/DEPTH.
- If the gateway's known-symbol list itself is empty (for example, engine config failed to load additional symbol metadata at gateway startup), known-symbol validation is skipped entirely and any non-wildcard symbol is accepted; this is a permissive fallback, not a documented steady-state mode, and operators should treat an empty known-symbol list as a configuration problem to fix rather than relied-upon behavior.

`DEPTH` and `CB` disallow `SYM=*` deliberately, not as an oversight: `DEPTH` messages carry up to `2 x depth_levels` price levels each, so a wildcard subscription could multiply one client's outbound bandwidth by the entire symbol count; `CB` halts/resumes are rare, per-symbol operator-relevant events rather than a firehose use case. `AUCTION`, by contrast, allows `SYM=*`: auction events are extremely low frequency (at most a handful per symbol per trading day), so a wildcard subscription poses none of `DEPTH`'s bandwidth risk, and — like `TRADE` — `AUCTION` has no baseline `SNAP` to burst per symbol on subscribe.

```
SUB|CH=TOP,TRADE|SYM=AAPL,MSFT
SUB|CH=STATE|SYM=*
SUB|CH=TRADE|SYM=*
SUB|CH=TOP,TRADE,STATE|SYM=*
SUB|CH=DEPTH|SYM=AAPL
SUB|CH=AUCTION|SYM=*
SUB|CH=CB|SYM=AAPL
```

5.7 Message catalog

5.7.1 Session control messages

Message	Direction	Purpose
`HELLO`	Client -> Gateway	Start session
`WELCOME`	Gateway -> Client	Confirm session and advertise parameters
`SUB`	Client -> Gateway	Add subscriptions
`RESUME`	Client -> Gateway	Replay one stream from a known sequence
`UNSUB`	Client -> Gateway	Remove subscriptions
`SYMBOLS`	Both directions	Request, and reply with, the instrument universe
`PING`	Client -> Gateway	Liveness probe
`PONG`	Gateway -> Client	Probe reply
`HB`	Gateway -> Client	Heartbeat when quiet
`ERR`	Gateway -> Client	Protocol or flow error
`EXIT`	Both	Clean disconnect; from the gateway, carries `REASON`

5.7.2 Market-data messages

Message	Direction	Purpose
`SNAP`	Gateway -> Client	Point-in-time baseline for one stream
`MD`	Gateway -> Client	Incremental top-of-book update
`INDIC`	Gateway -> Client	Indicative auction uncross, during a call phase
`TRADE`	Gateway -> Client	Trade print
`STATE`	Gateway -> Client	Session/symbol state transition
`IDX`	Gateway -> Client	Index level update
`DEPTH`	Gateway -> Client	Incremental multi-level order book update

Message	Direction	Purpose
`AUCTION`	Gateway -> Client	Auction uncross result
`CB`	Gateway -> Client	Circuit-breaker halt/resume detail

5.8 Message definitions

5.8.1 `HELLO`

Direction: Client -> Gateway

Purpose: Session handshake.

Response: `WELCOME` on successful handshake, or `ERR|CODE=AUTH_REQUIRED` (connection closed) if `CLIENT`/`PROTO` fail validation — the same code `pm-ralf-gwy` and `pm-dc-gwy` use.

Field	Req	Description
`CLIENT`	Yes	Client ID (ASCII, max 32 chars)
`PROTO`	Yes	Must be `CALF1`

Validation rules:

- Messages other than `HELLO` sent before successful handshake receive `ERR|CODE=AUTH_REQUIRED`.
- `HELLO` is only accepted while a session is unauthenticated. A second `HELLO` on an established session receives `ERR|CODE=BAD_MESSAGE`.

```
HELLO|CLIENT=bot01|PROTO=CALF1
```

> **Changed:** replay is no longer requested through a `RESUME=1` flag on `HELLO`. Because `HELLO` is only ever processed once per connection, that form could resume a single stream and no more — leaving any client that follows several streams, which is most of them, unable to recover the rest after a reconnect. Replay now has its own repeatable `RESUME` command, sent after the handshake.

5.8.2 `WELCOME`

Direction: Gateway -> Client

Field	Req	Description
`PROTO`	Yes	Echoes negotiated protocol
`GW`	Yes	Gateway instance name
`HBINT`	Yes	Heartbeat interval in seconds
`REPLAY`	Yes	Replay window in seconds

Field	Req	Description
SYMBOLS	No	Comma-separated snapshot of instrument symbols known to the gateway at connect time. Omitted when the gateway has no known symbols yet. The known-symbol set can grow after WELCOME is sent, as new book.{SYMBOL}/trade events arrive from the engine — this field is a point-in-time snapshot, not a fixed universe, and a symbol absent here may still become subscribable later without a new WELCOME.
REF	No	Per-symbol reference data as SYM:DEC tuples, where DEC is the instrument's display precision (tick_decimals). Covers exactly the symbols listed in SYMBOLS=, and is omitted whenever that field is. Absent entirely from gateways predating CALF 1.1.0 — a client detects support by its presence, not by PROTO, and falls back to 2 decimals when it is missing.
CH_SUPPORTED	No	Comma-separated list of channels this gateway build supports. Present on every CALF 1.0.0+ gateway; omitted entirely by earlier gateways. A client uses its presence — not the PROTO value, which does not change — to detect whether DEPTH/INDEX and the SYM=* wildcard extension are available.

```
WELCOME|PROTO=CALF1|GW=md-gwy01|HBINT=1|REPLAY=30|SYMBOLS=AAPL,MSFT|CH_SUPPORTED=AUCTION,CB,DEPTH,INDEX,STATE
```

A client that receives no CH_SUPPORTED field must assume only TOP, TRADE, and STATE are available and must not rely on SYM=* for TOP/ TRADE, INDEX, or DEPTH without first probing with SUB and handling a possible ERR|CODE=INVALID_CHANNEL/INVALID_SYMBOL response.

5.8.3 SUB

Direction: Client -> Gateway

Field	Req	Description
CH	Yes	Comma-separated channels
SYM	Yes	Comma-separated symbols (* only when CH is a subset of {STATE, TOP, TRADE, AUCTION})

Response semantics:

- No explicit ACK.
- New TOP/STATE/INDEX/DEPTH/CB subscriptions trigger SNAP. A wildcard TOP subscription (SYM=*) triggers one real SNAP per currently known symbol, never a single SNAP with a literal SYM=*.
- TRADE and AUCTION subscriptions do not have a baseline SNAP; only future events are sent for those two channels.
- Invalid requests return ERR.
- Existing successful subscriptions remain active when a later SUB request is invalid.

```
SUB|CH=TOP,TRADE|SYM=AAPL,MSFT
SUB|CH=STATE|SYM=*
SUB|CH=TRADE|SYM=*
SUB|CH=DEPTH|SYM=AAPL
SUB|CH=AUCTION|SYM=*
SUB|CH=CB|SYM=AAPL
```

5.8.4 RESUME

Direction: Client -> Gateway

Purpose: Subscribe to one stream and replay what was missed since `LASTSEQ`, instead of starting from a `SNAP`.

Response: replayed events in sequence order, or `ERR|CODE=REPLAY_MISS` when the gap is wider than the replay window or `LASTSEQ` is ahead of the stream. On the snapshot-backed channels (`TOP`, `STATE`, `INDEX`, `DEPTH`, `CB`) a fresh `SNAP` follows that error; on `TRADE` and `AUCTION` it does not, because a past print has no current state to snapshot. See "Reconnect behavior" below.

Field	Req	Description
`CH`	Yes	Exactly one channel
`SYM`	Yes	Exactly one concrete symbol; `SYM=*` is invalid
`LASTSEQ`	Yes	Last sequence the client received on that stream

Send one `RESUME` per stream being recovered. Unlike `SUB`, this message takes no comma-separated lists: `LASTSEQ` describes a single stream's position, so a multi-stream `RESUME` would have no coherent meaning.

```
RESUME|CH=TOP|SYM=AAPL|LASTSEQ=1042
RESUME|CH=TRADE|SYM=AAPL|LASTSEQ=88
RESUME|CH=DEPTH|SYM=MSFT|LASTSEQ=310
```

Validation rules:

- `CH` or `SYM` carrying more than one value is `ERR|CODE=BAD_MESSAGE`.
- A missing or non-positive `LASTSEQ` is `ERR|CODE=BAD_MESSAGE`.
- An unknown channel is `ERR|CODE=INVALID_CHANNEL`.
- `SYM=*` is `ERR|CODE=INVALID_SYMBOL` on every channel — see "Reconnect behavior".

All of these leave the session open. A client recovering several streams sends several `RESUME` messages, and one bad request must not cost it the others; only handshake-level failures close a connection.

5.8.5 SYMBOLS

Direction: Client -> Gateway (request), Gateway -> Client (reply)

Purpose: Ask which instruments this gateway knows about.

The request carries no fields:

```
SYMBOLS
```

The reply:

Field	Req	Description
`COUNT`	Yes	How many symbols are known; `0` is a valid answer
`SYMBOLS`	No	Comma-separated symbols, sorted; omitted when `COUNT=0`
`REF`	No	Per-symbol `SYM:DEC` display precision; same set as `SYMBOLS`, omitted alongside it

```
SYMBOLS|COUNT=3|SYMBOLS=AAPL,MSFT,TSLA|REF=AAPL:2,MSFT:2,TSLA:4
SYMBOLS|COUNT=0
```

Read `COUNT`, not the presence of `SYMBOLS`. An empty universe omits the field rather than sending it empty, so the two must not be conflated with a malformed reply.

Why this exists, given `WELCOME|SYMBOLS=`. That field is optional, sent once, and omitted entirely when the gateway was started without a readable engine config — a misconfiguration that otherwise looks, from the client side, exactly like an exchange with no instruments. The gateway's set also *grows* as symbols first appear on the engine bus, so a client that connected before a given instrument traded could not learn of it without reconnecting. `SYMBOLS` makes the universe both askable and refreshable.

Repeatable at any time. A client that needs the universe up front should send it immediately after `WELCOME` rather than depending on the handshake field.

`REF` — per-symbol display precision

`REF` answers "how many decimal places does this instrument quote in?", which a client otherwise has no way to discover. The only other source is `GET /api/symbols` on the API gateway, which requires a *trading* credential — something a market data consumer has no business holding — so before this field existed every CALF client had to assume the default of `2` and was quietly wrong about any instrument configured otherwise.

```
REF=AAPL:2,MSFT:2,TSLA:4
```

Three properties are deliberate:

- **It is reference data, not market data.** `tick_decimals` never changes for a symbol, so it rides the handshake and this reply rather than `TOP` or `TRADE`. Repeating a constant on every tick would be exactly what `MD`'s delta encoding exists to avoid.
- **Its presence is the capability signal.** Like `CH_SUPPORTED`, and for the same reason: `PROTO` stays `CALF1`. A client seeing no `REF` falls back to `2` *knowingly* rather than by accident.
- **The tuple has room to grow.** `SYM:DEC` reuses the colon-delimited grammar `DEPTH` already uses for `price:qty:count`, and extends to `SYM:DEC:MULT:CCY` as further reference fields are defined — without another protocol change. Clients must ignore trailing components they do not recognise rather than treating the entry as malformed.

Every symbol in `SYMBOLS` appears in `REF`, so a client never has to reason about one being listed in one field and missing from the other. A symbol first seen on the engine bus, which has no configured precision, is reported at the default rather than omitted.

5.8.6 `UNSUB`

Direction: Client -> Gateway

Field	Req	Description
CH	Yes	Comma-separated channels
SYM	Yes	Comma-separated symbols

UNSUB is idempotent. Removing a non-existent (CH,SYM) pair has no effect.

```
UNSUB|CH=TOP|SYM=AAPL
```

5.8.7 SNAP

Direction: Gateway -> Client

Purpose: Baseline for one stream.

SNAP uses channel-specific payload fields.

Common fields:

Field	Req	Description
CH	Yes	TOP, STATE, INDEX, DEPTH, or CB
SYM	Yes	Symbol, index id, or * for session state
SEQ	Yes	Current stream sequence
TS	Yes	Snapshot timestamp

CH=TOP fields:

Field	Req	Description
BID	No	Best bid price
BIDSZ	No	Best bid size
ASK	No	Best ask price
ASKSZ	No	Best ask size
LAST	No	Last trade price
LASTSZ	No	Last trade size

A SUB|CH=TOP|SYM=* never produces a SNAP with a literal SYM=* — see "Channel model" above. Each SNAP in the resulting burst has a real SYM and uses that symbol's own (TOP, SYM) sequence.

CH=STATE fields:

Field	Req	Description
SESSION	Yes	Current state value

CH=INDEX fields: identical field set to the IDX message — see that section below. Since CALF 1.0.0, SUB|CH=INDEX sends a baseline SNAP before any live IDX updates, the same as TOP/STATE/DEPTH.

CH=DEPTH fields: identical field set to the DEPTH message — see that section below. BIDS/ASKS are omitted entirely (not sent as empty strings) when a symbol's book has no resting orders on that side yet.

`CH=CB` fields: identical field set to the `CB` message — see that section below. Reflects the **last known** circuit-breaker status for the symbol — `STATUS=ACTIVE` with no further fields if the symbol has never halted.

`TRADE`/`AUCTION` stream note:

- Neither `CH=TRADE` nor `CH=AUCTION` has a `SNAP` variant in CALF `1.0.0` — both are pure event streams with no persistent "current value."
- Delivery for both starts from events that occur after the subscription is active (plus any replay via `RESUME`, see "Sequence and recovery semantics").

```
SNAP|CH=TOP|SYM=AAPL|SEQ=100|TS=2026-06-07T10:16:00.000Z|BID=150.10|BIDSZ=1200|ASK=150.12|ASKSZ=900|LAST=150.
SNAP|CH=STATE|SYM=*|SEQ=5|TS=2026-06-07T10:16:00.000Z|SESSION=CONTINUOUS
SNAP|CH=INDEX|SYM=EDU100|SEQ=42|TS=2026-06-12T10:15:23.000Z|LEVEL=1048.73|OPEN=1042.10|HIGH=1056.30|LOW=1040.
SNAP|CH=DEPTH|SYM=AAPL|SEQ=1|TS=2026-07-11T14:32:00.000Z|LEVELS=10|BIDS=150.10:1200:3,150.09:800:2|ASKS=150.1
SNAP|CH=CB|SYM=AAPL|SEQ=3|TS=2026-07-20T14:05:00.000Z|STATUS=HALTED|LEVEL=L2|TRIGGERPX=148.20|REFPX=150.10|RE
SNAP|CH=CB|SYM=MSFT|SEQ=1|TS=2026-07-20T14:05:00.000Z|STATUS=ACTIVE
```

5.8.8 `MD`

Direction: Gateway -> Client

Purpose: Incremental `TOP` update. Unchanged sides may be omitted.

Field	Req	Description
`CH`	Yes	`TOP`
`SYM`	Yes	Symbol
`SEQ`	Yes	Stream sequence
`TS`	Yes	Event timestamp
`BID`	No	Updated bid; **empty value means the bid side is now empty**
`BIDSZ`	No	Updated bid size
`ASK`	No	Updated ask; **empty value means the ask side is now empty**
`ASKSZ`	No	Updated ask size
`LAST`	No	Updated last trade price
`LASTSZ`	No	Updated last trade size

```
MD|CH=TOP|SYM=AAPL|SEQ=1051|TS=2026-06-07T10:16:00.115Z|BID=150.11|BIDSZ=1400|ASK=150.13|ASKSZ=800
```

Withdrawal of a book side

`BID` and `ASK` have three distinct states on an `MD`, and a client must treat them differently:

On the wire	Meaning
`BID=150.11`	New value for this side
`BID=`	This side is now **empty** — discard the price
(absent)	Unchanged since the previous message

An empty value is the *only* way the gateway can say "there is no bid". This is the one documented

exception to **Wire value types** above: `BID/ASK` are optional fields, and for these two an empty value is meaningful rather than malformed.

```
MD|CH=TOP|SYM=AAPL|SEQ=1052|TS=2026-06-07T10:16:00.210Z|BID=|BIDSZ=0
```

A client that treats a withdrawal as "unchanged" will display the last known price indefinitely, and will disagree with any client that reconnects and receives a fresh `SNAP` — which omits the side correctly. Merging code must therefore *remove* the field on an empty value, not overwrite it.

`LAST/LASTSZ` are never withdrawn: once a symbol has traded, its last price persists for the session, so an empty value is not valid for those fields.

`LAST` after a trade

A trade updates `LAST/LASTSZ` on the `TOP` channel as well as producing a `TRADE` message. The two arrive at different times and neither replaces the other:

Channel	Carries	When
`TRADE`	Every individual print (`PX`, `QTY`, `SIDE`)	Immediately, one message per trade
`TOP`	The latest price only (`LAST`, `LASTSZ`)	With the next book republish, throttled by the engine's `snapshot_interval_sec`

A client that wants every print subscribes to `TRADE`; a client that only wants "what did this last trade at" can rely on `TOP` alone, including its `SNAP` baseline. Several trades inside one throttle window collapse to the latest price on `TOP` — that is the intended behaviour, not a dropped update.

A `SNAP` reports a trade immediately, without waiting for the next book republish, so a client subscribing between the two is not handed a stale price. Gateway builds before this was fixed suppressed `LAST` from the following `MD` entirely, leaving a continuously-connected client on the price baked into its original `SNAP` while a reconnecting client saw the true value.

5.8.9 `TRADE`

Direction: Gateway -> Client

Field	Req	Description
`CH`	Yes	`TRADE`
`SYM`	Yes	Symbol
`SEQ`	Yes	Stream sequence
`TS`	Yes	Trade timestamp
`TRADE_ID`	Yes	Durable public trade identifier
`RUN_SEQ`	Yes	Persisted engine-run sequence in the identifier prefix
`PX`	Yes	Trade price
`QTY`	Yes	Trade quantity
`SIDE`	Yes	Aggressor side (`BUY` or `SELL`)

```
TRADE|CH=TRADE|SYM=AAPL|SEQ=809|TS=2026-06-07T10:16:00.141Z|TRADE_ID=000042-000000123|RUN_SEQ=42|PX=150.12|QT
```

5.8.10 STATE

Direction: Gateway -> Client

Field	Req	Description
CH	Yes	STATE
SYM	Yes	Symbol or *
SEQ	Yes	Stream sequence
TS	Yes	Transition timestamp
SESSION	Yes	New state value
PREV	No	Previous state when known
NEXTPHASE	No	Phase the session moves to next; SYM=* only
NEXTAT	No	When that transition is scheduled, UTC ISO-8601; SYM=* only

Valid SESSION values:

- PRE_OPEN
- OPENING_AUCTION
- CONTINUOUS
- CLOSING_AUCTION
- CLOSED
- HALTED (symbol-level)
- UNKNOWN (the gateway has not yet heard the phase from the engine)

UNKNOWN is what a freshly started gateway reports until the engine has answered its session-state query, normally within a couple of seconds of both running. The engine announces its phase only on a transition, and one started outside trading hours makes none, so the gateway asks the engine directly instead of assuming. A client should show UNKNOWN as "not known yet" rather than as either trading or closed. When the answer arrives (or a later one shows that a transition was missed, for example after an engine restart, or that the next scheduled transition has changed), the gateway publishes it as an ordinary SYM=* STATE line with no PREV. It carries NEXTPHASE and NEXTAT when the engine holds a next transition.

```
STATE|CH=STATE|SYM=*|SEQ=14|TS=2026-06-07T10:30:00.000Z|SESSION=CONTINUOUS|PREV=OPENING_AUCTION|NEXTPHASE=CLO
STATE|CH=STATE|SYM=AAPL|SEQ=8|TS=2026-06-07T10:30:00.000Z|SESSION=CONTINUOUS|PREV=OPENING_AUCTION
STATE|CH=STATE|SYM=AAPL|SEQ=9|TS=2026-06-07T11:02:17.330Z|SESSION=HALTED|PREV=CONTINUOUS
```

NEXTPHASE/NEXTAT: the next scheduled transition.

Both appear on the SYM=* stream only, and only when the session scheduler (the one component that knows the day's timetable) has named the next transition. It does so with each transition it drives, and also when it starts and has no transition to send yet: before the day's first one, or while the exchange is closed. A pm-scheduler --daily close names the next trading day's pre-open, so a closed exchange counts down to its reopening; a single-shot run's close names nothing, since no scheduler will be there to open. These fields are what lets a client show a countdown to the open, the closing auction, or the close, which is otherwise the most-glanced number on a trading screen and the one CALF could not answer.

They are sent together or not at all. A phase with no time cannot be counted down to, and a time with no phase does not say what happens when it arrives.

Their absence is information. A manual or admin-driven transition carries no timetable, and the engine *clears* whatever the scheduler last advertised rather than leaving it in place: it has just moved somewhere the schedule did not predict, so the old target has stopped being a fact about anything. A client must render that as silence, not as a countdown to zero — and must not substitute a schedule it read from configuration, which describes what *should* happen rather than what the engine is actually going to do.

`NEXTAT` may pass without the transition arriving, if the scheduler is late or has stopped. A client should say so rather than run a negative clock or freeze at zero, since a late scheduler, a wedged one, and an absent one otherwise look identical.

An exchange transition is published twice: once as `SYM=*`, and once per symbol. A subscription matches on `SYM=*` *or* an exact symbol, so a client that subscribed to one instrument would otherwise see its halts and resumes but never the session around them — it would not learn the exchange had opened or closed. Wildcard subscribers receive both forms; that is deliberate, and the `SYM=*` line remains the authoritative exchange-level event.

A symbol that is halted is **not** moved by an exchange transition. Its halt outlives the phase it began in, and the engine publishes an explicit resume when it ends.

A resume returns the symbol to whatever the exchange is doing *at that moment*, which is not necessarily `CONTINUOUS`. Circuit-breaker halts expire on elapsed time with no session check, so an L2 halt — 15 minutes by default — triggered shortly before the close resumes into `CLOSING_AUCTION` or `CLOSED`.

5.8.11 `IDX`

Direction: Gateway -> Client

Purpose: Index level update for one `INDEX` stream.

Field	Req	Description
`CH`	Yes	`INDEX`
`SYM`	Yes	Index identifier (e.g. `EDU50`)
`SEQ`	Yes	Stream-local sequence number
`TS`	Yes	Event timestamp
`LEVEL`	Yes	Current index level, decimal string
`SESSION`	Yes	Index session state
`OPEN`	No	Day open level, decimal string
`HIGH`	No	Day high, decimal string
`LOW`	No	Day low, decimal string
`CHG`	No	Change from open, signed decimal e.g. `+1.23`
`PCTCHG`	No	Percent change from open, signed e.g. `+0.45`
`AGGCAP`	No	Aggregate market cap, integer string

`IDX` has no baseline `SNAP` variant in CALF `1.0.0`. Delivery starts from events that occur after the

subscription becomes active.

```
IDX|CH=INDEX|SYM=EDU50|SEQ=12|TS=2026-06-07T10:16:00.000Z|LEVEL=5123.45|SESSION=CONTINUOUS|OPEN=5100.00|CHG=+
```

5.8.12 `DEPTH`

Direction: Gateway -> Client

Purpose: Aggregated, multi-level order book update — CALF's Level 2 view. Each level is an aggregate of every resting order at that price; individual order identity is never exposed (Level 3 is explicitly out of scope, see "Out of scope in CALF 1.0.0" above).

Response: No reply required. A gap in `SEQ` should trigger replay or resync, exactly as for `MD`.

Unlike `MD`, which omits individual unchanged `BID`/`ASK` fields, `DEPTH` is a **full-ladder replace per message**: whenever the top-`LEVELS` price levels on either side change, the message carries that side's complete current ladder, not a per-level diff. `DEPTH` is only sent when the tracked levels actually changed since the previous `DEPTH`/`SNAP` for the symbol.

Field	Req	Description
`CH`	Yes	`DEPTH`
`SYM`	Yes	Symbol
`SEQ`	Yes	Stream sequence for `(DEPTH, SYM)`
`TS`	Yes	Event timestamp
`LEVELS`	Yes	Number of price levels per side configured on this gateway (`market_data_gateway.depth_levels`, default `10`)
`BIDS`	No	Bid-side ladder, best price first; omitted if no resting bids
`ASKS`	No	Ask-side ladder, best price first; omitted if no resting asks

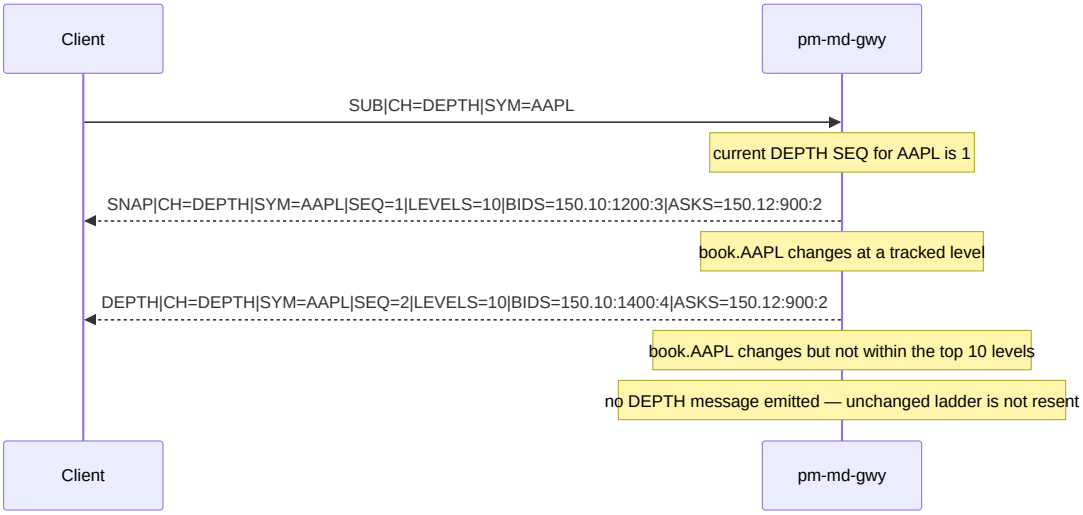
Level encoding grammar (applies to both `BIDS` and `ASKS`):

```
<LEVELS_VALUE> ::= <LEVEL> ("," <LEVEL>)*
<LEVEL>        ::= <PRICE> ":" <QTY> ":" <COUNT>
```

`PRICE` is decimal text, `QTY` is the aggregated resting quantity at that price, `COUNT` is the number of individual resting orders aggregated into it. `:` and `,` are ordinary field-value characters in CALF — the only reserved wire character is `|`.

```
DEPTH|CH=DEPTH|SYM=AAPL|SEQ=2|TS=2026-07-11T14:32:00.512Z|LEVELS=10|BIDS=150.10:1400:4,150.09:800:2,150.08:40
```

`SYM=*` is invalid for `SUB|CH=DEPTH` — see "Subscription rules" above.



5.8.13 `AUCTION`

Direction: Gateway -> Client

Purpose: Result of one auction uncross for a symbol — a scheduled opening/closing auction, a circuit-breaker reopening auction, or the pass over restored GTC orders at engine startup. Published exactly once per uncross, even when there was no crossable interest at all.

Field	Req	Description
`CH`	Yes	`AUCTION`
`SYM`	Yes	Symbol
`SEQ`	Yes	Stream sequence for `(AUCTION, SYM)`
`TS`	Yes	Event timestamp
`EQPX`	No	Equilibrium price; omitted when there was no crossable interest
`EQQTY`	Yes	Total executable quantity matched at `EQPX` (`0` when no cross)
`TRADES`	Yes	Number of trades produced by the uncross (`0` when no cross)
`IMBSIDE`	No	Residual imbalance side, `BUY` or `SELL`; omitted when balanced or no cross
`IMBQTY`	Yes	Residual imbalance quantity at `EQPX` (`0` when balanced or no cross)
`REASON`	No	Which uncross this was: `SCHEDULED` (leaving an auction or other non-matching phase), `REOPEN` (a halted symbol reopening) or `RECOVERY` (restored GTC orders at engine startup)

Without `REASON` the three are indistinguishable — the fields are otherwise identical — so a client cannot tell a circuit-breaker reopening from the closing auction. Treat an unrecognised value as absent rather than as an error: it means a gateway newer than the client.

`AUCTION` has no baseline `SNAP` (see "Channel model" above) — a new subscriber only receives auction results from the next uncross onward, unless it also uses `RESUME` to replay recent history.

```
AUCTION|CH=AUCTION|SYM=AAPL|SEQ=1|TS=2026-07-20T13:30:00.012Z|EQPX=150.10|EQQTY=48200|TRADES=37|IMBSIDE=BUY|IMBQTY=14
AUCTION|CH=AUCTION|SYM=TSLA|SEQ=4|TS=2026-07-20T20:00:00.004Z|EQQTY=0|TRADES=0|IMBQTY=0
AUCTION|CH=AUCTION|SYM=MSFT|SEQ=2|TS=2026-07-20T13:30:00.031Z|EQPX=421.00|EQQTY=15000|TRADES=12|IMBQTY=0
```

The second example is a no-cross auction (`EQPX`/`IMBSIDE` both omitted, all counts `0`); the third is a perfectly balanced cross (`IMBSIDE` omitted, `IMBQTY=0`, but `EQPX`/`EQQTY`/`TRADES` all present).

5.8.14 `INDIC`

Direction: Gateway -> Client

Purpose: Where a symbol *would* uncross if the call phase ended now. Published repeatedly, on an interval, for the whole of an `OPENING_AUCTION` or `CLOSING_AUCTION`.

This is the same channel as `AUCTION` and a different statement. `AUCTION` reports what happened at an uncross; `INDIC` reports what would happen while there is still time to act on it. A client must not treat one as the other: an `INDIC` price is not a trade and nothing has printed at it.

Field	Req	Description
`CH`	Yes	`AUCTION`
`SYM`	Yes	Symbol
`SEQ`	Yes	Stream sequence for `(AUCTION, SYM)`
`TS`	Yes	Event timestamp
`INDICPX`	No	Indicative uncross price; **omitted when the book would not cross at all**
`INDICQTY`	Yes	Quantity that would match at `INDICPX` (`0` when there is no cross)
`IMB`	No	Which side the surplus runs, `BUY` or `SELL`; omitted when balanced
`IMBQTY`	Yes	Surplus quantity that would go unmatched (`0` when balanced)
`PHASE`	No	The call phase running, so a client need not infer it from `STATE`

`INDICPX` absent is a reading, not a gap. It means the bids and offers collected so far do not overlap, so nothing would trade if the phase ended now. Rendering it as a price of zero is wrong in the way that matters: it asserts a clearing level the book does not have.

Why publish it at all. An imbalance nobody can see before the uncross is an imbalance nobody can offset. Disseminating it is what lets participants supply the offsetting interest that resolves it — the same reasoning the circuit-breaker path already applies to a reopening, and it holds with more force at the open and the close, where the largest volume of the day prints. The field names are shared with that path deliberately: a reopening auction and a scheduled one are the same mechanism, and a client that learned to read one should not have to learn the other.

Cadence. A fixed interval, configurable engine-side (`auction_indicative_interval_sec`, default 1s), not one message per book change: bounded cost regardless of how heavy order entry gets. Every symbol is republished every interval, including ones whose reading has not changed — otherwise a client cannot tell a stable indicative from a stalled feed.

Halted symbols are excluded. A halt is its own reopening auction with its own corridor, and the `CB` channel already carries an indicative for it. Two sources describing one symbol would eventually disagree.

Like `AUCTION`, `INDIC` has no baseline `SNAP`. A client joining mid-auction waits at most one interval for the next reading.

```
INDIC|CH=AUCTION|SYM=AAPL|SEQ=12|TS=2026-07-20T13:29:45.000Z|INDICPX=150.10|INDICQTY=48200|IMB=BUY|IMBQTY=140
INDIC|CH=AUCTION|SYM=TSLA|SEQ=12|TS=2026-07-20T13:29:45.000Z|INDICQTY=0|IMBQTY=0|PHASE=OPENING_AUCTION
```

5.8.15 CB

Direction: Gateway -> Client

Purpose: Full circuit-breaker halt/resume detail for one symbol — trigger price, reference price, ladder level, scheduled auto-resume time, and what caused the halt. `STATE` (above) still carries the coarse `SESSION=HALTED`/`SESSION=CONTINUOUS` transition unchanged; `CB` is emitted **alongside** `STATE`, from the same underlying engine event, for clients that also want the detail.

Field	Req	Description
`CH`	Yes	`CB`
`SYM`	Yes	Symbol
`SEQ`	Yes	Stream sequence for `(CB, SYM)`
`TS`	Yes	Event timestamp
`STATUS`	Yes	`ACTIVE` (not halted) or `HALTED`
`LEVEL`	No	Ladder level (e.g. `L1`/`L2`/`L3`, config-defined) or `ADMIN_ALL`/`ADMIN_SYMBOL` for an operator-initiated halt; present only when `STATUS=HALTED`
`TRIGGERPX`	No	Trigger price; present only for an automatic (non-`ADMIN_*`) halt currently in effect
`REFPX`	No	Reference price at trigger time; present only for an automatic halt currently in effect
`RESUMEAT`	No	Scheduled auto-resume time, UTC ISO-8601 with ms (same format as `TS`); present only for a timed halt currently in effect — absent for rest-of-day or manual/`ADMIN_*` halts
`CORRLO`	No	Lower bound of the ACE reopening corridor; present only while `STATUS=HALTED` and ACE is enabled for the symbol
`CORRHI`	No	Upper bound of the same corridor
`EXP`	No	Number of ACE extensions consumed so far. `0` on the initial halt
`SRC`	No	What halted the symbol: `CB` for an automatic breaker trigger, `ADMIN` for an operator halt
`INDICPX`	No	Indicative uncross price observed at the end of a call phase. Extension events only
`INDICQTY`	No	Quantity that would have executed at `INDICPX`. Extension events only
`IMB`	No	`BUY` or `SELL` — which side the imbalance ran. Extension events only
`REASON`	No	`CLOSING_BACKSTOP` when the resume was forced by the end of the trading day. Resume events only
`CLAMPED`	No	`1` when the backstop printed *at* the corridor boundary rather than at the equilibrium. Resume events only
`PRINTPX`	No	The price the backstop printed at. Resume events only

`LEVEL`/`TRIGGERPX`/`REFPX`/`RESUMEAT`/`CORRLO`/`CORRHI`/`EXP` describe the halt that just ended and are always omitted on a resume event (`STATUS=ACTIVE`) — only `SRC` carries over, since what caused the halt is meaningful on the way out as well as in.

ACE corridor expansions

A halt does not necessarily end when `RESUMEAT` arrives. If the indicative uncross price falls outside `[CORRLO, CORRHI]`, the symbol stays halted, the corridor widens by one ladder rung and a fresh call phase begins — see Risk Controls - Automated Corridor Expansion.

The gateway publishes this as a further `CB` event with `STATUS=HALTED` and an updated `RESUMEAT`, `CORRLO`, `CORRHI` and `EXP`. **A client that ignores these will show a `RESUMEAT` that has already passed and report the symbol as overdue to reopen.** No `STATE` event accompanies an extension: the symbol was halted before and is halted after, so the coarse session state has not changed.

`INDICPX`/`INDICQTY`/`IMB` are carried on extension events only. They are computed once, at the instant the call phase ends, so they are deliberately absent from the `SNAP` baseline — replaying them later would assert a stale price for a book that has kept moving. Disseminating them at all mirrors the order-imbalance indicator real venues publish during a reopening, which is what lets participants supply the offsetting interest that resolves the halt.

`CORRLO`/`CORRHI`/`EXP` *are* part of the `SNAP` baseline, because they describe the halt still in force: a client subscribing mid-halt otherwise cannot tell where the symbol is permitted to reopen.

A halt is a reopening auction's call phase. While a symbol is halted the engine accepts LIMIT orders and rests them, rejects MARKET/FOK/IOC, and runs no matching; when the halt ends it uncrosses at the equilibrium price and publishes an `AUCTION` with `REASON=REOPEN` before the `STATE` and `CB` resume events. There is no separate "resumption mode" to choose, and CALF no longer carries one: crossed interest accumulates during the call, so restarting continuous matching without an uncross would begin on a crossed book. `RESUMEAT` says whether the halt ends by itself; `SRC` says who started it. The two are independent.

`CB` has a baseline `SNAP` (see the `SNAP` section above) reflecting the last known status for the symbol.

```
CB|CH=CB|SYM=AAPL|SEQ=4|TS=2026-07-20T14:05:00.010Z|STATUS=HALTED|LEVEL=L2|TRIGGERPX=148.20|REFPX=150.10|RESU
CB|CH=CB|SYM=TSLA|SEQ=1|TS=2026-07-20T15:00:00.000Z|STATUS=HALTED|LEVEL=ADMIN_ALL|SRC=ADMIN
CB|CH=CB|SYM=AAPL|SEQ=5|TS=2026-07-20T14:20:00.010Z|STATUS=ACTIVE|SRC=CB
```

An ACE sequence — halt, one extension, then a reopen inside the widened corridor:

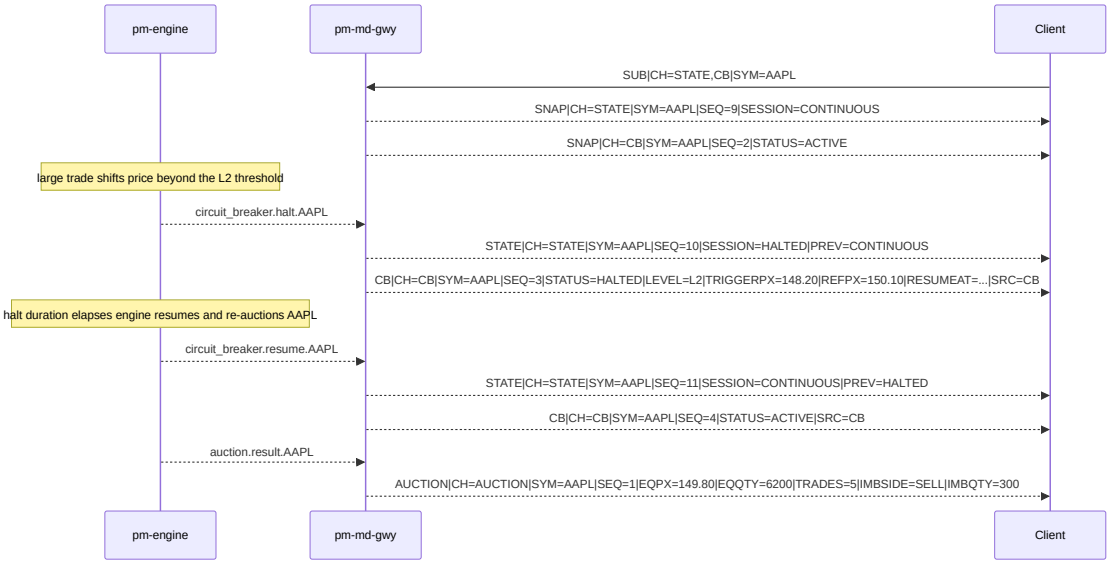
```
CB|CH=CB|SYM=AAPL|SEQ=4|TS=2026-07-20T13:30:00.010Z|STATUS=HALTED|LEVEL=L1|TRIGGERPX=122.00|REFPX=100.00|RESU
CB|CH=CB|SYM=AAPL|SEQ=5|TS=2026-07-20T13:35:00.010Z|STATUS=HALTED|LEVEL=L1|TRIGGERPX=122.00|REFPX=100.00|RESU
CB|CH=CB|SYM=AAPL|SEQ=6|TS=2026-07-20T13:37:00.010Z|STATUS=ACTIVE|SRC=CB
```

And a halt the trading day ended before ACE could resolve, printed at the corridor boundary rather than at the equilibrium:

```
CB|CH=CB|SYM=AAPL|SEQ=9|TS=2026-07-20T16:05:00.010Z|STATUS=ACTIVE|SRC=CB|REASON=CLOSING_BACKSTOP|CLAMPED=1|PR
```

The first example is an automatic threshold-breach halt (all detail fields present); the second is an ADMIN exchange-wide halt (`trigger`/`reference`/ `resume` all omitted, matching the engine's `None` values for that path); the third is the resume that follows the first halt (`STATUS=ACTIVE`, only `SRC` retained).

`SYM=*` is invalid for `SUB|CH=CB` — see "Subscription rules" above.



5.8.16 `HB`

Direction: Gateway -> Client

Sent when no outbound market-data line was emitted during the heartbeat interval.

Field	Req	Description
`TS`	Yes	Gateway timestamp

```
HB|TS=2026-06-07T10:16:05.000Z
```

5.8.17 `PING` / `PONG`

`PING` is client-initiated liveness check. `PONG` is immediate reply.

Field	Req	Description
`TS`	Yes	Gateway timestamp

```
PING
PONG|TS=2026-06-07T10:16:05.000Z
```

5.8.18 `ERR`

Direction: Gateway -> Client

Field	Req	Description
`CODE`	Yes	Machine-readable error code
`DETAIL`	No	Human-readable context
`CH`	No	Channel context when relevant
`SYM`	No	Symbol context when relevant

Normative error codes:

Code	Meaning
AUTH_REQUIRED	Non-HELLO message sent before successful handshake, or HELLO missing CLIENT, CLIENT over 32 chars, or PROTO != CALF1
INVALID_CHANNEL	CH not in {TOP, TRADE, STATE, INDEX, DEPTH, AUCTION, CB}
INVALID_SYMBOL	Symbol not in the gateway's known instrument list (does not apply to INDEX, which has no known-id check — see "Subscription rules"), SYM=* used with INDEX/DEPTH/CB on SUB (alone or mixed with any other channel), SYM=* used at all on RESUME (every channel, including TOP/TRADE/STATE/AUCTION), a SUB with no SYM at all, or CH=INDEX combined with an empty symbol
SUB_LIMIT	Subscription would exceed max_symbols_per_client
REPLAY_MISS	Events after LASTSEQ are no longer in the replay window, or LASTSEQ is ahead of the stream (e.g. after a gateway restart); a SNAP follows on TOP/STATE/INDEX/DEPTH/CB, but not on TRADE/AUCTION
SLOW_CLIENT	Outbound queue exceeded max_client_queue; connection closed
BAD_MESSAGE	Parse failure, oversized line (over 4096 bytes including the newline), or unsupported message type
RATE_LIMITED	Client exceeded max_messages_per_second (inbound token-bucket); connection stays open

Terminal behavior:

- AUTH_REQUIRED is terminal: any non-HELLO message before a successful handshake gets this error and the connection is closed immediately.
- SLOW_CLIENT is terminal for the current TCP session; gateway disconnects.
- BAD_MESSAGE is terminal for an oversized line (DETAIL=line exceeds 4096 bytes), the same in pm-ralf-gwy and pm-dc-gwy; for any other cause the session stays open.
- RATE_LIMITED is non-terminal; the offending message is dropped and the connection remains open. The client may retry once its send rate is back under max_messages_per_second.

```
ERR|CODE=REPLAY_MISS|DETAIL=Requested sequence outside replay buffer|CH=TOP|SYM=AAPL
```

5.8.19 EXIT

Direction: Both

From the client, requests a clean disconnect. From the gateway, announces that the gateway is ending the session, and why.

Field	Req	Description
REASON	Gateway only	Why the session ends, e.g. idle_timeout
TS	Gateway only	Gateway timestamp

```
EXIT
EXIT|REASON=idle_timeout|TS=2026-06-07T10:16:05.000Z
```

When the gateway ends a session — EXIT, SLOW_CLIENT, a refused HELLO, an oversized line — the final line is the last thing sent; the gateway then closes the socket once it has drained, or after at most 5 seconds if the client has stopped reading. This is the same close behaviour as pm-ralf-gwy and pm-dc-gwy.

5.9 Sequence and recovery semantics

5.9.1 Stream identity

Sequence numbers are maintained per `(CH, SYM)` stream.

Examples:

- `(TOP, AAPL)` has its own counter
- `(TRADE, AAPL)` has a different counter
- `(STATE, *)` and `(STATE, AAPL)` are distinct counters

5.9.2 Sequence rules

- Start value is `1` for each stream.
- Increment by `1` per emitted message in that stream.
- Sequence appears in `SNAP`, `MD`, `TRADE`, `STATE`, `IDX`, `DEPTH`, `AUCTION`, and `CB`.
- A client-detected gap means one or more missed messages.

5.9.3 First connect behavior

On first subscribe to a `TOP`, `STATE`, `INDEX`, `DEPTH`, or `CB` stream:

1. Gateway sends `SNAP` with current stream `SEQ`.
2. Client stores `last_seq[(CH,SYM)] = SNAP.SEQ`.
3. Next incremental event for that stream must be `SEQ + 1`.

`TRADE` and `AUCTION` have no step 1 — see the `SNAP` section above.

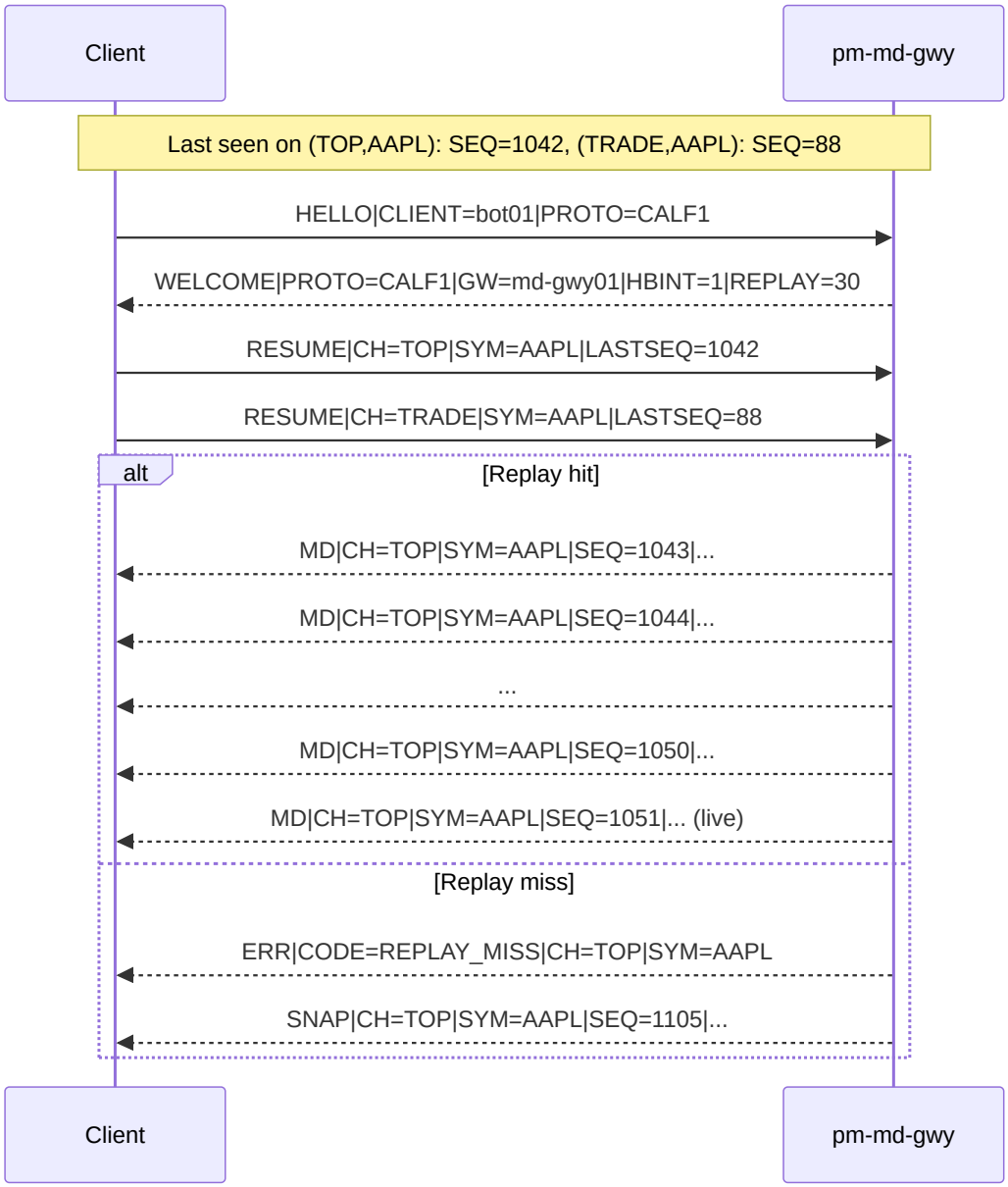
5.9.4 Reconnect behavior (`RESUME`)

`RESUME` applies to one stream per message, and may be sent as many times as there are streams to recover. Reconnect therefore looks like: `HELLO`, then one `RESUME` per stream the client was following, then a `SUB` for anything it wants that it has no sequence position for.

- Client supplies `CH`, `SYM`, and `LASTSEQ`.
- `CH` and `SYM` must each contain exactly one value.
- `LASTSEQ` must be a positive base-10 integer.
- If missing events are inside replay window, gateway replays in order then continues live.
- If missing range is outside window, gateway sends `ERR|CODE=REPLAY_MISS`. A fresh `SNAP` follows on `TOP`/`STATE`/`INDEX`/`DEPTH`/`CB`. It does **not** on `TRADE` or `AUCTION`: those carry discrete events, and there is no snapshot of a print that already happened, so the missed events are simply gone. Do not wait for a baseline that will not arrive — and if you are talking to an older gateway that sends one anyway, discard it: it is an envelope with no payload, and a decoder keyed on `CH` alone will read it as a print of zero shares at zero price.
- **A `SNAP` re-baselines the stream; it is never a gap.** Whatever `SEQ` it carries becomes your new `last_seq` for that stream, with no gap check. Gap checking a `SNAP` would ask to replay history it just superseded, and on the replay-miss path — whose answer *is* a `SNAP` — loops `RESUME` against a window already known to be too old.
- **A replay is not disjoint from live delivery.** `RESUME|LASTSEQ=n` returns every buffered message with `SEQ > n`, and `n` is your position from *before* the gap — so the reply re-sends the message that revealed the gap, plus anything delivered live while your request was in flight. Replayed and live lines share one ordered connection, so duplicates always arrive after their originals. **Discard any message at or below the `SEQ` you have already recorded, and**

never let one lower your `last_seq`. Track which sequence ranges you are actually missing: that is the only thing distinguishing the backfill you asked for from a print you already have.

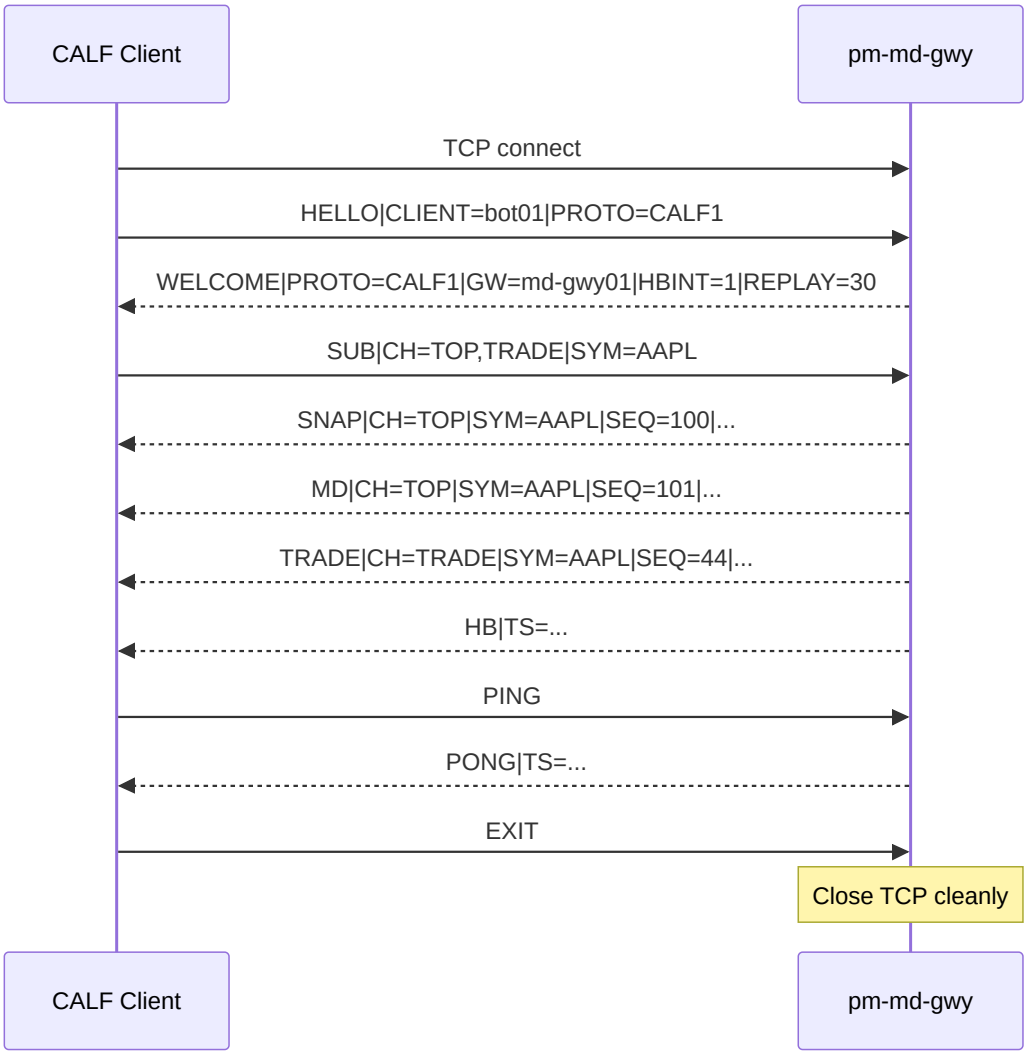
- Zero replay lines and no error mean exactly one thing: `LASTSEQ` is the stream's current sequence, so nothing was missed. Every other case the gateway cannot serve is a `REPLAY_MISS` — including a stream whose retained events have all aged out, and a `LASTSEQ` ahead of the stream (or on a stream this gateway has never emitted), which is what a client holding a position from before a gateway restart sends.
- `SYM=*` is always invalid for `RESUME`, for every channel, even for `TOP`/`TRADE`/`STATE`/`AUCTION` where `SYM=*` is otherwise allowed on `SUB`. `RESUME` has no equivalent of `SUB`'s per-symbol snapshot burst, so a wildcard resume cannot be served a meaningful baseline on a replay miss. `RESUME|CH=TOP|SYM=*` returns `ERR|CODE=INVALID_SYMBOL` and, unlike an ineligible wildcard on `HELLO` previously, leaves the session open. Clients must always resume a single concrete symbol and, if they also want an "everything" subscription, add it separately via `SUB|SYM=*` after reconnecting.
- Beyond the wildcard rule above, `RESUME`'s `SYM` value is otherwise **not** checked against the gateway's known-symbol list the way `SUB`'s is — a resume for a symbol the gateway doesn't currently know about is still accepted and added to the session's subscriptions; it simply won't match any live event until the gateway learns about that symbol.



5.10 Liveness and timeout rules

- Gateway emits `HB` every `heartbeat_interval_sec` when no outbound market-data line has been sent in that interval.
- Client may issue `PING` anytime; gateway must respond with `PONG`.
- If no inbound or outbound traffic occurs for `idle_timeout_sec`, gateway sends `EXIT|REASON=idle_timeout` and closes the connection.
- `HB`, `PING`, and `PONG` are liveness messages and do not participate in `(CH,SYM)` sequence counters.

5.11 Session lifecycle



5.12 Gateway behavior requirements

For CALF `1.0.0` interoperability, `pm-md-gwy` must:

1. Accept TCP clients and enforce HELLO-before-use semantics.
2. Normalize internal engine events into CALF lines.
3. Maintain independent sequence counters per `(CH, SYM)`.
4. Keep bounded replay buffers per `(CH, SYM)` stream.
5. Auto-send `SNAP` on new `TOP`/`STATE`/`INDEX`/`DEPTH`/`CB` subscriptions; for a wildcard `TOP` subscrip-

tion, auto-send one real per-symbol `SNAP` for every currently known symbol rather than a single `SYM=*` snapshot.
6. Enforce channel and symbol rules deterministically, including which channels accept `SYM=*` (`STATE`, `TOP`, `TRADE`, `AUCTION` only).
7. Advertise supported channels in `WELCOME|CH_SUPPORTED=`.
8. Disconnect slow clients when queue limits are exceeded.

5.13 Configuration reference

CALF gateway settings are part of the main engine configuration file (`engine_config.yaml`) as a top-level `market_data_gateway` block.

Path location:

- `engine_config.yaml -> market_data_gateway`

All supported CALF `1.0.0` configuration fields are listed below.

Field	Type / allowed range	Default	Description
`market_data_gateway.enabled`	Boolean (`true`/`false`)	`true` (recommended when CALF is used)	Enables/disables the CALF gateway process configuration.
`market_data_gateway.name`	Non-empty string	Implementation-defined	Gateway instance name advertised in `WELCOME|GW=...`.
`market_data_gateway.bind_address`	IP/host bind string	`0.0.0.0` (common)	Local interface address to bind for incoming TCP clients.
`market_data_gateway.port`	Integer, `1..65535`	`5570`	TCP listen port for CALF clients.
`market_data_gateway.heartbeat_interval_sec`	Integer, `>= 0`	`1`	Interval used to emit `HB` when no outbound market-data line was sent.
`market_data_gateway.idle_timeout_sec`	Integer, `> 0`	`5`	Maximum silent period (no inbound and no outbound traffic) before disconnect.
`market_data_gateway.replay_window_sec`	Integer, `>= 0`	`30`	Time-bounded replay retention per `(CH,SYM)` stream for resume/gap recovery.
`market_data_gateway.max_connections`	Integer, `> 0`	`64`	Maximum concurrent TCP client connections accepted by the gateway.

Field	Type / allowed range	Default	Description
`market_data_gateway.max_messages_per_second`	Integer, > 0	`200`	Per-client inbound token-bucket rate limit; excess messages receive `ERR|CODE=RATE_LIMITED` and are dropped without disconnecting the client.
`market_data_gateway.max_symbols_per_client`	Integer, > 0	`200`	Per-client subscription symbol limit across active subscriptions. A wildcard subscription counts as one entry.
`market_data_gateway.max_client_queue`	Integer, > 0	`10000`	Per-client outbound queue cap; overflow triggers `ERR|CODE=SLOW_CLIENT` and disconnect.
`market_data_gateway.depth_levels`	Integer, > 0	`10`	Number of price levels per side included in `DEPTH`/`SNAP(CH=DEPTH)` messages. There is no separate enable/disable flag — `DEPTH` is on by default in CALF `1.0.0`; this only tunes ladder depth.

Operational notes:

- `HBINT` in `WELCOME` must reflect `heartbeat_interval_sec`.
- `REPLAY` in `WELCOME` must reflect `replay_window_sec`.
- `max_connections` bounds concurrent TCP clients; connections beyond this limit are rejected at accept time.
- `max_messages_per_second` affects inbound `ERR|CODE=RATE_LIMITED` behavior; it is a non-terminal, per-client token bucket, unlike `max_client_queue`.
- `max_symbols_per_client` affects `SUB` validation and `ERR|CODE=SUB_LIMIT`.
- `max_client_queue` controls slow-client backpressure behavior.
- `depth_levels` affects `DEPTH` message size and bandwidth; lower it on bandwidth-constrained deployments rather than expecting clients to request fewer levels — there is no per-client `LEVELS=` override in CALF `1.0.0`.

Example:

```
market_data_gateway:
  enabled: true
  name: "md-gwy01"
  bind_address: "0.0.0.0"
  port: 5570
  heartbeat_interval_sec: 1
  idle_timeout_sec: 5
  replay_window_sec: 30
  max_connections: 64
  max_messages_per_second: 200
  max_symbols_per_client: 200
  max_client_queue: 10000
  depth_levels: 10
```

5.14 What to watch out for during implementation

- Implement line buffering correctly for TCP streams (`recv()` may return partial lines or multiple lines at once).
- Enforce HELLO-before-use strictly; all non-HELLO pre-auth messages must receive `ERR|CODE=AUTH_REQUI`
- Keep `SNAP` semantics explicit: it is a message type, not a subscribable channel; `TOP`, `STATE`, `INDEX`, `DEPTH`, and `CB` subscriptions auto-trigger `SNAP` — `TRADE` and `AUCTION` never do.
- For a wildcard `TOP` subscription, do not call the per-symbol snapshot builder with a literal `SYM="*"` — it has no meaningful per-symbol state and will silently produce an empty snapshot. Iterate known symbols and send one real `SNAP` each.
- Enforce `SYM=*` constraints exactly (`STATE`, `TOP`, `TRADE`, `AUCTION` only — never `INDEX`, `DEPTH`, or `CB`, and never when mixed with any of those three in the same `SUB`) and validate multi-value `SUB` as Cartesian stream requests.
- `DEPTH` is a full-ladder replace per message, not a per-level diff like `MD`. Do not attempt incremental per-level patching on either the gateway or client side.
- `CB` and `STATE` are emitted from the same underlying `circuit_breaker.halt.*`/`circuit_breaker.resume.*` engine event — do not let `CB`'s richer detail leak into `STATE`'s field set, and do not let a `CB` normaliser failure suppress the `STATE` emission (or vice versa); both should be independent `_emit_stream_event` calls from the same handler.
- Carry the halt's cause on a single wire key, `SRC`, in `CB` — do not propagate the internal inconsistency to clients.
- Track sequence numbers independently per `(CH,SYM)` stream; never use a single global counter.
- Treat `RESUME` as single-stream only and validate `CH`, `SYM`, and `LASTSEQ` strictly.
- Bound replay by configured window and emit deterministic `REPLAY_MISS` behavior when outside window — with a fresh `SNAP` only on the channels that have one (`TOP`/`STATE`/`INDEX`/`DEPTH`/`CB`), never on `TRADE`/`AUCTION`.
- Apply slow-client backpressure deterministically: queue overflow must produce `SLOW_CLIENT` and disconnect.
- Keep liveness signals (`HB`, `PING`, `PONG`) outside market-data sequencing; they do not consume `(CH,SYM)` sequence numbers.

5.15 Conformance notes

If you are implementing a CALF client, the most important protocol truths are:

1. CALF is line-based text over TCP, not message-framed datagrams.
2. `HELLO` is mandatory before any subscription command.
3. `SNAP` is a message type, not a subscribable channel.

4. Sequence tracking is per `(CH, SYM)` stream.
5. `SYM=*` is valid for `STATE`, `TOP`, `TRADE`, and `AUCTION` subscriptions — never for `INDEX`, `DEPTH`, or `CB`.
6. A wildcard `TOP` subscription never yields a `SNAP` with a literal `SYM=*`; it yields one real `SNAP` per known symbol.
7. Replay resume is single-stream per `RESUME`; send one per stream.
8. On replay miss, client must accept fresh `SNAP` and reset local baseline.
9. `DEPTH` messages replace a side's entire tracked ladder, never a single price level in isolation.
10. `WELCOME|CH_SUPPORTED=`, not `PROTO`, is how a client detects whether a gateway build supports `DEPTH`, `INDEX`, `AUCTION`, `CB`, or the `SYM=*` wildcard extension — `PROTO=CALF1` does not change across CALF `1.0.0`.
11. Heartbeats and ping/pong are separate liveness mechanisms.
12. A `SLOW_CLIENT` error indicates disconnect and reconnect is required.
13. Protocol values and keys are uppercase by convention and should be emitted uppercase for interoperability.
14. `CB` is always emitted alongside `STATE` for the same halt/resume engine event, never instead of it — a client that only wants the coarse transition can ignore `CB` entirely and keep using `STATE` exactly as before this extension.
15. `AUCTION` fires exactly once per uncross, including when there was no crossable interest — absence of `EQPX`/`IMBSIDE` signals "no cross" or "balanced," not a suppressed/missing event.

5.16 See also

In this book:

- CALF Protocol Spy (pm-calf-spy) — read-only CLI for inspecting the live wire format
- External Protocols Overview — ALF/BALF/CALF/RALF at a glance

Operator's Guide:

- Market Data Feed (CALF) — operational guide and client examples
- Risk Controls — circuit-breaker engine behavior behind the `CB` channel
- Market Index — auction uncross mechanics behind the `AUCTION` channel

Reference Manual:

- Processes, Environment, and Ports — where `pm-md-gwy` sits in the process model
- Engine Config Specification — `market_data_gateway` field law

CHAPTER 6

RALF Protocol Reference

Status: Normative. This appendix is the single source of truth for the RALF wire contract as implemented by `pm-ralf-gwy` (`ralf_gateway/`). For an operational, tutorial-style guide see Post-Trade Dissemination; for the gateway's configuration block see Engine Config Specification §6.4.

6.1 1. What RALF is

RALF (Reconciliation ALF) is EduMatcher's **text post-trade dissemination** protocol. It streams executed-trade and end-of-day events over TCP to external back-office consumers (clearing, drop-copy, audit). It is served by `pm-ralf-gwy`, which subscribes to the engine event bus and re-publishes derived events as RALF lines. RALF is **read-only**: clients subscribe and receive; they never submit orders.

6.2 2. Scope & conformance

This document defines the client-visible wire behaviour of RALF version token `RALF1`: framing, message set, fields, roles, sequencing, timeouts, and errors. The key words **MUST**, **MUST NOT**, **SHOULD**, and **MAY** are used per RFC 2119. A conforming client or gateway MUST implement the message set in §6 with the field constraints in §6 and §10. Behaviour not stated here is unspecified. Where the current implementation deviates from what a robust protocol would require, the deviation is called out with a **Known deviation** note and tracked in the project's suspected-bugs report.

6.3 3. Transport & session model

Property	Value
Transport	TCP
Default port	`5580` (`post_trade_gateway.port`)
Encoding	UTF-8 text lines
Delimiter	`\n`
Max line length	4096 bytes including newline
Version token	`RALF1`

Property	Value
Connection model	long-lived, one authenticated session per connection
Direction	gateway → client for all data; client → gateway for control only

A session proceeds: **connect → `HELLO` → `WELCOME` → one or more `SUB`/`UNSUB` → event stream + heartbeats → `EXIT`/disconnect**. A client MUST send `HELLO` before any other message; a non-`HELLO` first message is rejected with `ERR|CODE=AUTH_REQUIRED`. A connection that sends no `HELLO` within 5 seconds receives `EXIT|REASON=auth_timeout` and is closed. Once the gateway has sent the line that ends a session, it ignores anything more the client sends.

6.4 4. Wire format

Every message is one line:

```
MSGTYPE|KEY=VALUE|KEY=VALUE|...\n
```

- `MSGTYPE` MUST be uppercase ASCII from `A-Z`, `0-9`, `_`.
- Fields are `|`-delimited; each field is split into key/value on the **first** `=`.
- A parser MUST ignore unknown fields (forward compatibility) except where a field is required for validation.
- Values do not contain `|` or `\n`.

Reserved cross-message keys (present on most messages):

Key	Meaning
`CH`	Channel: `CLEARING`, `DROP_COPY`, or `AUDIT`
`SYM`	Symbol filter (control) or event symbol (data); * = all
`SEQ`	Channel-local monotonic message sequence (see §8)
`TS`	UTC ISO-8601 timestamp with milliseconds and `Z` suffix

6.5 5. Message catalog

Message	Direction	Purpose
`HELLO`	client → gw	open session; authenticate role; optional replay checkpoint
`WELCOME`	gw → client	session accepted; negotiated parameters
`SUB`	client → gw	subscribe channel(s) × symbol scope
`UNSUB`	client → gw	remove subscriptions

Message	Direction	Purpose
`SNAP`	gw → client	subscription/recovery baseline (carries current `SEQ`)
`EXEC`	gw → client	one executed trade (post-trade view)
`EOD`	gw → client	end-of-day per-symbol summary marker
`HB`	gw → client	heartbeat when the stream is quiet
`PING` / `PONG`	client → gw / gw → client	liveness probe and reply
`EXIT`	both	from the gateway: session terminated, with `REASON` (e.g. `idle_timeout`); from the client: close after flushing
`ERR`	gw → client	protocol, entitlement, or replay error

6.6 6. Message definitions

6.6.1 `HELLO` (client → gateway)

Field	Type	Req	Constraints
`CLIENT`	string		external client identifier (logging)
`PROTO`	string		MUST equal `RALF1`
`ROLE`	enum		one of `CLEARING`, `DROP_COPY`, `AUDIT`; MUST be in the gateway's `allowed_roles`
`LASTSEQ`	int	–	replay checkpoint; requests events with `SEQ > LASTSEQ` (see §8)

Missing `CLIENT`/`PROTO`/`ROLE` → `ERR|CODE=AUTH_REQUIRED`. A `ROLE` outside `allowed_roles` → `ERR|CODE=ENTITLEMENT` A non-integer `LASTSEQ` → `ERR|CODE=BAD_MESSAGE`.

6.6.2 `WELCOME` (gateway → client)

Field	Meaning
`PROTO`	negotiated version (`RALF1`)
`GW`	gateway id (`post_trade_gateway.name`)
`ROLE`	accepted role
`REPLAY`	replay retention window, seconds (`replay_retention_sec`)
`HBINT`	heartbeat interval, seconds (`heartbeat_interval_sec`)

6.6.3 `SUB` / `UNSUB` (client → gateway)

Field	Type	Req	Default	Constraints
`CH`	csv of channels		—	each {`CLEARING`,`DROP_COPY`,`AUDIT`}; entitlement checked (§7)
`SYM`	csv of symbols or *	–	*	scope filter

A subscription is the set of `(channel, symbol)` pairs. `SUB` is cumulative; `UNSUB` removes matching pairs and is idempotent. After a `SUB`, the gateway emits one `SNAP`. A missing/empty `CH` → `ERR|CODE=INVALID_CHANNEL`; an unknown channel → `ERR|CODE=INVALID_CHANNEL`; a channel the role may not access → `ERR|CODE=ENTITLEMENT_DENIED`.

6.6.4 `SNAP` (gateway → client)

Baseline emitted after a `SUB` and on replay-miss recovery. Carries `CH` (the subscribed channels, comma-joined), `SYM`, `SEQ` (the highest current sequence over the `CH` set in that `SNAP`), and `TS`. RALF `SNAP` carries **no book/position state** — it is a sequence baseline only.

6.6.5 `EXEC` (gateway → client)

One line per executed trade, per subscribed channel.

Field	Description
`CH SYM SEQ TS`	envelope (§4)
`EXEC_ID`	execution id (engine trade id)
`MATCH_ID`	match id — **identical to `EXEC_ID`** in `RALF1`
`BUY_ORDER_ID` / `SELL_ORDER_ID`	resting/aggressing order ids
`BUY_GW` / `SELL_GW`	buy-side / sell-side gateway ids
`SIDE`	aggressor side (`BUY`/`SELL`)
`QTY`	executed quantity
`PX`	execution price (display units)

A single trade is emitted **once per channel** (`CLEARING`, `DROP_COPY`, `AUDIT`), each copy consuming its own `SEQ` (§8).

6.6.6 `EOD` (gateway → client)

Per-symbol end-of-day marker, derived from the engine `system.eod` broadcast.

Field	Description
`CH SYM SEQ TS`	envelope
`TRADE_COUNT`	trades executed for the symbol since the previous `EOD` (the counter resets to zero at every `EOD`, not just once per session)

Field	Description
EXEC_COUNT	execution count — **identical to TRADE_COUNT** in RALF1

EOD is emitted once per symbol on all three channels: CLEARING, DROP_COPY, and AUDIT.

6.6.7 HB, PING/PONG, EXIT

HB|TS=… is sent to every authenticated client every HBINT seconds. A client MAY send PING; the gateway replies PONG|TS=…. EXIT|REASON=…|TS=… signals gateway-initiated termination (idle_timeout, auth_timeout). A client MAY send a bare EXIT to end its session; the gateway closes after flushing — as pm-md-gwy and pm-dc-gwy do.

6.7 7. Roles & entitlements

Roles and channels share the same three names. Entitlement rules:

Role	May subscribe to
CLEARING	CLEARING only
DROP_COPY	DROP_COPY only
AUDIT	**any** channel (CLEARING, DROP_COPY, AUDIT)

A role subscribing outside its entitlement receives ERR|CODE=ENTITLEMENT_DENIED. Only roles listed in post_trade_gateway.allowed_roles may authenticate.

6.8 8. Sequencing & recovery

SEQ is maintained **per channel** (CLEARING, DROP_COPY, AUDIT).

- Within one channel, SEQ is strictly increasing by 1 per emitted line.
- Across different channels, SEQ values are independent and are not globally ordered or globally unique.
- Clients MUST do gap detection and ordering **within a given CH stream**, not across mixed channels.

LASTSEQ in HELLO is a single integer checkpoint. The gateway applies that same threshold to each entitled channel independently and replays events where event.SEQ > LASTSEQ for that event's channel.

Replay. If HELLO carries LASTSEQ, the gateway attempts to resend journalled events with SEQ > LASTSEQ (per channel, as above) and only for channels the role is entitled to receive (§7). If LASTSEQ predates the retained journal window for any entitled channel (replay_retention_sec) — including a channel whose journal has aged out entirely — or is ahead of any entitled channel, as after a gateway restart, the gateway emits ERR|CODE=REPLAY_MISS followed by a SNAP baseline; the client MUST reset to that baseline.

⚠ Known deviation — replay is entitlement-filtered but not subscription-filtered

Replay happens during HELLO, before any SUB is processed, so replay cannot apply symbol/channel subscription state from the current connection. The current gateway filters replay by role entitlement (§7), but does **not** filter by subscriptions.

6.9 9. Liveness & timeouts

Parameter	Source	Default
Heartbeat interval (`HBINT`)	`heartbeat_interval_sec`	2 s
Idle timeout	`idle_timeout_sec`	20 s
Replay retention (`REPLAY`)	`replay_retention_sec`	86400 s

The gateway sends `HB` to every authenticated client every heartbeat interval, whether or not other lines were sent in between. The idle timeout counts traffic in **both** directions — a line received, or a line the client actually took off the socket — so a client that keeps reading is never idle: the heartbeats alone keep it alive. In practice the idle timeout fires only when writes to a client have stopped succeeding; it then sends `EXIT|REASON=idle_timeout`. A client whose outbound queue exceeds `max_client_queue` receives `ERR|CODE=SLOW_CLIENT` and is disconnected.

6.10 10. Error codes

Code	Meaning / trigger
`AUTH_REQUIRED`	first message was not `HELLO`, or `HELLO` missing `CLIENT`/`PROTO`/`ROLE`
`BAD_MESSAGE`	unparseable line, unsupported message type, or malformed field (e.g. non-integer `LASTSEQ`); an oversized line (`DETAIL=line exceeds 4096 bytes`) also ends the session
`INVALID_CHANNEL`	`SUB` with missing/empty or unknown `CH`
`ENTITLEMENT_DENIED`	role not in `allowed_roles`, or channel outside the role's entitlement
`REPLAY_MISS`	requested `LASTSEQ` is older than the retained journal window, or ahead of the channel (e.g. after a gateway restart)
`SLOW_CLIENT`	outbound queue exceeded `max_client_queue`; session disconnected

`ERR` carries `CODE` and a human-readable `DETAIL`, e.g. `ERR|CODE=REPLAY_MISS|DETAIL=requested seq outside retention`.

6.11 11. Configuration reference

`pm-ralf-gwy` reads the `post_trade_gateway` block of `engine_config.yaml` (consumed only by this process). Full field law is in Engine Config Specification §6.4.

Key	Default	Purpose
`name`	`ralf-gwy01`	gateway id in `WELCOME|GW=`
`bind_address`	`0.0.0.0`	TCP listen address
`port`	`5580`	TCP listen port
`replay_retention_sec`	`86400`	journal window; advertised as `WELCOME|REPLAY=`

Key	Default	Purpose
`heartbeat_interval_se`	`2`	`HB` cadence; advertised as `WELCOME|HBINT=`
`idle_timeout_sec`	`20`	disconnect after this long with no traffic in either direction
`max_connections`	`64`	simultaneous TCP connections; one beyond this is accepted and closed at once
`max_client_queue`	`10000`	outbound backlog before `SLOW_CLIENT`
`allowed_roles`	`[CLEARING, DROP_COPY, AUDIT]`	roles permitted to authenticate

6.12 12. Conformance notes

A conforming `RALF1` implementation:

- derives events solely from engine broadcasts: `trade.executed` → `EXEC`, `system.eod` → `EOD`. No other event families exist in `RALF1`.
- MUST send `HELLO`/`WELCOME` before any data; MUST reject pre-auth data with `AUTH_REQUIRED`.
- MUST enforce the §7 entitlement matrix on `SUB` (live path).
- MUST apply per-channel `SEQ` semantics from §8 (no global `SEQ` stream).
- MUST advertise `REPLAY`/`HBINT` in `WELCOME` matching its configuration.
- SHOULD treat replay as role-entitled but pre-`SUB` (therefore not subscription-filtered) on reconnect.

Future revisions may add `CORRECT`, `BUST`, `ALLOC`, and `SETTLE` event families as the corresponding engine events become available; they are **not** part of `RALF1`.

6.13 See also

In this book:

- RALF Protocol Spy (pm-ralf-spy) — read-only CLI for inspecting the live wire format
- External Protocols Overview — ALF/BALF/CALF/RALF at a glance

Operator's Guide:

- Post-Trade Dissemination — operational guide and client examples

Reference Manual:

- Processes, Environment, and Ports — where `pm-ralf-gwy` sits
- Engine Config Specification — `post_trade_gateway` field law

CHAPTER 7

LALF Protocol Reference

Status: Normative. This appendix is the single source of truth for the LALF `1` wire contract (`PROTO=LALF1`) as implemented by `pm-log-srv` (`edumatcher.log_srv`). For an operational, tutorial-style guide — how to start the server, how to use `pm-log-cli`, and worked query examples — see Centralized Log Server. The key words MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY are used per RFC 2119. This appendix follows the same conventions as the normative ALF Protocol Reference and CALF Protocol Reference, to which it should be read as a sibling, not a subset.

7.1 What LALF is

LALF stands for **Logging ALF**.

LALF is EduMatcher's centralized-logging transport protocol: a small, newline-delimited, line-oriented TCP protocol that every `pm-*` process can use to ship its own operational log records to a single collector process, `pm-log-srv`. Like its siblings, a LALF client establishes one long-lived TCP connection, exchanges a `HELLO`/`WELCOME` handshake, and then streams typed, `KEY=VALUE`-framed messages until the connection ends.

LALF complements the other application protocols:

Protocol	Purpose
ALF	Text order entry (interactive)
BALF	Binary order entry (low-latency programmatic)
CALF	Channelized text market data
RALF	Post-trade dissemination
LALF	Centralized process logging

LALF departs from its siblings in exactly one structural respect, required by what it carries: every LALF client is a producer of exactly one thing — its own process's log records — so LALF is unidirectional and subscription-free where ALF/CALF/RALF are request/response or publish/subscribe. "Scope & conformance" and "Wire format" below state precisely what this means for conformance.

This appendix is the **normative reference** for LALF `1` semantics.

7.2 Scope & conformance

LALF is the transport between any `pm-*` process and `pm-log-srv`, the centralized log collector. It is not exposed to end users or external bots the way CALF/ALF are — it exists purely so every process's logging lands in one queryable place (`log.db`).

7.2.1 Supported in LALF 1

- One TCP connection per client process, carrying that process's own log records only, from `HELLO` to session end.
- Nine message types: `HELLO`, `WELCOME`, `LOG`, `ACK`, `ERR`, `HB`, `PING`, `PONG`, `EXIT`.
- A fixed, non-extensible-in-this-revision `LOG` header field set plus an explicit `LEN`-prefixed binary-safe payload.
- Server-side backpressure via TCP flow control; no application-level flow-control message type.

7.2.2 Out of scope in LALF 1

- Replay, resume, or any form of gap recovery on reconnect. A reconnecting client MUST start a new session (fresh `HELLO`, `SEQ` restarting at `1`) and MUST NOT expect the server to have retained anything about a prior connection beyond what is durably stored in `log_events`.
- Any read/query path on the LALF socket itself. A LALF server MUST NOT expose a mechanism for retrieving previously stored records over this protocol — that is `pm-log-cli`'s job, reading `log.db` directly. See Centralized Log Server.
- Authentication, authorization, and transport encryption. LALF assumes the same trusted-network posture as CALF (see the normative CALF Protocol Reference).
- Multi-host routing, service discovery, or multi-server fan-out. A LALF client speaks to exactly one `pm-log-srv` at a time, at a statically configured `host:port`.

7.3 Transport and session model

Property	Value
Transport	TCP, one connection per client process
Default port	`5600`
Encoding	UTF-8 for all header lines and payload bytes
Header line delimiter	`\n` (LF); a bare LF terminates every header line, matching ALF/CALF
Payload framing	Explicit byte count (`LEN=<n>`), not delimiter-based
Max header line length	4096 bytes including the terminating `\n`
Max payload length	`max_message_bytes`, default 65536 bytes; a server MUST truncate (not reject) an oversized payload and MUST record `truncated=1` for the stored row
Handshake timeout	A server MUST close the connection if no valid `HELLO` is received within 5 seconds of accept

Property	Value
Heartbeat interval	`HBINT` seconds, server-assigned in `WELCOME`, default `5`
Idle/dead-connection timeout	A server MUST treat a connection as dead, and close it, if no message of any kind (including `HB`) arrives within `2 × HBINT` seconds
Session cardinality	Exactly one `HELLO`/`WELCOME` per TCP connection; a second `HELLO` on an already-established connection is a protocol violation (`ERR|CODE=PROTO_MISMATCH`)
Direction	`LOG`, `HB`, and `EXIT` are client → server only; `WELCOME` is server → client only; `ERR` is server → client only; `PING`/`PONG` are wire-symmetric (either endpoint MAY send `PING`, the receiver MUST reply `PONG`), though the shipped `pm-log-srv` implementation never initiates a `PING` itself — it only replies to a client-sent one
Subscriptions	None. LALF has no `SUB`/`UNSUB`, no channel model, and no `SYM=`-style filtering — every connected client's data is, by construction, exactly its own log stream

7.4 Wire format

7.4.1 Line structure

Every LALF message begins with a header line of the form:

```
<MSGTYPE>|KEY=VALUE|KEY=VALUE|...\n
```

identical in shape to a CALF line: `MSGTYPE` is an uppercase token, each subsequent field is a `KEY=VALUE` pair separated by `|`, and the line is terminated by a single `\n`. A message type that carries a payload (only `LOG`, in this revision) MUST include a `LEN` field as its final `KEY=VALUE` pair, whose value is the exact number of UTF-8 bytes that immediately follow the header line's terminating `\n`. A receiver MUST read exactly `LEN` bytes as the payload — it MUST NOT scan those bytes for a delimiter, and the payload MUST NOT be assumed to end with its own trailing `\n`.

This exists because a log message is arbitrary text and cannot be assumed free of `|`, `\n`, or any other byte value that would collide with a delimiter-based grammar the way CALF's `KEY=VALUE|KEY=VALUE` grammar can assume for its own constrained field values (symbols, prices, enums).

```
HELLO|CLIENT=pm-api-gwy|PID=48213|HOST=trader-laptop|PROTO=LALF1
LOG|SEQ=1042|TS=2026-07-28T14:32:07.511Z|LEVEL=WARNING|LOGGER=edumatcher.md_gateway.gateway|LEN=57
slow client detected on channel DEPTH, symbol AAPL, dropping
```

7.4.2 Parsing behavior

A conforming parser MUST:

1. Read bytes up to and including the next `\n` as one header line.
2. Split the header line on `|`; the first token is `MSGTYPE`, every subsequent token MUST parse as

`KEY=VALUE` (split on the first `=`).

3. If the parsed fields include `LEN`, read exactly that many further bytes as the payload before considering the message complete and before parsing the next header line. A connection that closes before `LEN` bytes have arrived MUST be treated as an incomplete message, not a valid zero-length one.
4. If the parsed fields do not include `LEN`, the message has no payload; the parser proceeds directly to the next header line.
5. Reject (via `ERR`) any header line exceeding 4096 bytes before its terminating `\n` is found, any unrecognized `MSGTYPE`, and any message missing a field marked required below.

7.4.3 TCP stream requirement

TCP is a byte stream, not a message queue. A LALF implementation MUST treat the connection as one continuous byte stream and MUST NOT assume any correspondence between `send()`/`recv()` calls on either side and logical message boundaries. A single `recv()` may deliver part of a header line, a whole message, or several messages concatenated; a conforming implementation buffers and re-frames accordingly, exactly as required of ALF/CALF/RALF implementations.

7.5 Message catalog

Message	Direction	Purpose
`HELLO`	Client -> Server	Start session; identifies the connecting process
`WELCOME`	Server -> Client	Confirm session and advertise parameters
`LOG`	Client -> Server	One formatted log record, header + payload
`ACK`	Server -> Client	Reserved for future use — not sent in this revision
`ERR`	Server -> Client	Protocol violation or advisory condition
`HB`	Client -> Server	Heartbeat, sent whether or not `LOG` was recently sent
`PING`	Either direction	On-demand liveness check
`PONG`	Either direction	Reply to `PING`
`EXIT`	Client -> Server	Clean disconnect

7.6 Message definitions

7.6.1 `HELLO`

Direction: Client -> Server, exactly once, as the first message on a new connection.

Purpose: identifies the connecting process and negotiates protocol version; MUST precede any other message type.

Field	Req	Type	Description
`CLIENT`	Yes	string	Process name, matching the connecting `pm-*` command name

Field	Req	Type	Description
`PID`	Yes	int	OS process ID of the connecting client
`HOST`	Yes	string	Hostname the client is running on
`PROTO`	Yes	string	MUST be `LALF1` in this revision
`INSTANCE`	No	string	Disambiguator when multiple instances of the same `CLIENT` connect concurrently

```
HELLO|CLIENT=pm-api-gwy|PID=48213|HOST=trader-laptop|PROTO=LALF1
```

A server receiving a second `HELLO` on a connection that has already completed a handshake MUST respond `ERR|CODE=PROTO_MISMATCH` and close the connection. A `HELLO` missing `CLIENT`, `PID`, `HOST`, or `PROTO` MUST be rejected with `ERR|CODE=MISSING_FIELD`; a non-integer `PID` is rejected the same way. A non-`HELLO` message sent as the *first* message on a new connection — before any handshake has completed — MUST be rejected with `ERR|CODE=AUTH_REQUIRED`, and the connection closed.

7.6.2 `WELCOME`

Direction: Server -> Client, exactly once, in direct reply to a valid `HELLO`.

Purpose: confirms the session is established and communicates server-assigned session parameters the client MUST honor for the remainder of the connection.

Field	Req	Type	Description
`PROTO`	Yes	string	Echoes `LALF1`
`SRV`	Yes	string	Configured name of the responding `pm-log-srv` instance
`HBINT`	Yes	int	Heartbeat interval in seconds; the client MUST send `HB` at least this often
`SESSION`	Yes	string	Opaque per-connection session identifier, included in every stored row for this connection; not a security token

```
WELCOME|PROTO=LALF1|SRV=log-srv01|HBINT=5|SESSION=a1b2c3d4
```

A `HELLO`/`WELCOME` exchange is a valid use of the protocol on its own: a client may complete the handshake purely to discover whether a server is listening, then disconnect without logging. The

shipped `pm-log-srv` therefore does not enter a session into its `processes` registry until the first `LOG` or `HB` frame arrives, so such probes leave no record of a process that never ran.

A `WELCOME` is the sole positive acknowledgment of a `HELLO`; there is no separate `ACK` for the handshake.

7.6.3 `LOG`

Direction: Client -> Server, any number of times after `WELCOME`. This is the only message type in LALF that carries a payload, and the only message type a conforming server is required to durably store.

Purpose: carries one formatted log record — header fields plus a length-prefixed message body.

Field	Req	Type	Description
`SEQ`	Yes	int	Monotonic per-connection sequence number, starting at 1
`TS`	Yes	string	UTC ISO-8601 timestamp with milliseconds, set by the client at emission time (`LogRecord.created`)
`LEVEL`	Yes	enum	One of `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`
`LOGGER`	Yes	string	Originating logger name (`LogRecord.name`)
`MODULE`	No	string	`LogRecord.module`, when available
`LINE`	No	int	`LogRecord.lineno`, when available
`EXC`	No	bool (`1` or absent)	Set when the payload includes a formatted exception/traceback
`LEN`	Yes	int	Byte length of the UTF-8 payload following this header line's `\n`

```
LOG|SEQ=1042|TS=2026-07-28T14:32:07.511Z|LEVEL=WARNING|LOGGER=edumatcher.md_gateway.gateway|LEN=57
slow client detected on channel DEPTH, symbol AAPL, dropping
```

A server MUST reject (`ERR|CODE=INVALID_LEVEL`) any `LOG` whose `LEVEL` is not one of the five listed values, and MUST reject (`ERR|CODE=MISSING_FIELD`) any `LOG` missing `SEQ`, `TS`, `LEVEL`, `LOGGER`, or `LEN`. Neither rejection is session-ending — the client SHOULD fix the sending code and MAY continue sending subsequent `LOG` messages on the same connection.

A server MUST NOT reject a `LOG` solely for exceeding `max_message_bytes`; it MUST instead truncate the stored payload to that limit (never splitting a multi-byte UTF-8 codepoint), record `truncated=1`,

and MAY send an advisory `ERR|CODE=PAYLOAD_TOO_LARGE` — this is the one `ERR` code in this revision that does not imply the client did anything requiring a fix before continuing.

7.6.4 `ACK`

Direction: Server -> Client.

Purpose: reserved for future use. A server MUST NOT send `ACK` for individual `LOG` messages in this revision — `LOG` is fire-and-forget by design, and per-message acknowledgment would be pure overhead at realistic logging rates. `HELLO` success is communicated exclusively via `WELCOME`; no separate `ACK` follows it. `ACK` is defined here only so its wire shape is reserved and unambiguous for a future revision that adds a request/response exchange to LALF. The shipped `pm-log-srv` never sends this message type.

```
ACK|SEQ=1042
```

Field	Req	Type	Description
`SEQ`	Yes	int	Echoes the `SEQ` of the message being acknowledged

7.6.5 `ERR`

Direction: Server -> Client.

Purpose: reports a protocol violation or an advisory condition on a message the client sent. Whether `ERR` precedes a connection close depends on the code (see table).

```
ERR|CODE=INVALID_LEVEL|MSG=unknown LEVEL value: TRACE
```

Field	Req	Type	Description
`CODE`	Yes	enum	One of the codes below
`MSG`	Yes	string	Free-text, human-readable detail; MUST NOT be parsed programmatically beyond logging/display

Code	Meaning	Session-ending?
`AUTH_REQUIRED`	First message on the connection was not `HELLO`	Yes — server MUST close the connection after queuing this `ERR`
`INVALID_LEVEL`	`LOG.LEVEL` not one of the five valid values	No — indicates a client bug; client SHOULD fix and MAY continue sending subsequent `LOG` messages on the same connection

Code	Meaning	Session-ending?
`MISSING_FIELD`	A required header field absent on the message	No, same as `INVALID_LEVEL`
`PAYLOAD_TOO_LARGE`	`LOG.LEN` exceeded `max_message_bytes`	No — advisory only; the server has already truncated and stored the record
`PROTO_MISMATCH`	`HELLO.PROTO` was not `LALF1`, or a second `HELLO` was sent on an established connection	Yes — server MUST close the connection after queuing this `ERR`
`HELLO_TIMEOUT`	No `HELLO` received within 5 seconds of accept	Yes — the server queues this `ERR` and disconnects in the same step, without an intervening flush of the outbound queue; a client SHOULD NOT rely on receiving this `ERR` before the socket closes, and MUST instead treat an unexpected close with no prior `WELCOME` as an implicit handshake timeout

Implementation note on `HELLO_TIMEOUT` delivery. Unlike every other terminal `ERR` code above — which are queued and then the connection is marked closing-after-flush, guaranteeing the `ERR` bytes are sent before the socket closes — `HELLO_TIMEOUT` is queued and the connection is closed immediately afterward, in the same pass through the server's idle-connection sweep, with no guarantee the outbound queue was flushed first. In practice this means a client is unlikely to ever observe this specific `ERR` on the wire; it should be treated as effectively equivalent to a bare connection close with no prior `WELCOME`.

7.6.6 `HB`

Direction: Client -> Server, periodically. The server never sends `HB`; it is purely a client-liveness signal the server listens for.

Purpose: liveness signal sent at least every `HBINT` seconds (from `WELCOME`), whether or not the client has sent any `LOG` in that interval, so the server can distinguish a quiet-but-alive client from a dead connection.

```
HB|TS=2026-07-28T14:32:10.000Z
```

Field	Req	Type	Description
`TS`	Yes	string	UTC ISO-8601 timestamp with milliseconds, at the moment this `HB` was sent

The shipped `pm-log-srv` records the time of the most recent contact from each session in `processes.last_heartbeat_at`,

separately from `last_seen_at` which advances only on `LOG`. The two together let `pm-log-cli diagnose` tell a process with nothing to say apart from one that has stopped talking.

7.6.7 `PING` / `PONG`

Direction: wire-symmetric; either endpoint MAY send `PING` at any time after `WELCOME`, and the receiver MUST reply with `PONG` as soon as possible. The shipped `pm-log-srv` server implementation only ever replies to a client-initiated `PING` — it has no code path that sends `PING` itself. The symmetric grammar is reserved for a future operator tool that might want to probe a specific client connection's liveness on demand.

Purpose: an on-demand liveness check independent of the regular `HB` cadence.

```
PING
PONG
```

Neither message carries fields. A `PONG` is a direct, unsolicited-content reply to a `PING` and carries no correlation identifier in this revision (a connection has, at any time, at most one outstanding `PING` it is waiting on).

7.6.8 `EXIT`

Direction: Client -> Server.

Purpose: graceful, client-initiated end of session.

```
EXIT
```

On receiving `EXIT`, a server MUST flush any buffered rows already accepted for that connection before closing the socket. A client disconnecting without sending `EXIT` (process killed by signal, crash) is not a protocol violation; the server MUST treat the closed socket identically to an explicit `EXIT` for the purpose of flushing buffered rows and marking the connection's stored process record disconnected.

The shipped `pm-log-srv` does, however, record *which* of the two happened in `processes.disconnect_reason` (`client_exit` versus `peer_closed`, `idle_timeout`, `hello_timeout`, or `protocol_error`). Sending `EXIT` is therefore worth doing even though its absence is legal: it is what lets `pm-log-cli diagnose` tell a deliberate shutdown from a process that crashed or failed over to file logging.

7.7 Backpressure

LALF defines no application-level flow-control message. A server experiencing a write bottleneck MUST apply backpressure by ceasing to read from a connection's TCP receive buffer once its per-connection internal queue exceeds `max_client_queue`, relying on standard TCP flow control to slow the client's `send()` calls. A server MUST NOT silently discard a `LOG` message it has already accepted from the TCP layer solely due to write-path load; the sole exception is oversized-payload truncation (`PAYLOAD_TOO_LARGE`), which is a per-message content limit, not a load-shedding mechanism.

7.8 Sequence semantics

- `SEQ` is a monotonic per-**connection** counter starting at `1`, present on every `LOG` message.
- A reconnecting client MUST start a new session (fresh `HELLO`) and MUST restart `SEQ` at `1`; a server MUST NOT interpret a lower `SEQ` on a new connection as a gap or a duplicate of a prior connection's stream — `SESSION` (from `WELCOME`), not `SEQ` alone, is what disambiguates rows from different connections in storage.

- A server MAY log a client-visible gap in `SEQ` for diagnostic purposes (the reference `pm-log-srv` implementation does, at `DEBUG` level) but MUST NOT reject or otherwise treat a `SEQ` gap as a protocol violation — LALF defines no retransmission or replay mechanism, so a gap is purely informational.
- LALF defines no cross-connection replay or resume. There is nothing equivalent to CALF's `RESUME/LASTSEQ=` — a reconnecting client's records generated during the disconnect window were never buffered server-side to begin with.

7.9 Configuration reference

LALF's wire-level parameters are exposed as `pm-log-srv` configuration, read from an optional top-level `log_server` block in `engine_config.yaml`. See Configuration — Configuring pm-log-srv for the full field reference and example. They are not negotiated on the wire beyond `WELCOME.HBINT`:

Parameter	Wire effect	Config key
Listen port	Default port clients connect to	`log_server.port` (default `5600`)
Handshake timeout	5-second `HELLO` deadline	Fixed at 5s in this revision, not separately configurable
Heartbeat interval	Value sent in `WELCOME.HBINT`	`log_server.heartbeat_interval_sec` (default `5`)
Idle timeout	`2 × HBINT` dead-connection rule	Derived, not independently configurable
Max header line length	Header-line rejection ceiling	Fixed at 4096 bytes, matching CALF
Max payload length	`LOG.LEN` ceiling before truncation	`log_server.max_message_bytes` (default `65536`)
Per-connection queue limit	Backpressure trigger	`log_server.max_client_queue` (default `10000`)

7.10 What to watch out for during implementation

- **Do not treat `LOG`'s payload as line-oriented.** The single most common implementation mistake: reading the payload with a line-based `readline()`-style call instead of reading exactly `LEN` bytes will silently truncate any log message containing an embedded `\n` (i.e., any formatted traceback). Always read the payload as a fixed byte count.
- **`SEQ` is per-connection, not global.** A reconnecting client MUST restart `SEQ` at 1; a server MUST NOT interpret a lower `SEQ` on a new connection as a gap or a duplicate — `SESSION`, not `SEQ` alone, disambiguates rows from different connections in storage.
- **`ERR` is not always fatal.** Unlike `PROTO_MISMATCH`/`HELLO_TIMEOUT`, `INVALID_LEVEL`/`MISSING_FIELD`/`PAYLOAD_TOO_LAR` do not end the session — an implementation that closes the connection on every `ERR` will disconnect a client over one malformed `LOG` line instead of continuing to accept its subsequent, valid ones.
- **`pm-log-srv` itself never speaks LALF as a client to another instance.** It is the one `pm-*` process whose own logging configuration hard-codes stdout/file only — there is nothing paradoxical about this in the protocol itself, but it is easy to forget when wiring `TcpLogHandler` into every other entrypoint.

- **A missing `LEN` means no payload, not a zero-length one.** Only `LOG` carries `LEN` in this revision; every other message type MUST be parsed as header-only. An implementation that defaults to expecting a payload after every message type will stall waiting for bytes that are never sent.
- **Do not assume `HELLO_TIMEOUT` is reliably observable.** As noted under `ERR` above, the reference server implementation queues this `ERR` and closes the connection without guaranteeing a flush first. Client code should treat "connected but no `WELCOME` within a few seconds, then the socket closes" as the practical signal, not a specific received `ERR` code.

7.11 Conformance notes

If you are implementing a LALF client or server, the most important protocol truths are:

1. Every LALF session begins with exactly one `HELLO` and, on success, exactly one `WELCOME`; a second `HELLO` on the same connection is a protocol violation.
2. `LOG` is the only message type that carries a payload, and its `LEN` field is mandatory; every other message type is header-only with no `LEN` field.
3. Payload bytes are read by exact count, never scanned for a delimiter; they may contain any UTF-8 byte sequence, including `|`, `\n`, and embedded control characters, without escaping.
4. `LOG` is fire-and-forget: a server never sends a per-message `ACK` for it, and a client never waits for one before sending the next `LOG`.
5. Oversized payloads are truncated and stored, never dropped; only a malformed or incomplete message is refused outright.
6. `SEQ` numbering, and any gap-detection built on it, is scoped to a single TCP connection and MUST be reset on reconnect; LALF defines no cross-connection replay or resume.
7. A client MUST send `HB` at least every `HBINT` seconds; a server MUST consider a connection dead after `2 × HBINT` seconds of total silence (no message of any kind, `HB` included). `HB` is client-to-server only.
8. LALF has no subscription, filtering, or channel model of any kind — every connected client's data is exactly and only its own log stream.
9. LALF defines no read/query path; retrieval of stored records happens entirely outside this protocol, against `log.db` directly via `pm-log-cli`.
10. `PING`/`PONG` are wire-symmetric by grammar, but in the shipped implementation only a client ever initiates `PING` — the server only replies.

7.12 See also

In this book:

- ALF Protocol Reference — the normative sibling document this appendix's structure and conventions follow
- CALF Protocol Reference — the normative sibling document LALF's transport/session model and line-oriented framing are most directly modeled on
- External Protocols Overview — ALF/BALF/CALF/RALF at a glance

Operator's Guide:

- Centralized Log Server — operational guide: starting `pm-log-srv`, using `pm-log-cli`, and query cookbook
- Configuration — Configuring pm-log-srv — `log_server` field law

Reference Manual:

- Processes — where `pm-log-srv`/`pm-log-cli` sit in the process model

CHAPTER 8

REST and WebSocket API Reference

Status: Normative. This appendix is the single source of truth for the `pm-api-gwy` REST contract as exposed under `/api/v1`. For an operational, tutorial-style guide see API Gateway (REST/WebSocket). The key words MUST, MUST NOT, SHOULD, and MAY are used per RFC 2119.

8.1 Scope & conventions

This appendix documents the client-visible REST surface of the API gateway. It covers request shapes, replies, query parameters, auth rules, and the stable error codes returned by the gateway. WebSocket streams live under `/api/v1/ws/` (`private`, `market-data`, `admin/monitor`, `admin/sim`) and are specified normatively in the WebSocket API Reference; they are not repeated here.

All paths are rooted at `/api/v1`. Admin paths name their target as a path segment — `/admin/symbols/{symbol}/`, `/admin/gateways/{gateway_id}/...`, `/admin/market/...` — and a body, where one is accepted, carries only optional parameters.

8.2 Endpoint index

8.2.1 Bootstrap

Endpoint	Access	Purpose
GET /api/v1/bootstrap/trader	Any valid key	One-request startup payload for TRADER / MARKET_MAKER / ADMIN
GET /api/v1/bootstrap/mm	MARKET_MAKER key	One-request startup payload for MARKET_MAKER (adds quote state)
GET /api/v1/bootstrap/admin	Admin role	One-request startup payload for ADMIN

8.2.2 Trading REST

Endpoint	Access	Purpose
POST /api/v1/orders	Trading key	Submit one order for the caller's gateway
DELETE /api/v1/orders/{order_id}	Trading key	Cancel one live order
PATCH /api/v1/orders/{order_id}	Trading key	Amend price and/or quantity on one live order
POST /api/v1/orders/{order_id}/replace	Trading key	Cancel then replace one live order
GET /api/v1/orders	Trading key	Return the caller gateway's live order cache
GET /api/v1/orders/{order_id}	Trading key	Return one cached order
POST /api/v1/oco	Trading key	Submit an OCO pair
DELETE /api/v1/oco/{oco_id}	Trading key	Cancel an OCO pair
POST /api/v1/combos	Trading key	Submit a combo order
DELETE /api/v1/combos/{combo_id}	Trading key	Cancel a combo order
POST /api/v1/quotes	Trading key	Submit a two-sided market-maker quote
DELETE /api/v1/quotes/{symbol}	Trading key	Cancel the active quote for one symbol
POST /api/v1/mass-cancel	Trading key	Cancel all resting exposure for the caller or one symbol
GET /api/v1/symbols	Trading key	Return instrument metadata
GET /api/v1/news	Authenticated key	Return the latest market-sim headlines and sector map
GET /api/v1/companies/{symbol}	Authenticated key	Return one company profile from the company database
GET /api/v1/session	Authenticated key	Return the session phase, exchange timezone and trading date
GET /api/v1/halts	Authenticated key	Return currently-halted symbols
GET /api/v1/status	Trading key	Return the caller's gateway id and role
GET /api/v1/healthz	Public	Liveness probe
GET /api/v1/quotes	Trading key	Return the caller's active market-maker quotes
GET /api/v1/quotes/legs	Trading key	Return current quote-leg state

GET /api/v1/positions	Trading key	Return current net positions

8.2.3 Reference data

Endpoint	Access	Purpose
GET /api/v1/reference	Authenticated key	Return the full reference bundle
GET /api/v1/reference/config-version	Authenticated key	Return the reference bundle version
GET /api/v1/reference/symbols	Authenticated key	Return per-symbol tick and risk metadata
GET /api/v1/reference/risk	Authenticated key	Return risk-band definitions
GET /api/v1/reference/indexes	Authenticated key	Return configured exchange index definitions
GET /api/v1/reference/schedule	Authenticated key	Return session schedule metadata

8.2.4 Market data

Endpoint	Access	Purpose
GET /api/v1/book	Authenticated key	Return best bid, best ask and last price for every symbol
GET /api/v1/book/{symbol}	Authenticated key	Return one symbol's book levels, last trade and recent prints

8.2.5 History

Endpoint	Access	Purpose
GET /api/v1/history/orders	Trading key	Return the caller gateway's order lifecycle events
GET /api/v1/history/orders/{order_id}	Trading key	Return the full lifecycle of one order
GET /api/v1/history/fills	Trading key	Return fill events
GET /api/v1/history/trades	Authenticated key	Return public trade tape rows
GET /api/v1/history/daily	Authenticated key	Return daily OHLCV rows
GET /api/v1/history/price-snapshots	Authenticated key	Return intraday price snapshots
GET /api/v1/history/index-daily	Authenticated key	Return daily index OHLC rows

Endpoint	Access	Purpose
GET /api/v1/history/index-snapshots	Authenticated key	Return intraday index snapshots
GET /api/v1/history/indexes	Authenticated key	List index ids with recorded data
GET /api/v1/history/index-events	Authenticated key	Return index structural and audit events

8.2.6 Admin REST

Endpoint	Access	Purpose
POST /api/v1/admin/session/transition	Admin role	Request a session-phase transition
GET /api/v1/admin/sim	Admin role	Market simulator state and its true fair values
GET /api/v1/admin/gateways	Admin role	List configured gateways and live connection state
POST /api/v1/admin/gateways/{gateway_id}/disconnect	Admin role	Forcibly disconnect one gateway
POST /api/v1/admin/symbols/{symbol}/halt	Admin role	Halt one symbol through the circuit breaker
POST /api/v1/admin/symbols/{symbol}/resume	Admin role	Resume one halted symbol
GET /api/v1/admin/risk/state	Admin role	Return live per-symbol risk state
GET /api/v1/admin/orders	Admin role	Return the cross-gateway active-order table
GET /api/v1/admin/orders/{order_id}/audit	Admin role	Return the full cross-gateway lifecycle of one order
GET /api/v1/admin/gateways/{gateway_id}/positions	Admin role	Return net position and average cost for any one gateway
POST /api/v1/admin/symbols/{symbol}/cancel-orders	Admin role	Cancel all resting exposure on one symbol
POST /api/v1/admin/gateways/{gateway_id}/kill-switch	Admin role	Cancel all resting exposure for one gateway
POST /api/v1/admin/market/halt	Admin role	Halt every symbol and cancel every quote
POST /api/v1/admin/market/resume	Admin role	Resume every symbol halted by a market halt

Endpoint	Access	Purpose
GET /api/v1/admin/market/volume	Admin role	Today's traded volume per symbol and in total
POST /api/v1/admin/symbols/{symbol}/reopen	Admin role	Uncross and reopen a symbol, optionally at an operator price
GET /api/v1/admin/symbols/{symbol}/orders	Admin role	Every resting order on a symbol, optionally one price level
GET /api/v1/admin/gateways/{gateway_id}/orders	Admin role	One gateway's orders as the engine holds them
DELETE /api/v1/admin/gateways/{gateway_id}/quotes/{symbol}	Admin role	Withdraw one gateway's quote on one symbol
POST /api/v1/admin/market/kill-switch	Admin role	Cancel all resting exposure across every gateway and symbol
POST /api/v1/admin/indexes/{index_id}/rebalance	Admin role	Rebalance one configured index
POST /api/v1/admin/reference/reload	Admin role	Request a reference-data reload (currently refused by the engine)

8.2.7 Auth and roles

- Trading and admin REST requests use `Authorization: Bearer <api_key>`.
- `gateway_id: null` credentials are read-only and may access public data — reference data, public history, `GET /session`, `GET /halts`, news and company profiles — but no gateway-scoped or admin endpoint.
- ADMIN endpoints require a credential whose resolved engine gateway role is `ADMIN`.

8.2.8 Key types and provisioning

This appendix uses two access labels in endpoint tables:

- `Authenticated key` (also called `Authentication key`): any configured API key accepted by `pm-api-gwy`. This includes both read-only and trading keys.
- `Trading key`: an authenticated key whose configured `gateway_id` is not `null`. This key is bound to one engine gateway identity and can submit and manage that gateway's orders.

How keys are specified:

- Keys are configured by operators in `engine_config.yaml` under `api_gateways.<INSTANCE>.credentials`.
- Each credential entry has `api_key` and `gateway_id`.
- `gateway_id: null` creates a read-only authenticated key.
- `gateway_id: <GW_ID>` creates a trading-capable key for that gateway.

Example:

```
api_gateways:
  default:
    credentials:
      - api_key: key-trader-demo
        gateway_id: TRADER01
        description: Trading client for TRADER01
      - api_key: key-dashboard-demo
        gateway_id: null
        description: Read-only dashboard client
```

How to get a key:

- API keys are provisioned out-of-band by the venue/operator team.
- There is no REST endpoint in `pm-api-gwy` to self-issue or rotate keys.
- After updating credentials, restart/redeploy the API gateway instance so the new key set is loaded.

8.2.9 Common error codes

Every non-2xx reply carries one of these `error.code` values. `reject_code` is the engine vocabulary shared with ALF; it is `UNKNOWN` where no engine reject applies.

Every endpoint can answer `AUTH`, `ENGINE_AUTH`, `ENGINE_UNAVAILABLE` (the engine could not be reached to confirm a trading key) and `INTERNAL_ERROR`. Each endpoint's **Errors** table lists exactly the other codes it can return; the OpenAPI document lists all of them per operation.

Code	HTTP status	`reject_code`	Meaning
`VALIDATION`	400	`MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`	A body field, query parameter or history filter (date, cursor, `event_type`) failed validation; `field` names it. A price off the instrument's tick grid carries `TICK_VIOLATION`
`AUTH`	401	`AUTH_REQUIRED`	Missing, malformed or unknown API key. The reply carries `WWW-Authenticate: Bearer`
`ENGINE_AUTH`	403	`AUTH_FAILED`	The engine refused the key's gateway identity
`READ_ONLY`	403	`ROLE_DENIED`	Read-only key used on a gateway-scoped endpoint

Code	HTTP status	`reject_code`	Meaning
`ROLE_DENIED`	403	`ROLE_DENIED`	The key's gateway lacks the role the endpoint needs (`ADMIN`, or `MARKET_MAKER` for `/bootstrap/mm`)
`NOT_FOUND`	404	`UNKNOWN`	No such path
`UNKNOWN_ORDER`	404	`UNKNOWN`	Order id unknown to the gateway (`GET /orders/{id}`) or absent from the audit trail (`GET /admin/orders/{id}`)
`UNKNOWN_COMPANY`	404	`UNKNOWN`	No company profile for the symbol
`UNKNOWN_SYMBOL`	404	`UNKNOWN`	The symbol is not in the venue's reference data (`GET /book/{symbol}`)
`METHOD_NOT_ALLOWED`	405	`UNKNOWN`	The path exists but not for this method; the reply carries `Allow`
`TRANSITION_REJECTED`	409	`UNKNOWN`	The engine refused the session transition
`RELOAD_REJECTED`	409	`UNKNOWN`	The engine refused the reference reload
`HALT_REJECTED`	409	`UNKNOWN`	The engine refused the halt (for example `Unknown symbol: XYZ`)
`RESUME_REJECTED`	409	`UNKNOWN`	The engine refused the resume (for example `TSLA is not halted`)
`KILL_SWITCH_REJECTED`	409	`UNKNOWN`	The engine refused a symbol, gateway or global kill switch
`REOPEN_REJECTED`	409	`UNKNOWN`	The engine refused a symbol reopen (force-uncross), for example the symbol is not halted

Code	HTTP status	`reject_code`	Meaning
`QUERY_REJECTED`	`409`	`UNKNOWN`	The engine refused an admin query, for example `Unknown symbol: XYZ`
`REBALANCE_REJECTED`	`409`	`UNKNOWN`	`pm-index` refused the rebalance
`REPLACE_REJECTED`	`409`	engine's	The engine refused `replace`'s cancel (for example the order already filled); nothing was sent
`IDEMPOTENCY_KEY_REUSED`	`409`	`UNKNOWN`	The `Idempotency-Key` was already used for a different request
`RATE_LIMIT`	`429`	`RATE_LIMITED`	Per-key write limit exceeded
`INTERNAL_ERROR`	`500`	`INTERNAL_ERROR`	A gateway bug; the message carries no internals
`INDEX_ERROR`	`502`	`UNKNOWN`	`pm-index` answered with an error
`ENGINE_TIMEOUT`	`503`	`INTERNAL_ERROR`	No engine reply in time to a read, or to `replace`'s cancel. Transient — retry. A write whose ack is late is not an error: `202` with `ack_timeout: true`
`ENGINE_UNAVAILABLE`	`503`	`INTERNAL_ERROR`	The command could not be handed to the engine (down or backpressured). Transient — retry
`SYMBOLS_NOT_READY`	`503`	`SYMBOL_NOT_READY`	Order entry before the symbol's tick precision reached the gateway. Transient — retry
`INDEX_TIMEOUT`	`503`	`UNKNOWN`	No `pm-index` reply in time
`STATS_DB`	`503`	`UNKNOWN`	The `pm-stats` database is not present

Code	HTTP status	reject_code	Meaning
AUDIT_INDEX_UNAVAILABLE	503	UNKNOWN	The pm-audit index is not built or not running
COMPANY_DB_UNAVAILABLE	503	UNKNOWN	The company database is missing or unreadable

For an engine refusal (409), message is the engine's own reason. New codes may be added within v1; clients must treat an unknown code by its HTTP status.

8.2.10 Swagger/OpenAPI documentation

The API gateway supports live API documentation through FastAPI:

- OpenAPI schema: /openapi.json
- Swagger UI: /docs

The same document is committed in the repository as spec/rest/openapi.json, so a client can be generated or reviewed without a running gateway. It does not depend on deployment configuration: every v1 gateway publishes exactly that document, and CI fails if the code and the committed copy disagree.

These endpoints are available when swagger_enabled: true in the api_gateways configuration. When disabled, /docs and /openapi.json are not exposed.

8.2.11 Common reply shapes

- 200 OK means the response body carries the outcome: every read, and every command the gateway waited for (admin commands, POST /mass-cancel, and any order, cancel, amend, replace, OCO, combo or quote write with ?wait=ack whose verdict arrived in time).
- 202 Accepted means the request was handed on and the outcome follows on a WebSocket: those writes without ?wait=ack (or with it, when the verdict was late: ack_timeout: true), and POST /admin/gateways/{gateway_id}/disconnect.
- Pagination endpoints return count, has_more, and next_cursor where applicable; after is the opaque cursor input.

8.2.12 Response contract

Every reply has a declared schema in the OpenAPI document (/openapi.json), and the gateway validates what it sends against it. The schemas of replies the gateway forwards from the engine are generated from the same message specs as the engine's own bindings, so the REST shape and the bus payload cannot drift apart (see Messages).

- **Omitted versus null.** A field the schema marks optional is *omitted* when it does not apply (an ordinary order has no oco_group_id); a field it marks required is always present, and is null only where the schema allows it.
- **Collections** are { "<plural>": [...], "count": n }; a paginated one adds has_more and next_cursor. Envelopes do not nest: a bootstrap's orders is the list itself, not { "orders": [...] }.
- **Status fields** are closed enums declared in the schema.
- **Time.** *_ns fields are epoch nanoseconds; ts fields and every history from/to are ISO-8601 (or a YYYY-MM-DD date); date is a trading day in the exchange timezone (session_timezone in GET

`/session`).

- **Prices** are JSON numbers in display units (`187.25`), always on the symbol's tick grid. A field named `*_ticks` carries the same price as an integer tick count; it supplements a display field and never replaces one.
- **Extension.** New optional fields, new endpoints and new enum members may be added within v1. A client must ignore fields and enum values it does not recognise.

8.2.13 Canonical error envelope

All non-2xx replies MUST use this JSON envelope shape — including an unknown path (`404 NOT_FOUND`), a wrong method (`405 METHOD_NOT_ALLOWED`) and an unhandled gateway error (`500 INTERNAL_ERROR`).

Field	Type	Required	Description
`error`	`Object`	yes	Top-level error container
`error.code`	`String`	yes	Stable machine-readable code (for example `VALIDATION`)
`error.message`	`String`	yes	Human-readable summary
`error.field`	`String`	no	Field name when validation pinpoints one input field
`error.reject_code`	`String`	no	Canonical order-rejection class for ALF/REST comparison, when applicable
`error.client_tag`	`String`	no	Echo of the submitted order correlation tag, when available
`error.request_tag`	`String`	no	Echo of the amend/cancel request tag, when available

Canonical example:

```
{
  "error": {
    "code": "VALIDATION",
    "reject_code": "INVALID_VALUE",
    "message": "Input should be greater than 0",
    "field": "quantity",
    "client_tag": "desk1-aapl-00042"
  }
}
```

8.2.14 Pagination contract

Endpoints that support pagination use keyset cursoring with these rules:

- Clients MAY pass `after` (opaque cursor from a previous response).
- Replies include `count`, `has_more` and `next_cursor`.
- `next_cursor` is a string when another page exists and `null` on the last.
- Clients fetch the next page by passing `after=<next_cursor>` unchanged.
- Ordering is deterministic and backend-defined per endpoint; cursors are valid only for the same endpoint and compatible filter set.
- A malformed, stale, or cross-endpoint cursor returns `400 VALIDATION` with `field: "after"`.

8.2.15 Write outcomes and safe retries

Every write that creates or changes orders (`POST /orders`, `DELETE` and `PATCH /orders/{order_id}`, `POST /orders/{order_id}/replace`, `POST` and `DELETE` on `/oco`, `/combos` and `/quotes`) answers in one of three ways, so a client can always tell whether something reached the engine:

Reply	Sent to the engine?	What to do
`200 OK`, `status: "ACKED"`	Yes, and the verdict is in the body (`accepted`, `reject_code`, `event`)	Done
`202 Accepted`, `ack_timeout: false`	Yes; no `?wait=ack`, the verdict follows on the private WebSocket	Watch the WebSocket
`202 Accepted`, `ack_timeout: true`	Yes; `?wait=ack` gave up waiting (`wait_ack_sec`)	Watch the WebSocket or `GET /orders`; **do not resubmit**
`4xx` or `5xx`	No (`503 ENGINE_UNAVAILABLE`, `400`, `429`, …)	Fix and retry

The reply always carries the id to cancel or track (`order_id`, `oco_id`, `combo_id`, `symbol`), including after an ack timeout. Two exceptions, both on `replace`, which only sends the replacement once the original's cancel is confirmed: a refused cancel is `409 REPLACE_REJECTED` (nothing sent), and a cancel whose outcome did not arrive is `503 ENGINE_TIMEOUT` (no replacement sent; check the original with `GET /orders/{order_id}`).

`Idempotency-Key`. A client whose connection dropped cannot know which of those replies it missed. The submit writes (`POST /orders`, `/oco`, `/combos`, `/quotes` and `/orders/{order_id}/replace`) accept an optional `Idempotency-Key` header (1–255 characters; a UUID per logical request):

- The first request with a key runs. A later one with the same key, from the same gateway, is not sent again: it gets the first reply (same status, same body) with the header `Idempotent-Replayed: true`. A duplicate that arrives while the first is still waiting for its ack waits for it too.
- The same key on a different request (another path, query or body) is `409 IDEMPOTENCY_KEY_REUSED`.
- A request that failed before the engine was reached is not remembered, so it may be retried with the same key.
- `replace` is two writes — cancel, then submit the replacement — and the key records which of them succeeded. If the cancel is confirmed but the replacement fails, a retry with the same key resumes at the replacement: it does not cancel again, and it does not leave you with no order. A retry after both steps succeeded replays the first reply as usual.

- Keys are kept for `order_retention_sec` and are lost when the gateway restarts.

`client_tag` is unrelated: it is an opaque, non-unique label (below).

8.2.16 Order correlation with `client_tag`

Order submit supports `client_tag` as an optional client-supplied correlation tag.

- Scope: the tag is client-scoped and opaque.
- Behavior: the gateway and engine do not enforce uniqueness.
- Lifetime: when supplied, the tag is echoed on order lifecycle events and on cached order reads so clients can map exchange-assigned `order_id` values back to their own submissions. That includes events the client did not ask for — an unsolicited cancel, an expiry at the close — which are precisely the ones it cannot correlate any other way.

8.2.17 Unsolicited cancels and `cancel_reason`

An `order.cancelled` event does not always answer a `DELETE`. When the exchange cancels an order itself, `request_tag` is absent and `cancel_reason` says why:

`cancel_reason`	Meaning
`SELF_MATCH_PREVENTED`	The order would have traded against another order from the same gateway
`INSUFFICIENT_LIQUIDITY`	A `MARKET` or `IOC` order exhausted the book; this cancels the remainder, which never rests
`KILL_SWITCH`	The gateway's kill switch (`KILL` / `KILL_ACK`) cancelled this resting order
`CIRCUIT_BREAKER_HALT`	A circuit-breaker halt cancelled this resting order
`GATEWAY_DISCONNECT`	The owning gateway disconnected and its `disconnect_behaviour` cancelled this order
`ADMIN_CANCEL_SYMBOL`	An operator-issued admin cancel for the symbol cancelled this order
`QUOTE_REPLACED`	A market-maker's new quote replaced this quote leg
`QUOTE_LEG_FILLED`	The sibling leg of a two-sided quote filled, cancelling this leg

Both fields are omitted rather than null when they do not apply, so `request_tag` absent with `cancel_reason` also absent means the exchange cancelled the order for a cause not yet classified under this vocabulary. `cancel_reason` is a separate vocabulary from `reject_code`: a cancel is not a rejection. New members may be added; existing members are never removed or renamed. Clients must ignore values they do not recognise. See Messages — `order.cancelled.{gateway_id}` for the same enum at the engine-message level.

8.2.18 Category examples

The examples below are normative shape examples for each category. Values are illustrative.

Trading example (`POST /api/v1/orders`)

Minimal request:

```
{
  "symbol": "AAPL",
  "side": "BUY",
  "order_type": "LIMIT",
  "quantity": 100,
  "price": 187.25
}
```

Full request:

```
{
  "symbol": "AAPL",
  "side": "BUY",
  "order_type": "LIMIT",
  "quantity": 100,
  "price": 187.25,
  "tif": "DAY",
  "smp_action": "CANCEL_AGGRESSOR",
  "client_tag": "desk1-aapl-00042"
}
```

Minimal response:

```
{
  "order_id": "ORD-20260806-00042",
  "status": "PENDING"
}
```

Full response:

```
{
  "order_id": "ORD-20260806-00042",
  "client_tag": "desk1-aapl-00042",
  "status": "ACKED",
  "accepted": true,
  "event": {
    "order_id": "ORD-20260806-00042",
    "accepted": true,
    "gateway_id": "G1"
  }
}
```

Reference data example (`GET /api/v1/reference`)

Minimal response:

```
{
  "symbols": [],
  "risk": {},
  "indexes": [],
  "schedule": {},
  "config_version": "sha256:..."
}
```

Full response:

```
{
  "symbols": [
    {
      "symbol": "AAPL",
      "tick_decimals": 2,
      "level": "L1"
    }
  ],
  "risk": {
    "default_level": "L1",
    "levels": {
      "L1": {
        "soft_pct": 5.0,
        "hard_pct": 10.0
      }
    }
  },
  "indexes": [
    {
      "id": "TECH10",
      "description": "Tech sample index"
    }
  ],
  "schedule": {
    "sessions_enabled": true,
    "country": "US"
  },
  "config_version": "sha256:6d8d..."
}
```

History example (`GET /api/v1/history/trades`)

Minimal response:

```
{
  "trades": [],
  "count": 0,
  "has_more": false
}
```

Full response:

```
{
  "trades": [
    {
      "symbol": "AAPL",
      "price": 187.3,
      "quantity": 50,
      "ts": "2026-08-06T09:30:00Z"
    }
  ],
  "count": 1,
  "has_more": true,
  "next_cursor": "eyJhZnRlciI6Ii4uLiJ9"
}
```

Admin example (`POST /api/v1/admin/session/transition`)

Minimal request:

```
{
  "to_state": "CONTINUOUS"
}
```

Full request:

```
{
  "to_state": "CONTINUOUS"
}
```

Minimal response:

```
{
  "requested_state": "CONTINUOUS",
  "status": "APPLIED",
  "command_id": "c_01K17P..."
}
```

Full response:

```
{
  "requested_state": "CONTINUOUS",
  "status": "APPLIED",
  "command_id": "c_01K17P9B7C6W9X0Y8Z7"
}
```

8.2.19 Compatibility and deprecation policy

- Versioning: this surface is namespaced under `/api/v1`. The WebSocket streams under `/api/v1/ws/` (the WebSocket API Reference) are part of the same v1 contract and follow every rule below.
- Backward compatibility: additive changes (new optional fields/endpoints) MAY be introduced within v1 without changing the base path.
- Breaking changes (field removal, required-field additions, semantic redefinition) MUST ship under a new versioned base path.
- Deprecation: deprecated fields/endpoints SHOULD be documented with migration guidance before removal in a later major API version.
- The contract: `spec/rest/openapi.json` is the v1 contract. Any change to it is a reviewed diff, and each diff is judged against these rules.
- Additive in practice:
  - A new endpoint is additive.
  - A new optional request field is additive.
  - A new response field is additive. Clients must ignore unknown response fields.
  - A new error `code` is additive. Clients handle an unknown one by its HTTP status.
  - A new enum value in a response is additive only if the gateway and engine are deployed together. The gateway validates its responses, so a gateway older than the engine answers `500` on a value it does not know. EduMatcher releases both from one spec.
- What clients MUST tolerate, so that the additive changes above stay additive for them:
  - **Unknown fields**, in any REST response body and in any WebSocket frame or `data`: ignore them.
  - **Unknown enum values**, in any enum (`status`, `reject_code`, `cancel_reason`, `order_type`, …): treat them as "other", not as an error. New members may be added to any enum within v1.
  - **Unknown WebSocket `type` values**: ignore the frame. New event types may be added

within v1.
  - **Unknown error `code` values**: handle them by HTTP status.
- What is breaking for error codes: removing a `code` a route can return, changing a `code`'s HTTP status or meaning, or returning a different `code` for a condition that already has one. Adding a `code`, or returning an existing `code` from one more route, is additive.

8.3 Bootstrap

Bootstrap endpoints return a single composed response that a browser client can fetch immediately after authenticating, replacing the 6–13 sequential round-trips the current login flow requires. Subqueries inside each handler run in parallel using `asyncio.gather`.

Each response includes an `incomplete` array. When a field is listed there its value is `null`; the rest of the response is still usable. Required fields (marked below) return `503 ENGINE_TIMEOUT` on failure — the response would be too incomplete to be useful without them.

8.3.1 `GET /api/v1/bootstrap/trader`

Purpose: one-request startup payload for TRADER, MARKET_MAKER, and ADMIN sessions. Returns identity, full reference data, live session state, positions, active orders, today's recent fills, and capability flags.

Access: any valid key (read-only keys receive `gateway_role: "READ_ONLY"`, empty `positions` and `orders` and `null recent_fills`, but `session` and `halts` like any key).

Arguments

Name	Type	Req	Description
`fills_limit`	`Int`	no	Maximum number of today's fill events in `recent_fills`; `1..500`, default `50`

Reply

Status	Shape	Meaning
`200 OK`	the object below	Startup payload

```
{
  "ts": "2026-07-27T09:30:01.123Z",
  "incomplete": [],               // field names that timed out (values are null)

  "gateway_id": "TRADER01",       // null for read-only keys
  "gateway_role": "TRADER",       // TRADER | MARKET_MAKER | ADMIN | READ_ONLY
  "session_timezone": "Europe/Stockholm", // IANA name; the zone clients render wall-clock times in

  // identical to GET /reference
  "reference": {
    "symbols": [ { "symbol": "AAPL", "tick_decimals": 2, ... } ],
    "risk":     { "default_level": "L2", "levels": { ... } },
    "schedule": { "sessions_enabled": true, "country": "Sweden", "schedule": { ... } },
    "config_version": "7f3a2c1"
  },

  // identical to GET /session; null if engine timed out (optional)
  "session": { "state": "CONTINUOUS", "sessions_enabled": true, "next": { ... },
               "session_timezone": "Europe/Stockholm", "session_date": "2026-07-27" },

  // identical to GET /positions: an engine round trip (required — 503 on failure)
  "positions": [ { "symbol": "AAPL", "net_qty": 200, "avg_cost": 208.40, "last_price": 210.25,
                   "market_value": 42050.0, "unrealized_pnl": 370.0,
                   "realized_pnl_session": 0.0, "realized_pnl_mtd": 0.0, "realized_pnl_ytd": 0.0 } ],

  // the `orders` list of GET /orders (required — 503 on failure)
  "orders": [ /* OrderDisplay[] */ ],

  // today's fill events, at most fills_limit; null if stats DB absent (optional)
  "recent_fills": [ /* OrderEvent[] */ ],

  // the `halted` list of GET /halts; null if engine timed out (optional)
  "halts": [ ],

  // capability flags — assembled from config, never null
  "capabilities": {
    "sessions_enabled": true,    // from reference.schedule.sessions_enabled
    "stats_db_available": true,  // false → history unavailable
    "audit_db_available": false, // false → order lifecycle drill-down unavailable
    "index_available": false     // false → index tab unavailable
  }
}
```

Required fields (return `503 ENGINE_TIMEOUT` if they fail): `reference`, `orders` and `positions` (the last two are empty, without an engine query, for read-only keys).

Optional fields (appear as `null` + listed in `incomplete` on failure): `session`, `recent_fills`, `halts`.

Errors

Code	Status	When
`VALIDATION`	`400`	`fills_limit` is not an integer in 1–500
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint

Code	Status	When
`ENGINE_TIMEOUT`	`503`	`reference`, `orders` or `positions` could not be fetched

8.3.2 `GET /api/v1/bootstrap/mm`

Purpose: one-request startup payload for MARKET_MAKER sessions. Superset of `/bootstrap/trader` — adds active quote bootstrap state and quote legs.

Access: MARKET_MAKER key only. TRADER and ADMIN keys receive `403`.

Arguments

Name	Type	Req	Description
`fills_limit`	`Int`	no	As for `/bootstrap/trader`

Reply

Status	Shape	Meaning
`200 OK`	the object below	Startup payload

All fields from `/bootstrap/trader`, plus:

```
{
  // ...trader fields...
  "gateway_role": "MARKET_MAKER",

  // the `quotes` list of GET /quotes; null if engine timed out (optional)
  "quotes": [ /* ActiveQuote[] */ ],

  // the `legs` list of GET /quotes/legs; null if engine timed out (optional)
  "quote_legs": [ /* QuoteLeg[] */ ]
}
```

Required fields: `reference`, `orders`, `positions`.

Optional fields: `session`, `recent_fills`, `halts`, `quotes`, `quote_legs`.

Errors

Code	Status	When
`VALIDATION`	`400`	`fills_limit` is not an integer in 1–500
`READ_ONLY`	`403`	Read-only key (no gateway_id)
`ROLE_DENIED`	`403`	Key resolves to TRADER or ADMIN role
`ENGINE_TIMEOUT`	`503`	`reference`, `orders` or `positions` could not be fetched

8.3.3 `GET /api/v1/bootstrap/admin`

Purpose: one-request startup payload for ADMIN sessions. Returns reference data, session state, the full gateway roster, active halts, per-gateway active-order counts, per-gateway drop-copy sequence numbers, and capability flags.

Access: ADMIN role required. TRADER and MARKET_MAKER keys receive `403`.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	the object below	Startup payload

```
{
  "ts": "2026-07-27T09:30:01.250Z",
  "incomplete": [],

  "gateway_id": "INSTRUCTOR",
  "gateway_role": "ADMIN",

  // identical to GET /reference (required — 503 on failure)
  "reference": { /* symbols, risk, schedule, config_version */ },

  // identical to GET /session; null if engine timed out (optional)
  "session": { "state": "CONTINUOUS", "sessions_enabled": true, "session_timezone": "...", "session_date": ".

  // the `gateways` list of GET /admin/gateways; null if engine timed out (optional)
  "gateways": [
    { "id": "TRADER01", "role": "TRADER", "description": "...", "connected": true }
  ],

  // the `halted` list of GET /halts; null if engine timed out (optional)
  "halts": [ { "symbol": "AAPL", "level": "L2", ... } ],

  // per-gateway active (non-terminal) order count — pure cache, never null
  "active_order_counts": { "TRADER01": 3, "MM01": 12 },

  // each active gateway's private-stream stream_seq now — pure cache, never null
  "monitor_last_seq": { "TRADER01": 100482, "MM01": 88213 },

  "capabilities": {
    "sessions_enabled": true,
    "stats_db_available": true,
    "audit_db_available": false,
    "index_available": false
  }
}
```

Required fields: `reference`.

Optional fields: `session`, `gateways`, `halts`. The `gateways` and `halts` fields are also supplied by the `monitor.snapshot` frame that the `WS /api/v1/ws/admin/monitor` socket sends immediately after auth — a partial bootstrap response is therefore recoverable without a manual retry.

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Key is not ADMIN
`ENGINE_TIMEOUT`	`503`	`reference` could not be fetched

8.4 Trading REST

8.4.1 `POST /api/v1/orders`

Purpose: submit one order for the caller's gateway.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes	Instrument symbol
`side`	`Side`	yes	`BUY` or `SELL`
`order_type`	`OrderType`	yes	`MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, or `TRAILING_STOP`
`quantity`	`Int`	yes	Positive order quantity
`tif`	`Tif`	no	Time-in-force; defaults to `DAY`
`price`	`Price`	conditional	Required for `LIMIT`, `FOK`, `IOC`, `STOP_LIMIT`, `ICEBERG`
`stop_price`	`Price`	conditional	Required for `STOP`, `STOP_LIMIT`; optional initial stop for `TRAILING_STOP` (default: last trade ± `trail_offset`)
`visible_qty`	`Qty`	conditional	Required for `ICEBERG`; must be less than `quantity`
`trail_offset`	`Price`	conditional	Required for `TRAILING_STOP`

Name	Type	Req	Description
`smp_action`	`SmpAction`	no	Self-match prevention action
`client_tag`	`Str`	no	Optional opaque client correlation tag
`wait`	`Bool`	no	`?wait=ack` waits for the engine's verdict (see Write outcomes and safe retries)
`Idempotency-Key`	header	no	Makes a retry safe; see Write outcomes and safe retries

Reply

Status	Shape	Meaning
`202 Accepted`	`{"order_id": "...", "status": "PENDING", "ack_timeout": false, ...}`	Default immediate reply
`200 OK`	`{"order_id": "...", "status": "ACKED", "accepted": false, "reject_code": "COLLAR_BREACH", "event": {..., "reason": "collar breach"}, "ack_timeout": false}`	Returned when `?wait=ack` waits for the matching ACK; a rejected ACK carries the stable machine-readable `reject_code`, and the engine's `order.ack` in `event` has the human-readable `reason`
`202 Accepted`	`{"order_id": "...", "status": "PENDING", "ack_timeout": true, ...}`	`?wait=ack` and the ACK did not arrive in time; the order was sent

Takes an optional `Idempotency-Key` header; see Write outcomes and safe retries.

Errors

Code	Status	When
`VALIDATION`	`400`	Body does not match the order type, or a price is not an exact multiple of the instrument's tick size (`reject_code: TICK_VIOLATION`)
`READ_ONLY`	`403`	Read-only credential used
`IDEMPOTENCY_KEY_REUSED`	`409`	The `Idempotency-Key` was used for a different request
`RATE_LIMIT`	`429`	Write limit exceeded

Code	Status	When
`SYMBOLS_NOT_READY`	`503`	Symbol metadata has not reached the gateway yet (`reject_code: SYMBOL_NOT_READY`)

A conditional field the `order_type` does not use is rejected (`400 VALIDATION`, `reject_code: UNSUPPORTED_FIELD`, for example `"LIMIT forbids trail_offset"`), never silently dropped; a missing one is `MISSING_FIELD`. The same per-type rule applies to OCO and combo legs, and ALF, BALF and the engine apply it too, so an order gets the same `reject_code` on every transport.

Prices are refused rather than rounded to the nearest tick: rounding would rest the order at a price the client never sent. See Tick precision.

8.4.2 `DELETE /api/v1/orders/{order_id}`

Purpose: cancel one live order in the caller's gateway.

Arguments

Name	Type	Req	Description
`order_id`	`Str`	yes (path)	Order to cancel
`wait`	`Bool`	no	`?wait=ack` waits for `order.cancelled.*`
`request_tag`	`Str`	no	Query parameter echoed on `order.cancelled.*` or rejected ACK; max 64 chars

Reply

Status	Shape	Meaning
`202 Accepted`	`{"order_id": "...", "request_tag": "...", "status": "PENDING_CANCEL"}`	Default: cancel sent; the outcome follows on the private WebSocket
`200 OK`	`{"order_id": "...", "status": "ACKED" | "REJECTED", "accepted": ..., "reject_code": ..., "event": {...}}`	With `?wait=ack`: the cancel's outcome
`202 Accepted`	`{..., "status": "PENDING_CANCEL", "ack_timeout": true}`	`?wait=ack` and the outcome did not arrive in time; the cancel was sent

Without a `request_tag`, `?wait=ack` waits for `order.cancelled` alone, so a refused cancel shows as an ack timeout; pass a `request_tag` to get it as `REJECTED`.

Errors

Code	Status	When
`VALIDATION`	`400`	Bad path or query
`READ_ONLY`	`403`	Read-only credential used
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause

8.4.3 `PATCH /api/v1/orders/{order_id}`

Purpose: amend price and/or quantity on one live order.

Arguments

Name	Type	Req	Description
`order_id`	`Str`	yes (path)	Order to amend
`price`	`Price`	conditional	New order price
`quantity`	`Qty`	conditional	New order quantity
`request_tag`	`Str`	no	Body field echoed on `order.amended.*` or rejected ACK; max 64 chars
`wait`	`Bool`	no	`?wait=ack` waits for `order.amended.*`

Reply

Status	Shape	Meaning
`202 Accepted`	`{"order_id": "...", "request_tag": "...", "status": "PENDING_AMEND"}`	Default: amend sent; the outcome follows on the private WebSocket
`200 OK`	`{"order_id": "...", "status": "ACKED" | "REJECTED", "accepted": ..., "reject_code": ..., "event": {...}}`	With `?wait=ack`: the amend's outcome
`202 Accepted`	`{..., "status": "PENDING_AMEND", "ack_timeout": true}`	`?wait=ack` and the outcome did not arrive in time; the amend was sent

Errors

Code	Status	When
`VALIDATION`	`400`	Neither or both fields invalid

Code	Status	When
`READ_ONLY`	`403`	Read-only credential used
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause

8.4.4 `POST /api/v1/orders/{order_id}/replace`

Purpose: cancel one live order and submit a replacement in one workflow.

Arguments

Name	Type	Req	Description
`order_id`	`Str`	yes (path)	Order to replace
body	`Order` fields	yes	Same shape as `POST /orders`, on the same symbol
`wait`	`Bool`	no	`?wait=ack` also waits for the replacement's `order.ack`
`Idempotency-Key`	header	no	See Write outcomes and safe retries

The gateway always waits for the original's cancel, and sends the replacement only once the engine has confirmed it.

Reply

Status	Shape	Meaning
`202 Accepted`	`{"cancelled_order_id": "...", "replacement_order_id": "...", "status": "PENDING", "accepted": null, ...}`	Original cancelled; replacement sent
`200 OK`	`{..., "status": "ACKED", "accepted": true | false, "reject_code": ..., "event": {...}}`	With `?wait=ack`: the replacement's verdict

Errors

Code	Status	When
`VALIDATION`	`400`	Replacement body invalid, or names another symbol than the original (`field: symbol`)
`READ_ONLY`	`403`	Read-only credential used

Code	Status	When
`IDEMPOTENCY_KEY_REUSED`	`409`	The `Idempotency-Key` was used for a different request
`REPLACE_REJECTED`	`409`	The engine refused the cancel (the order already filled, say; `reject_code` says why); nothing was sent
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	The cancel's outcome did not arrive in time; no replacement was sent
`SYMBOLS_NOT_READY`	`503`	The symbol's metadata is not loaded; nothing was cancelled

8.4.5 `GET /api/v1/orders`

Purpose: return the caller gateway's orders as the engine holds them: resting orders plus recently finished ones.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "orders": [...] }`	The engine's order snapshot for the caller

Errors

Code	Status	When
`READ_ONLY`	`403`	Read-only credential used
`ENGINE_TIMEOUT`	`503`	The engine did not answer in time. There is no fallback to the gateway's cache; the private WebSocket `orders.snapshot` is the cached view

8.4.6 `GET /api/v1/orders/{order_id}`

Purpose: return one cached order from the caller gateway.

Arguments

Name	Type	Req	Description
`order_id`	`Str`	yes (path)	Order to fetch

Reply

Status	Shape	Meaning
`200 OK`	cached order object	Current order state
`404 Not Found`	error envelope	Order id not in cache

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	Read-only credential used
`UNKNOWN_ORDER`	`404`	Order id unknown to the cache

8.4.7 `POST /api/v1/oco`

Purpose: submit an OCO pair.

Arguments

Name	Type	Req	Description
`oco_id`	`Str`	yes	Your id for the pair, unique among your live pairs: reusing one is refused (`accepted: false`). Free again once the pair is done
`symbol`	`Symbol`	yes	Instrument symbol
`quantity`	`Qty`	yes	Total quantity

Name	Type	Req	Description
`leg1`	`OrderLeg`	yes	First leg definition: `side`, `order_type`, and the `price` / `stop_price` / `trail_offset` that type needs (same rule as `POST /orders`; `ICEBERG` is not supported on a leg)
`leg2`	`OrderLeg`	yes	Second leg definition, same shape
`tif`	`Tif`	no	Time-in-force of both legs; defaults to `DAY`
`client_tag`	`Str`	no	Opaque label echoed on both legs' events; max 64 chars
`wait`	`Bool`	no	`?wait=ack` waits for the engine's verdict (see Write outcomes and safe retries)
`Idempotency-Key`	header	no	Makes a retry safe; see Write outcomes and safe retries

Reply

Status	Shape	Meaning
`202 Accepted`	`{ "oco_id": "...", "status": "PENDING", "accepted": null, "reason": null, "event": null, "ack_timeout": false }`	Sent; the verdict follows on the private WebSocket (`oco.ack`)
`200 OK`	`{ "oco_id": "...", "status": "ACKED", "accepted": true | false, "reason": ..., "event": {...}, "ack_timeout": false }`	With `?wait=ack`: the engine's `oco.ack`
`202 Accepted`	`{..., "status": "PENDING", "ack_timeout": true}`	`?wait=ack` and the verdict did not arrive in time; it was sent

Takes `?wait=ack` and an optional `Idempotency-Key` header; see Write outcomes and safe retries.

Errors

Code	Status	When
`VALIDATION`	`400`	Body invalid

Code	Status	When
`READ_ONLY`	`403`	Read-only credential used
`IDEMPOTENCY_KEY_REUSED`	`409`	The `Idempotency-Key` was used for a different request
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`SYMBOLS_NOT_READY`	`503`	Order entry before the symbol's tick precision reached the gateway. Transient; retry

8.4.8 `DELETE /api/v1/oco/{oco_id}`

Purpose: cancel an OCO pair.

Arguments

Name	Type	Req	Description
`oco_id`	`Str`	yes (path)	One of your own OCO pairs; another gateway's id is `ORDER_NOT_FOUND`
`request_tag`	`Str`	no	Query: echoed on the engine's `oco.cancel_ack`; max 64 chars
`wait`	`Bool`	no	`?wait=ack` waits for the `oco.cancel_ack`

Reply

Status	Shape	Meaning
`202 Accepted`	`{ "oco_id": "...", "request_tag": ..., "status": "PENDING_CANCEL", "accepted": null, ..., "ack_timeout": false }`	Default: cancel sent; the outcome follows on the private WebSocket (`oco.cancel_ack`)
`200 OK`	`{ ..., "status": "ACKED" | "REJECTED", "accepted": ..., "reject_code": ..., "event": {...} }`	With `?wait=ack`: the engine's `oco.cancel_ack`. A refusal (`ORDER_NOT_FOUND`, `ORDER_ALREADY_TERMINAL`) is `REJECTED`, not an error
`202 Accepted`	`{ ..., "status": "PENDING_CANCEL", "ack_timeout": true }`	`?wait=ack` and the ack did not arrive in time; the cancel was sent

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	Read-only credential used
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause

8.4.9 `POST /api/v1/combos`

Purpose: submit a combo order.

Arguments

Name	Type	Req	Description
`combo_id`	`Str`	yes	Combo group id
`legs`	`List<ComboLeg>`	yes	2–10 legs: `symbol`, `side`, `order_type` (default `LIMIT`), `quantity`, and the `price` / `stop_price` that type needs (same rule as `POST /orders`; `ICEBERG` and `TRAILING_STOP` are not supported on a leg)
`combo_type`	`ComboType`	no	Fill rule; `AON` (all or none), the default and only value
`tif`	`Tif`	no	Time-in-force of every leg; defaults to `DAY`
`smp_action`	`SmpAction`	no	Self-match prevention action, as on `POST /orders`
`client_tag`	`Str`	no	Opaque label echoed on every leg's events; max 64 chars
`wait`	`Bool`	no	`?wait=ack` waits for the engine's verdict (see Write outcomes and safe retries)

Name	Type	Req	Description
`Idempotency-Key`	header	no	Makes a retry safe; see Write outcomes and safe retries

Reply

Status	Shape	Meaning
`202 Accepted`	`{ "combo_id": "...", "status": "PENDING", "accepted": null, "reason": null, "event": null, "ack_timeout": false }`	Sent; the verdict follows on the private WebSocket (`combo.ack`)
`200 OK`	`{ "combo_id": "...", "status": "ACKED", "accepted": true | false, "reason": ..., "event": {...}, "ack_timeout": false }`	With `?wait=ack`: the engine's `combo.ack`
`202 Accepted`	`{..., "status": "PENDING", "ack_timeout": true}`	`?wait=ack` and the verdict did not arrive in time; it was sent

Takes `?wait=ack` and an optional `Idempotency-Key` header; see Write outcomes and safe retries.

Errors

Code	Status	When
`VALIDATION`	`400`	Body invalid
`READ_ONLY`	`403`	Read-only credential used
`IDEMPOTENCY_KEY_REUSED`	`409`	The `Idempotency-Key` was used for a different request
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`SYMBOLS_NOT_READY`	`503`	Order entry before the symbol's tick precision reached the gateway. Transient; retry

8.4.10 `DELETE /api/v1/combos/{combo_id}`

Purpose: cancel a combo order.

Arguments

Name	Type	Req	Description
`combo_id`	`Str`	yes (path)	Combo group to cancel

Name	Type	Req	Description
`request_tag`	`Str`	no	Query: echoed on the engine's `combo.cancel_ack`; max 64 chars
`wait`	`Bool`	no	`?wait=ack` waits for the `combo.cancel_ack`

Reply

Status	Shape	Meaning
`202 Accepted`	`{ "combo_id": "...", "request_tag": ..., "status": "PENDING_CANCEL", "accepted": null, ..., "ack_timeout": false }`	Default: cancel sent; the outcome follows on the private WebSocket (`combo.cancel_ack`)
`200 OK`	`{ ..., "status": "ACKED" | "REJECTED", "accepted": ..., "reject_code": ..., "event": {...} }`	With `?wait=ack`: the engine's `combo.cancel_ack`. A refusal (`ORDER_NOT_FOUND`, `ORDER_ALREADY_TERMINAL`) is `REJECTED`, not an error
`202 Accepted`	`{ ..., "status": "PENDING_CANCEL", "ack_timeout": true }`	`?wait=ack` and the ack did not arrive in time; the cancel was sent

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	Read-only credential used
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause

8.4.11 `POST /api/v1/quotes`

Purpose: submit a two-sided market-maker quote.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes	Quoted instrument
`bid_price`	`Price`	yes	Bid price
`bid_qty`	`Qty`	yes	Bid quantity
`ask_price`	`Price`	yes	Ask price
`ask_qty`	`Qty`	yes	Ask quantity
`tif`	`Tif`	no	Time-in-force of both sides; defaults to `DAY`
`quote_id`	`Str`	no	Your id for the quote; defaults to the symbol
`wait`	`Bool`	no	`?wait=ack` waits for the engine's verdict (see Write outcomes and safe retries)
`Idempotency-Key`	header	no	Makes a retry safe; see Write outcomes and safe retries

Reply

Status	Shape	Meaning
`202 Accepted`	`{ "symbol": "...", "quote_id": "..." | null, "status": "PENDING", "accepted": null, "reason": null, "event": null, "ack_timeout": false }`	Sent; the verdict follows on the private WebSocket (`quote.ack`)
`200 OK`	`{ "symbol": "...", ..., "status": "ACKED", "accepted": true | false, "reason": ..., "event": {...} }`	With `?wait=ack`: the engine's `quote.ack`
`202 Accepted`	`{..., "status": "PENDING", "ack_timeout": true}`	`?wait=ack` and the verdict did not arrive in time; it was sent

Takes `?wait=ack` and an optional `Idempotency-Key` header; see Write outcomes and safe retries.

Errors

Code	Status	When
`VALIDATION`	`400`	Quote body invalid
`READ_ONLY`	`403`	Read-only credential used

Code	Status	When
`IDEMPOTENCY_KEY_REUSED`	`409`	The `Idempotency-Key` was used for a different request
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`SYMBOLS_NOT_READY`	`503`	Order entry before the symbol's tick precision reached the gateway. Transient; retry

8.4.12 `DELETE /api/v1/quotes/{symbol}`

Purpose: cancel the active quote for one symbol.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes (path)	Quoted instrument
`request_tag`	`Str`	no	Query: echoed on the engine's `quote.cancel_ack`; max 64 chars
`wait`	`Bool`	no	`?wait=ack` waits for the `quote.cancel_ack`

Reply

Status	Shape	Meaning
`202 Accepted`	`{ "symbol": "...", "request_tag": ..., "status": "PENDING_CANCEL", "accepted": null, ..., "ack_timeout": false }`	Default: cancel sent; the outcome follows on the private WebSocket (`quote.cancel_ack`)
`200 OK`	`{ ..., "status": "ACKED" | "REJECTED", "accepted": ..., "reject_code": ..., "event": {...} }`	With `?wait=ack`: the engine's `quote.cancel_ack`. A refusal (`ORDER_NOT_FOUND`, `ORDER_ALREADY_TERMINAL`) is `REJECTED`, not an error
`202 Accepted`	`{ ..., "status": "PENDING_CANCEL", "ack_timeout": true }`	`?wait=ack` and the ack did not arrive in time; the cancel was sent

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	Read-only credential used
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause

8.4.13 `POST /api/v1/mass-cancel`

Purpose: cancel all resting exposure for the caller or one symbol.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	no	Restrict the cancel to one symbol

Reply

Status	Shape	Meaning
`200 OK`	kill-switch ack	What the engine cancelled (the call waits for the ack)

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	Read-only credential used
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not ACK in time

8.4.14 `GET /api/v1/symbols`

Purpose: return instrument metadata for the caller gateway.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbols": [...] }`	Current symbol metadata

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.4.15 `GET /api/v1/news`

Purpose: return the latest market-sim headlines, oldest first, plus the sector → symbol mapping. Access: Authenticated key (read-only keys included) — unlike most endpoints in this section, it does not require a trading key, since this is public information, not a trading action.

Arguments

Name	Type	Req	Description
`limit`	int	No	Maximum headlines to return, `1-200` (default `50`)

Reply

Status	Shape	Meaning
`200 OK`	`{ "news": [...], "sectors": {...} }`	`news` is a list of `news.event`-shaped objects (see Message Reference — news.event), oldest first; `sectors` maps each sector name to its sorted list of symbols. Empty `news` list when `pm-market-sim` is not running.

Errors

Code	Status	When
`VALIDATION`	`400`	`limit` outside `1-200`

8.4.16 `GET /api/v1/companies/{symbol}`

Purpose: return the company profile behind one symbol, from the company database `pm-compdb-cli init` builds (`companies.db` in the data directory). The gateway opens it read-only. Access: Authenticated key (read-only keys included) — the profiles are public information.

Arguments

Name	Type	Req	Description
`symbol`	path	Yes	Ticker, case-insensitive

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbol": "AAPL", "name": ..., "pe_ratio": ..., ... }`	Every column of `company_view`: the stored fields plus the derived `years_in_business`, `ceo_tenure_years`, `pe_ratio`, `ps_ratio` and `net_margin_pct`. Money is in millions of USD; a field the database has no value for is `null`. `pm-compdb-cli columns` lists them all.

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`UNKNOWN_COMPANY`	`404`	The database has no profile for `symbol`
`COMPANY_DB_UNAVAILABLE`	`503`	No `companies.db` in the data directory, or it cannot be read

8.4.17 `GET /api/v1/session`

Purpose: return the venue's session phase and the locale clients render it in. Open to every key, read-only included.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "state", "sessions_enabled", "next", "session_timezone", "session_date" }`	`state` and `next` come from the engine; `session_timezone` (IANA) is the exchange timezone every wall-clock time and history `date` filter is in, and `session_date` today's trading date in it

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.4.18 `GET /api/v1/halts`

Purpose: return the currently-halted symbols table, for any key, read-only included. Halt state is public: every client needs it at bootstrap and again on every market-data reconnect, or a halt missed while disconnected leaves a stale badge for the rest of the session.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "halted": [...] }`	Currently-halted symbols; each `{ symbol, resume_at_ns?, level?, halt_source? }`

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.4.19 `GET /api/v1/status`

Purpose: return who the caller is: its gateway id and engine role.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "gateway_id", "gateway_role", "gateway_count"? }`	`gateway_role` is TRADER, MARKET_MAKER or ADMIN; ADMIN keys also get `gateway_count`, the connected gateways

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint

8.4.20 `GET /api/v1/healthz`

Purpose: liveness probe for the API gateway.

Arguments

Name	Type	Req	Description
none	—	—	No auth required

Reply

Status	Shape	Meaning
`200 OK`	`{ "ok": true, ... }`	Gateway is enabled and its engine listener is alive
`503 Service Unavailable`	`{ "ok": false, ... }`	Gateway is disabled or its engine listener has stopped; probes read the status code

Errors

Code	When
none	This endpoint does not require auth

8.4.21 `GET /api/v1/quotes`

Purpose: return the caller's active market-maker quotes: both legs' order ids, sizes, remaining quantities, statuses and prices.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "quotes": [...] }`	The caller's active quotes

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.4.22 `GET /api/v1/quotes/legs`

Purpose: return the current quote-leg state for the caller.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "legs": [...] }`	Current quote legs

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.4.23 `GET /api/v1/positions`

Purpose: return current net positions by symbol.

To query a *different* gateway's position (e.g. checking on a running `pm-mm-bot`), see GET /api/v1/admin/gatewa which requires the ADMIN role.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "positions": [{ "symbol", "net_qty", "avg_cost", "last_price", "market_value", "unrealized_pnl", "realized_pnl_session", "realized_pnl_mtd", "realized_pnl_ytd" }, ...] }`	Net positions for the caller gateway, from the engine; P&L fields are `null` when unknown

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.5 Reference data

Base path: `/api/v1/reference`. These endpoints expose compiled reference data that changes only when an admin reloads it.

8.5.1 `GET /api/v1/reference`

Purpose: return the full reference bundle in one round-trip.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbols", "risk", "indexes", "schedule", "config_version" }`	Full resolved reference bundle

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.5.2 `GET /api/v1/reference/config-version`

Purpose: return the content-hash version of the reference bundle.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "config_version": "..." }`	Opaque bundle version hash

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.5.3 `GET /api/v1/reference/symbols`

Purpose: return per-symbol tick and risk metadata.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbols": [...], "config_version": "..." }`	One object per symbol (each carries its own `symbol`)

Each symbol object carries `tick_decimals`, `level`, and — when configured — `collar`, `order_limits` (`max_order_qty` / `max_order_value`) and `circuit_breaker`. `collar` is the symbol's **effective** band, its risk level's defaults already merged under any per-symbol override; `order_limits` is configured per symbol only, so it is simply what that symbol declared. Either is omitted entirely when not configured. Full field law: `ReferenceSymbol` in the message reference.

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.5.4 `GET /api/v1/reference/risk`

Purpose: return risk-band definitions and the default risk level.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "default_level", "levels", "config_version" }`	Risk-band configuration

Each entry of `levels` carries its `name` and, when the level configures one, `collar` — as configured, before any per-symbol override. Levels do not carry order limits; those are per symbol and appear on `/reference/symbols`. Full field law: `RiskLevel` in the message reference.

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.5.5 `GET /api/v1/reference/indexes`

Purpose: return configured exchange index definitions.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "indexes": [...], "config_version": "..." }`	Static index definitions

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.5.6 `GET /api/v1/reference/schedule`

Purpose: return session schedule metadata.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	schedule object	`sessions_enabled`, `country`, and session transition times

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.6 Market data

Base path: `/api/v1/book`. The order book as of now, without a WebSocket. Both endpoints read the gateway's copy of the `book.{symbol}` stream, the one a WebSocket `snapshot` is answered from, so a REST read and a WebSocket subscription agree on the same snapshot.

- **`seq`** is the market-data sequence of the snapshot on topic `book.{symbol}`, the `seq` its WebSocket `book` frame carries. A client that reads here and then subscribes ignores `book` frames whose `seq` is at or below it (the subscription replays the latest snapshot first).
- **`ts_ns`** is when the engine produced the snapshot. The engine publishes a book when it changes, so a quiet book keeps an old `ts_ns` and is still current.
- Every price is a display price with a `*_ticks` sibling: the integer `price * 10**tick_decimals` (see *Prices* in Response contract). Use the ticks to compare or sum prices exactly.
- Iceberg orders contribute their displayed size only.

8.6.1 `GET /api/v1/book/{symbol}`

Purpose: return one symbol's latest book: bid and ask levels (best first), the last trade, and the last five prints. Access: Authenticated key (read-only keys included) — market data is public.

Arguments

Name	Type	Req	Description
`symbol`	path	Yes	Ticker, case-insensitive

Name	Type	Req	Description
`levels`	query	No	Price levels per side, `1`–`100`, default `10`

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbol", "tick_decimals", "ts_ns", "seq", "bids": [...], "asks": [...], "last_price", "last_price_ticks", "last_qty", "last_buy_price", "last_buy_price_ticks", "last_sell_price", "last_sell_price_ticks", "recent_trades": [...] }`	`bids` descending and `asks` ascending, each level `{ price, price_ticks, qty, count }`; `last_*` are `null` until the symbol has traded; `recent_trades` are the last five prints, oldest first, each as in `trade.executed` plus `price_ticks`

Errors

Code	Status	When
`VALIDATION`	`400`	`levels` outside `1`–`100`, or `symbol` over-long
`UNKNOWN_SYMBOL`	`404`	`symbol` is not in the reference data
`ENGINE_TIMEOUT`	`503`	The engine did not publish the book in time (only on the first read of a book the gateway has not seen)

8.6.2 `GET /api/v1/book`

Purpose: return best bid, best ask and last price for every symbol in the reference data, sorted by symbol, in one call. Access: Authenticated key (read-only keys included).

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "books": [...], "count": n }`	One entry per symbol: `{ symbol, tick_decimals, ts_ns, seq, best_bid, best_ask, last_price, last_price_ticks }`. A side with no orders is `null`; `best_bid` and `best_ask` are levels as above

Errors

Code	Status	When
`ENGINE_TIMEOUT`	`503`	The engine did not reply in time. The list is all or nothing: one book that cannot be had fails the call

8.7 History

Base path: `/api/v1/history`. Trading endpoints are scoped to the caller's gateway id; public history accepts any valid key.

8.7.1 `GET /api/v1/history/orders`

Purpose: return the caller gateway's order lifecycle events.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	no	Filter by symbol
`event_type`	`Str`	no	Filter by event type
`date`	`Date`	no	Single trading date
`from`	`DateTime`	no	Range start
`to`	`DateTime`	no	Range end
`limit`	`Int`	no	Page size
`after`	`Cursor`	no	Opaque page cursor

Reply

Status	Shape	Meaning
`200 OK`	paginated list envelope	Order lifecycle events for the caller gateway

Errors

Code	Status	When
`VALIDATION`	`400`	Query invalid
`READ_ONLY`	`403`	Read-only credential used
`STATS_DB`	`503`	`pm-stats` database missing

8.7.2 `GET /api/v1/history/orders/{order_id}`

Purpose: return the full lifecycle of one order for the caller gateway.

Arguments

Name	Type	Req	Description
`order_id`	`Str`	yes (path)	Order to fetch

Reply

Status	Shape	Meaning
`200 OK`	`{ "order_id", "count", "events" }`	Full lifecycle for one order

Errors

Code	Status	When
`VALIDATION`	`400`	Path invalid
`READ_ONLY`	`403`	Read-only credential used
`STATS_DB`	`503`	`pm-stats` database missing

8.7.3 `GET /api/v1/history/fills`

Purpose: return fill events for the caller gateway.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	no	Filter by symbol
`date`	`Date`	no	Trading date
`from`	`DateTime`	no	Range start
`to`	`DateTime`	no	Range end
`limit`	`Int`	no	Page size
`after`	`Cursor`	no	Opaque page cursor

Reply

Status	Shape	Meaning
`200 OK`	paginated list envelope	Fill history for the caller gateway

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	Read-only credential used
`STATS_DB`	`503`	`pm-stats` database missing

8.7.4 `GET /api/v1/history/trades`

Purpose: return public trade tape rows.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	no	Filter by symbol
`date`	`Date`	no	Trading date
`from`	`DateTime`	no	Range start
`to`	`DateTime`	no	Range end
`limit`	`Int`	no	Page size
`after`	`Cursor`	no	Opaque page cursor
`order`	`asc` | `desc`	no	`asc` (default) is oldest first; `desc` is newest first, so `limit` returns the most recent trades. The cursor continues in the chosen direction

Reply

Status	Shape	Meaning
`200 OK`	paginated list envelope	Public trade rows

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`STATS_DB`	`503`	`pm-stats` database missing

8.7.5 `GET /api/v1/history/daily`

Purpose: return daily OHLCV rows.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	no	Filter by symbol
`date`	`Date`	no	Single trading day
`from`	`Date`	no	Range start
`to`	`Date`	no	Range end
`limit`	`Int`	no	Page size
`after`	`Cursor`	no	Opaque page cursor

Reply

Status	Shape	Meaning
`200 OK`	paginated list envelope	Daily OHLCV rows

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`STATS_DB`	`503`	`pm-stats` database missing

8.7.6 `GET /api/v1/history/price-snapshots`

Purpose: return intraday price snapshots.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes	Instrument symbol
`date`	`Date`	no	Trading day
`from`	`DateTime`	no	Range start
`to`	`DateTime`	no	Range end
`limit`	`Int`	no	Page size
`after`	`Cursor`	no	Opaque page cursor

Reply

Status	Shape	Meaning
`200 OK`	paginated list envelope	Intraday mid/bid/ask series

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`STATS_DB`	`503`	`pm-stats` database missing

8.7.7 `GET /api/v1/history/index-daily`

Purpose: return daily index OHLC rows.

Arguments

Name	Type	Req	Description
`index_id`	`IndexId`	no	Index id
`date`	`Date`	no	Single trading day
`from`	`Date`	no	Range start
`to`	`Date`	no	Range end
`limit`	`Int`	no	Page size
`after`	`Cursor`	no	Opaque page cursor

Reply

Status	Shape	Meaning
`200 OK`	paginated list envelope	Daily index rows

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`STATS_DB`	`503`	`pm-stats` database missing

8.7.8 `GET /api/v1/history/index-snapshots`

Purpose: return intraday index snapshots.

Arguments

Name	Type	Req	Description
`index_id`	`IndexId`	yes	Index id
`date`	`Date`	no	Trading day
`from`	`DateTime`	no	Range start
`to`	`DateTime`	no	Range end
`limit`	`Int`	no	Page size
`after`	`Cursor`	no	Opaque page cursor

Reply

Status	Shape	Meaning
`200 OK`	paginated list envelope	Intraday index series

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`

Code	Status	When
`STATS_DB`	`503`	`pm-stats` database missing

8.7.9 `GET /api/v1/history/indexes`

Purpose: list index ids with recorded data.

Arguments

Name	Type	Req	Description
`date`	`Date`	no	Filter by trading day

Reply

Status	Shape	Meaning
`200 OK`	`{ "index_ids": [...], "count": N }`	Index ids with data

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`STATS_DB`	`503`	`pm-stats` database missing

8.7.10 `GET /api/v1/history/index-events`

Purpose: return index structural and audit events from `pm-index`.

Arguments

Name	Type	Req	Description
`index_id`	`IndexId`	yes	Index id
`from`	`Timestamp`	no	ISO-8601 or `YYYY-MM-DD`, like every history range (no offset = exchange time); default 30 days ago
`to`	`Timestamp`	no	Same form; default now

Name	Type	Req	Description
`types`	`Str`	no	Repeatable event-type filter
`max_records`	`Int`	no	Reply cap

Reply

Status	Shape	Meaning
`200 OK`	`{ "events": [...], "count": N }`	Structural audit log

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`INDEX_ERROR`	`502`	`pm-index` rejected the request
`INDEX_TIMEOUT`	`503`	`pm-index` did not reply in time

8.8 Admin REST

Base path: `/api/v1/admin`. ADMIN role required.

8.8.1 `POST /api/v1/admin/session/transition`

Purpose: request a session-phase transition from the engine.

Arguments

Name	Type	Req	Description
`to_state`	`SessionState`	yes	Requested session phase

Reply

Status	Shape	Meaning
`200 OK`	`{"requested_state": "...", "status": "APPLIED", "command_id": "..."}`	Transition applied
`409 Conflict`	error envelope	The engine refused the transition

Errors

Code	Status	When
`VALIDATION`	`400`	Invalid `to_state`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`TRANSITION_REJECTED`	`409`	Engine rejected the transition
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.2 `GET /api/v1/admin/sim`

Purpose: the market simulator's latest state and the model's true (hidden) fair value per symbol, from `pm-market-sim`. For instructors: students see only prices, never these values.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "available": true, "state": {...}, "values": {...} }`	`available` is false, and `state`/`values` null, when the simulator is not running

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	The key's gateway is not ADMIN

8.8.3 `GET /api/v1/admin/gateways`

Purpose: list configured gateways and live connection state.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "gateways": [...] }`	Gateway roster and connection status

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.4 `POST /api/v1/admin/gateways/{gateway_id}/disconnect`

Purpose: forcibly disconnect one gateway.

Arguments

Name	Type	Req	Description
`gateway_id`	`GatewayId`	yes (path)	Gateway to disconnect (case-insensitive)
`reason`	`Str`	no	Body, optional: note recorded in the engine log (default `admin disconnect`)

Reply

Status	Shape	Meaning
`202 Accepted`	`{ "gateway_id": "...", "status": "PENDING_DISCONNECT" }`	Disconnect sent; not awaited — the roster in `GET /admin/gateways` shows the result

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause

8.8.5 `POST /api/v1/admin/symbols/{symbol}/halt`

Purpose: halt one symbol through the engine's circuit-breaker path. The body is optional.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes (path)	Symbol to halt (case-insensitive)
`level`	`Str`	no	A rung of the symbol's ladder; omit for an indefinite halt until resumed
`reason`	`Str`	no	Optional note

Reply

Status	Shape	Meaning
`200 OK`	halt ack	The symbol is halted
`409 Conflict`	error envelope	The engine refused the halt

Errors

Code	Status	When
`VALIDATION`	`400`	Body invalid
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN

Code	Status	When
`HALT_REJECTED`	`409`	Engine refused (unknown symbol, no circuit breaker, unknown level); `message` is its reason
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.6 `POST /api/v1/admin/symbols/{symbol}/resume`

Purpose: resume one halted symbol.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes (path)	Symbol to resume (case-insensitive)
`reason`	`Str`	no	Body, optional: note for the audit log

Reply

Status	Shape	Meaning
`200 OK`	resume ack	The symbol is trading again
`409 Conflict`	error envelope	The engine refused the resume

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`RESUME_REJECTED`	`409`	Engine refused (for example the symbol is not halted); `message` is its reason

Code	Status	When
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.7 `GET /api/v1/admin/risk/state`

Purpose: return live per-symbol risk state.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbols": [...] }`	Current collar and circuit-breaker state (one object per symbol)

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.8 `GET /api/v1/admin/gateways/{gateway_id}/orders`

Purpose: one gateway's resting and recently finished orders as the engine holds them: the authoritative view. `GET /admin/orders` is the gateway's own read model across all gateways, faster and filterable but limited to what this gateway has seen since it started.

Arguments

Name	Type	Req	Description
`gateway_id`	`GatewayId`	yes (path)	Whose orders (case-insensitive)

Reply

Status	Shape	Meaning
`200 OK`	`{ "orders": [...], "count": N }`	Same shape as `GET /orders`

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.9 `GET /api/v1/admin/symbols/{symbol}/orders`

Purpose: every resting order on one symbol across all gateways, each with its `gateway_id`, ordered by price then arrival (time priority). With `price`, only that level: the per-order composition behind the book's aggregated levels.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes (path)	Instrument (case-insensitive)
`price`	`Price`	no	Query: only this price level

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbol", "price"?, "rejected": false, "orders": [...], "count": N }`	The resting orders

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`QUERY_REJECTED`	`409`	Engine refused (for example `Unknown symbol: XYZ`)
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.10 `GET /api/v1/admin/orders`

Purpose: return the cross-gateway active-order table.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	no	Filter by symbol
`gateway_id`	`GatewayId`	no	Filter by gateway
`status`	`OrderStatus`	no	Filter by status

Reply

Status	Shape	Meaning
`200 OK`	`{ "count": N, "orders": [...], "retention_sec": N }`	Current cross-gateway active orders

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN

Code	Status	When

8.8.11 `GET /api/v1/admin/orders/{order_id}/audit`

Purpose: return the full cross-gateway lifecycle of one order.

Arguments

Name	Type	Req	Description
`order_id`	`Str`	yes (path)	Order to fetch
`limit`	`Int`	no	Maximum number of events

Reply

Status	Shape	Meaning
`200 OK`	`{ "order_id", "count", "events" }`	Full audited lifecycle

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`UNKNOWN_ORDER`	`404`	No audited events for the order
`AUDIT_INDEX_UNAVAILABLE`	`503`	No audit index available

8.8.12 `GET /api/v1/admin/gateways/{gateway_id}/positions`

Purpose: return net position and average cost for *any one* gateway (not just the caller's own) – the admin counterpart to GET /api/v1/positions, which serves the same engine round trip for the caller enriched with prices and P&L. It is the REST equivalent of `pm-alf-console`'s and `pm-alf-gwy`'s `POS|GW=<gateway_id>` command – all three read the same `system.position_request` / `system.position_snapshot.<gatew` pair, so they always agree.

Arguments

Name	Type	Req	Description
`gateway_id`	`GatewayId`	yes (path)	Gateway to query (any gateway, case-insensitive)

Reply

Status	Shape	Meaning
`200 OK`	`{ "gateway_id", "count", "positions": [{ "symbol", "net_qty", "avg_cost" }, ...] }`	Net position and average cost per symbol for the target gateway

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.13 `POST /api/v1/admin/symbols/{symbol}/cancel-orders`

Purpose: cancel all resting exposure on one symbol, across every gateway.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes (path)	Symbol to clear (case-insensitive)
`reason`	`Str`	no	Body, optional: note for the audit log

Reply

Status	Shape	Meaning
`200 OK`	kill-switch ack	What the engine cancelled
`409 Conflict`	error envelope	The engine refused the command

Status	Shape	Meaning

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`KILL_SWITCH_REJECTED`	`409`	Engine refused; `message` is its reason
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.14 `POST /api/v1/admin/gateways/{gateway_id}/kill-switch`

Purpose: cancel all resting exposure for one target gateway, without disconnecting it.

Arguments

Name	Type	Req	Description
`gateway_id`	`GatewayId`	yes (path)	Gateway whose orders and quotes to cancel (case-insensitive)
`symbol`	`Symbol`	no	Query: only this instrument; omit for all
`reason`	`Str`	no	Body, optional: note for the audit log

The target need not be connected, and the audit trail names the ADMIN.

Reply

Status	Shape	Meaning
`200 OK`	kill-switch ack	What the engine cancelled; `symbol` is echoed when the cancel was scoped

Status	Shape	Meaning
`409 Conflict`	error envelope	The engine refused the command

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`KILL_SWITCH_REJECTED`	`409`	Engine refused; `message` is its reason
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.15 `DELETE /api/v1/admin/gateways/{gateway_id}/quotes/{symbol}`

Purpose: withdraw one participant's two-sided quote on one symbol, leaving its other orders. Runs through the ADMIN kill switch (`quotes_only`), so it works whether or not that gateway is connected and the audit trail names the ADMIN.

Arguments

Name	Type	Req	Description
`gateway_id`	`GatewayId`	yes (path)	Whose quote (case-insensitive)
`symbol`	`Symbol`	yes (path)	Instrument (case-insensitive)

Reply

Status	Shape	Meaning
`200 OK`	kill-switch ack, `quotes_only: true`	The quote legs cancelled; `cancelled_quotes` is 0 when there was no quote

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`KILL_SWITCH_REJECTED`	`409`	Engine refused; `message` is its reason
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.16 `POST /api/v1/admin/market/halt`

Purpose: halt every symbol (halt source `ADMIN`) and cancel every market-maker quote. Resting orders stay on the book; `POST /admin/market/kill-switch` cancels those.

Arguments

Name	Type	Req	Description
none	—	—	No arguments

Reply

Status	Shape	Meaning
`200 OK`	`{ "accepted": true, "halted_symbols": 3, "halted_symbol_ids": [...], "cancelled_quotes": 4, ... }`	What the engine halted

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`HALT_REJECTED`	`409`	Engine refused; `message` is its reason

Code	Status	When
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.17 `POST /api/v1/admin/market/resume`

Purpose: lift the halt of `POST /admin/market/halt` on every symbol.

Arguments

Name	Type	Req	Description
none	—	—	No arguments

Reply

Status	Shape	Meaning
`200 OK`	`{ "accepted": true, "resumed_symbols": 3, "resumed_symbol_ids": [...], ... }`	What the engine resumed

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`RESUME_REJECTED`	`409`	Engine refused; `message` is its reason
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.18 `GET /api/v1/admin/market/volume`

Purpose: today's traded quantity, value and trade count per symbol, and the venue totals.

Arguments

Name	Type	Req	Description
none	—	—	No query parameters

Reply

Status	Shape	Meaning
`200 OK`	`{ "symbols": [{ "symbol", "qty", "value", "trades" }], "count": 2, "total_qty", "total_value", "total_trades" }`	Today's volume; `value` is display money

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.19 `POST /api/v1/admin/symbols/{symbol}/reopen`

Purpose: force one symbol to uncross now and clear any halt in one step. Without `price` it opens at the natural equilibrium; with one, at that operator price, for when a failed auction left no equilibrium. `dry_run` reports the indicative print without changing anything.

Arguments

Name	Type	Req	Description
`symbol`	`Symbol`	yes (path)	Instrument (case-insensitive)
`price`	`Price`	no	Body: operator opening price, on the tick grid (refused, not rounded, otherwise)
`dry_run`	`Bool`	no	Body: report the indicative print only; default `false`
`reason`	`Str`	no	Body: note for the audit log

Reply

Status	Shape	Meaning
`200 OK`	`{ "accepted": true, "symbol", "dry_run", "indicative_price", "indicative_qty", "surplus", "imbalance_side"?, "printed_price", "traded_qty" }`	The uncross result; `printed_price` is null on a dry run

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`REOPEN_REJECTED`	`409`	Engine refused (for example the symbol is not halted); `message` is its reason
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time
`SYMBOLS_NOT_READY`	`503`	`price` given before the symbol's tick precision reached the gateway

8.8.20 `POST /api/v1/admin/market/kill-switch`

Purpose: cancel all resting exposure across every gateway and symbol.

Arguments

Name	Type	Req	Description
`reason`	`Str`	no	Optional note

Reply

Status	Shape	Meaning
`200 OK`	kill-switch ack	What the engine cancelled
`409 Conflict`	error envelope	The engine refused the command

Errors

Code	Status	When
`VALIDATION`	`400`	A body field, query parameter, header or history filter failed validation; `field` names it. `reject_code` is one of `MISSING_FIELD`, `UNSUPPORTED_FIELD`, `INVALID_VALUE`, `TICK_VIOLATION`
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`KILL_SWITCH_REJECTED`	`409`	Engine refused; `message` is its reason
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.8.21 `POST /api/v1/admin/indexes/{index_id}/rebalance`

Purpose: rebalance one configured index through `pm-index`.

Arguments

Name	Type	Req	Description
`index_id`	`IndexId`	yes (path)	Index to rebalance
`updates`	`List<RebalanceUpdate>`	yes	Shares-outstanding updates

Reply

Status	Shape	Meaning
`200 OK`	rebalance ack	Update applied
`409 Conflict`	error envelope	Rebalance rejected

Errors

Code	Status	When
`VALIDATION`	`400`	Batch body invalid
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`REBALANCE_REJECTED`	`409`	`pm-index` rejected the request
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`INDEX_TIMEOUT`	`503`	`pm-index` did not reply in time

8.8.22 `POST /api/v1/admin/reference/reload`

Purpose: request a reference-data reload. The current engine rejects this request because the active configuration cannot be both reloaded and reconstructed from the audit trail. Deploy a compiled config artifact and restart the engine to apply configuration changes.

Arguments

Name	Type	Req	Description
none	—	—	No request body

Reply (if an engine accepts the request)

Status	Shape	Meaning
`200 OK`	`{ "status": "RELOADED", "config_version": "..." }`	Reload accepted by an engine that supports it
`409 Conflict`	error envelope	`RELOAD_REJECTED`: the current engine rejects reloads that cannot be recorded

Errors

Code	Status	When
`READ_ONLY`	`403`	A read-only key used on a gateway-scoped endpoint
`ROLE_DENIED`	`403`	Caller is not ADMIN
`RELOAD_REJECTED`	`409`	Reload would change the symbol or index set
`RATE_LIMIT`	`429`	Per-key write limit exceeded; retry after a short pause
`ENGINE_TIMEOUT`	`503`	Engine did not reply in time

8.9 Operational notes

- Read-only dashboard keys can call reference and public history endpoints, but not trading or admin write endpoints.
- `order_retention_sec` bounds the live order cache, the private `orders.snapshot` frame, and `GET /api/v1/admin/orders`.
- `market_data_cache_sec` bounds the market-data stream cache that backs the snapshot-on-subscribe, `snapshot`, and `resume` controls on `WS /api/v1/ws/market-data` (latest `book`/`depth`/`auction` snapshots are kept regardless of age; only the `trades` tail is bounded).
- WebSocket streams are specified in the WebSocket API Reference; this appendix is REST only.

CHAPTER 9

WebSocket API Reference

Status: Normative. This chapter is the contract for the four WebSocket streams of `pm-api-gwy` (`api_gateway/routers/ws.py`). It is versioned with the REST API under `/api/v1` and follows the same compatibility policy. For a tutorial with worked client code see API Gateway (REST/WebSocket). The key words MUST, MUST NOT, SHOULD and MAY are used per RFC 2119. `tests/test_ws_spec_matches_code.py` holds the catalogues below to the code.

9.1 Streams

Every socket lives under `/api/v1/ws/`, so a reverse proxy can route upgrades by that one prefix.

Path	Access	Purpose	Queue
`/api/v1/ws/private`	Trading key	The caller gateway's order, quote and risk events	`256`
`/api/v1/ws/market-data`	Any key	Public book, depth, trades, auctions, session, halts and news, by subscription	`4096`
`/api/v1/ws/admin/monitor`	ADMIN	Every engine event across all gateways except public market data and `engine.trace`, plus `admin.action`	`512`
`/api/v1/ws/admin/sim`	ADMIN	`pm-market-sim` state and the model's true values	`64`

Queue is the number of events the gateway holds for one socket that has not yet read them. See Flow control.

9.2 Connecting

1. The client opens the socket. The server accepts the upgrade.
2. The client MUST send, as its first frame and within **5 seconds**, a JSON object `{"api_key": "<key>"}`.
3. On success the server sends `{"type": "authenticated", ...}` and then the socket's opening snapshot, if it has one, before any live event.

The checks, in order, and what a failure looks like:

Check	Sockets	On failure
A first frame arrives within 5 s and is JSON	all	close `1008`
`api_key` is a configured key	all	close `1008`
The key is a trading key (has a `gateway_id`)	`private`, `admin/monitor`, `admin/sim`	`error` frame (admin sockets), then close `1008`
The engine accepts the gateway's identity	`private`, `admin/monitor`	`error` frame with the engine's reason, then close `1008`
The gateway holds the ADMIN role	`admin/monitor`, `admin/sim`	`error` frame `"ADMIN role required"`, then close `1008`

9.2.1 Close codes

The gateway itself closes a socket with exactly one code:

Code	Meaning
`1008`	Policy violation: authentication or authorisation failed (see above). The client MUST NOT reconnect with the same key without fixing the cause

Any other closure (a normal close, the server shutting down, or a dropped connection with no close frame) is not a verdict on the client: it SHOULD reconnect with backoff and recover as described for each socket.

9.3 The event envelope

Every engine event on every socket arrives in one envelope:

```
{
  "type": "order.fill",
  "topic": "order.fill.TRADER01",
  "seq": 4127,
  "stream_seq": 88,
  "ts": "2026-06-24T10:15:03.221Z",
  "gateway_id": "TRADER01",
  "data": { "order_id": "ORD-...", "fill_qty": 50, "fill_price": 150.5 }
}
```

Field	Present	Meaning
`type`	always	The stable event name; see Event types
`topic`	always	The engine topic the event came from
`seq`	engine events	Per-`topic` counter, assigned once per event by this gateway process
`stream_seq`	`private` only	Per-gateway counter across the whole private stream
`ts`	always	When this gateway sent the event, UTC, millisecond ISO-8601
`gateway_id`	gateway-scoped events	The gateway the event belongs to
`data`	always	The engine message body; its schema is the linked message reference

Server frames that are not engine events (`authenticated`, snapshots, the market-data control replies, `error`) carry `type` and `data` and only the other fields their row below names.

9.3.1 Sequencing

- `seq` counts per `topic`, within one gateway process. A client checking for gaps MUST key on `topic`, not `type`: one type (`depth`) spans many topics (`depth.AAPL`, `depth.MSFT`), each numbered independently.
- `trade.executed` is one topic for every symbol, so its `seq` is contiguous only on a `*` trades subscription. A subscription to some symbols' trades sees jumps where other symbols traded; it cannot detect a drop from `seq` and SHOULD subscribe to `*` when it needs to.
- `stream_seq` counts every event on one gateway's private stream, so on `/ws/private` it is contiguous: a gap of any size means events were lost.
- An event is numbered before it is queued, so an event dropped for a slow socket still consumes its number. A gap is therefore always detectable.
- Counters restart when the gateway process restarts. A client SHOULD treat a `seq` lower than the last one seen on a topic as a restart, not a gap.

9.3.2 Event types

`type` is derived from `topic`: the first two dot-separated segments (`order.fill.TRADER01` gives `order.fill`), with these exceptions:

Topic	`type`
`trade.executed`	`trade`
`book.{symbol}`	`book`
`depth.{symbol}`	`depth`

Topic	type
session.state	session
circuit_breaker.*	circuit_breaker
auction.result.{symbol}	auction
auction.indicative.{symbol}	auction.indicative
risk.kill_switch_ack.{gateway_id}	mass_cancel.ack
admin.action.{gateway_id}	admin.action

A client MUST ignore an event whose `type` it does not recognise. New types may be added within v1.

9.4 Flow control

Each socket has a bounded queue (see Streams). The gateway never blocks the engine feed for a slow client: when a socket's queue is full, the event is **dropped for that socket** and the socket stays open.

- Every drop is counted per stream (`private`, `market_data`, `admin`) in `GET /api/v1/healthz` as `dropped_events` and in the gateway's metrics as `api.dropped_events`.
- The client detects a drop from a `seq` or `stream_seq` gap and recovers as its socket describes below.
- `/ws/admin/sim` and `news` events are not numbered. Each `sim` frame supersedes the previous one, so a dropped one needs no recovery.

9.5 `/api/v1/ws/private`

The caller gateway's own events, unfiltered.

9.5.1 Opening frames

type	Fields	Meaning
authenticated	gateway_id, stream_seq	Authenticated; stream_seq is the stream position now
orders.snapshot	gateway_id, stream_seq, ts, data: {orders, positions, quote_legs}	The gateway's view of the caller's orders, positions and quote legs, accurate as of stream_seq

The event sink is registered before the snapshot is taken: the worst case is an event both in the snapshot and delivered live, never one in neither. Order state is idempotent, so a client MAY apply such a duplicate twice.

9.5.2 Private event catalogue

type	Engine topic	Payload
order.ack	order.ack.{gateway_id}	order.ack
order.fill	order.fill.{gateway_id}	order.fill
order.amended	order.amended.{gateway_id}	order.amended
order.cancelled	order.cancelled.{gateway_id}	order.cancelled
order.expired	order.expired.{gateway_id}	order.expired
order.triggered	order.triggered.{gateway_id}	order.triggered
order.orders	order.orders.{gateway_id}	order.orders
combo.ack	combo.ack.{gateway_id}	combo.ack
combo.status	combo.status.{gateway_id}	combo.status
oco.ack	oco.ack.{gateway_id}	oco.ack
oco.cancelled	oco.cancelled.{gateway_id}	oco.cancelled
quote.ack	quote.ack.{gateway_id}	quote.ack
quote.status	quote.status.{gateway_id}	quote.status
oco.cancel_ack	oco.cancel_ack.{gateway_id}	oco.cancel_ack
combo.cancel_ack	combo.cancel_ack.{gateway_id}	combo.cancel_ack
quote.cancel_ack	quote.cancel_ack.{gateway_id}	quote.cancel_ack
mass_cancel.ack	risk.kill_switch_ack.{gateway_id}	risk.kill_switch_ack
system.gateway_auth	system.gateway_auth.{gateway_id}	system.gateway_auth
system.symbols	system.symbols.{gateway_id}	system.symbols
system.quote_bootstrap	system.quote_bootstrap.{gateway_id}	system.quote_bootstrap
system.quote_legs	system.quote_legs.{gateway_id}	system.quote_legs
system.session_status	system.session_status.{gateway_id}	system.session_status

Each OCO, combo or quote cancel is answered by its own `*.cancel_ack`, accepted or refused, echoing the cancel's `request_tag`; the submit acks (`oco.ack`, `combo.ack`, `quote.ack`) report submissions only.

The `order.*` events carry the order's `client_tag`, and cancels and amends their `request_tag` (see the REST spec's order correlation).

9.5.3 Recovery

On reconnect the server sends a fresh `orders.snapshot`. A client MUST rebuild its order state from it, then apply live events with a higher `stream_seq`. There is no replay of missed private events: the snapshot is the recovery. For history, use `GET /api/v1/history/orders` and `GET /api/v1/history/fills`.

9.6 `/api/v1/ws/market-data`

Public events, filtered by the socket's subscription. Any configured key may connect, including a read-only one.

9.6.1 Channels

A client subscribes to `(symbol, channel)` pairs. Three channels are always on: their events reach every market-data socket without a subscription, because a halt or a phase change alters the meaning of every other channel.

Channel	Subscribable	Event `type`s	Engine topics
`book`	yes	`book`	`book.{symbol}`
`depth`	yes	`depth`	`depth.{symbol}`
`trades`	yes	`trade`	`trade.executed`
`auction`	yes	`auction`, `auction.indicative`	`auction.result.{symbol}`, `auction.indicative.{symbol}`
`session`	always on	`session`	`session.state`
`circuit_breaker`	always on	`circuit_breaker`	`circuit_breaker.halt.{symbol}`, `.extend.{symbol}`, `.resume.{symbol}`
`news`	always on	`news`	`news.event` (from `pm-market-sim`; not numbered)

9.6.2 Control frames

The client sends control frames as JSON objects at any time after `authenticated`.

`action`	Fields	Effect	Reply
`subscribe`	`symbols`, `channels` (flat form) and/or `items`	Add the pairs; then send the current snapshot of each	`subscription`, then snapshot events
`unsubscribe`	as `subscribe`	Remove the pairs	`subscription`
`snapshot`	as `subscribe`	Re-send the current snapshot, subscription unchanged	snapshot events
`resume`	`topic`, `from_seq`, optional `symbols`	Replay one topic after `from_seq`	replayed events, or `resume.rejected`

- **Flat form:** `{"action": "subscribe", "symbols": ["AAPL"], "channels": ["book", "trades"]}` is the cross product of its two lists.
- **Items form:** `"items": [{"symbols": ["*"], "channels": ["book"]}, {"symbols": ["AAPL"], "channels": ["depth"]}]` is a list of independent rules: book for every symbol, depth for AAPL only.
- An empty `symbols` list, or `["*"]`, means every symbol.
- An item may carry `resume_from: {"trades": <seq>}`; on `subscribe` and `snapshot` the `trades` channel then replays buffered prints after that `seq` instead of sending the whole tail.
- A malformed frame, or one that fails validation (an unknown `action` or channel, an unknown field), is answered with an `error` frame and otherwise ignored; the socket stays open.

9.6.3 Server control frames

type	data	Sent
authenticated	none	After authentication
subscription	items (effective rules, one per symbol), always (the always-on channels), rejected	After every subscribe/unsubscribe
resume.rejected	topic, from_seq, reason	A resume (or resume_from) that cannot be served
trades.reset	symbol	Before a fresh trade tail, when a symbol's trades could not be replayed
error	message	A control frame could not be parsed or validated

subscription.data.rejected lists the items that contributed nothing:

reason	Meaning
no_channels	The item named no channel
wildcard_still_subscribed	An unsubscribe of a named symbol that a * rule still covers

resume.rejected.data.reason:

reason	Meaning
unknown_topic	The topic is not a market-data topic, or there is nothing cached for it
too_old	from_seq is older than the replay window; a fresh tail follows

9.6.4 Recovery

- **Snapshot channels** (book, depth, auction) are self-healing: the current snapshot replaces whatever was missed. After a gap or a reconnect, send snapshot (or subscribe) for the pairs concerned. The book snapshot is also available without a socket as GET /api/v1/book/{symbol}, whose seq is that of the book frame for the same snapshot.
- **Trades** are replayable for the gateway's replay window (market_data_cache_sec, 60 seconds by default): resume with topic: "trade.executed", the last processed seq and the symbol. Past the window the server sends trades.reset and resume.rejected (too_old), then the current tail. A client SHOULD de-duplicate trades by their id across a replay.
- session and circuit_breaker state after a reconnect is available from GET /api/v1/session and GET /api/v1/halts.

9.7 /api/v1/ws/admin/monitor

Every engine event this gateway receives, across all gateways, in the standard envelope, plus `admin.action`. ADMIN only.

Two families are left out: public market data (`book`, `depth`, `trade`, `auction`, `auction.indicative`), which an ADMIN reads on `/api/v1/ws/market-data`, and the audit-only `engine.trace.{symbol}`. Under load these were about half of the feed, and no monitor renders them. Whole topics are omitted, so the per-topic `seq` of everything that is sent stays contiguous.

9.7.1 Opening frames

type	Fields	Meaning
`authenticated`	none	Authenticated
`monitor.snapshot`	`ts`, `data: {orders, halts, gateways, last_seq, incomplete}`	Current state: the gateway's cross-gateway order view, the engine's halts and gateway roster, each active gateway's `stream_seq`; names of any part that timed out in `incomplete`

9.7.2 Events

All private event types above, for every gateway, `session` and `circuit_breaker`, and:

type	Engine topic	Payload
`admin.action`	`admin.action.{gateway_id}`	admin.action: one admin-gated command, who ran it and what it did

Other engine topics (system replies addressed to any gateway, index and reference events) also arrive here, with `type` derived as in Event types. A monitor MUST ignore types it does not render.

9.7.3 Recovery

None by replay: on reconnect the snapshot is current state. History is the audit trail (`GET /api/v1/admin/orders/{orde` `pm-audit-cli`).

A `seq` gap on a live connection means the gateway dropped events because this client's queue was full. A dropped fill or cancel leaves an order looking working. The monitor SHOULD repair its working-order view from `GET /api/v1/admin/orders?status=…` (one call per working status) rather than by reconnecting: the opening snapshot also carries every cached terminal order, which under the load that caused the gap is far larger than the working set.

9.8 /api/v1/ws/admin/sim

`pm-market-sim`'s state and the model's true (hidden) fair values. ADMIN only. Frames carry `type` and `data` only, and are not numbered.

type	data	Sent
authenticated	none	After authentication
sim.snapshot	{available, state, values}, as GET /api/v1/admin/sim	Opening state
sim.state	sim.state	Each simulator heartbeat
sim.value	sim.value	Each step's true values

Each frame supersedes the previous one; a slow viewer simply misses steps.

CHAPTER 10

Market Data Feed (CALF)

Learning objectives

After reading this page you will understand:

- What CALF is and why a market-data protocol is separate from order entry
- How CALF channels expose different levels of market information
- How `TOP`, `TRADE`, and `STATE` complement each other
- How participants subscribe and stay synchronized using a generic text protocol
- How sequence numbers and replay make reconnect behavior deterministic
- How the same CALF model can disseminate derived index calculations

10.1 Why CALF Exists

EduMatcher already provides order-entry protocols:

- `ALF`: simple text order entry
- `BALF`: binary order entry

`CALF` adds the missing piece: **market data distribution**.

Order entry answers: "How do I submit intent to trade?" Market data answers: "What is happening in the market right now?"

`CALF` is designed as a readable, line-based TCP protocol so learners can inspect it in a terminal while still following exchange-grade ideas:

- channelized subscriptions
- snapshots plus incrementals
- sequence-based gap detection
- bounded replay recovery

10.2 Information Levels by Channel

CALF separates data into channels so clients can subscribe only to what they need.

Channel	What it tells you	Typical client use
`TOP`	Best bid/ask, top sizes, last-trade fields (`SNAP` + `MD`)	UI book widgets, spread monitoring, execution bots

Channel	What it tells you	Typical client use
`TRADE`	Executed prints (`TRADE`)	Tape display, strategy triggers, volume analytics
`STATE`	Session/instrument state transitions (`STATE`)	Session-aware bots, operational monitors

Think of the three channels as answering increasingly specific questions about the market:

Question	Channel
Is trading open, auction, halted, or closed?	`STATE`
What is the current best bid, best ask, and spread?	`TOP`
What price and size just executed, and who was the aggressor?	`TRADE`



The channels are **independent subscriptions** — you can take any combination depending on what your client needs.

10.3 Core Message Types

CALF uses one UTF-8 line per message (`\n` terminated):

```
<MSGTYPE>|KEY=VALUE|KEY=VALUE|...\n
```

The messages you will encounter first as a client:

Message	Who sends it	Purpose
`HELLO`	Client	Start the session
`WELCOME`	Gateway	Confirm session, advertise parameters
`SUB`	Client	Subscribe to channels and symbols
`SNAP`	Gateway	Point-in-time baseline for a stream
`MD`	Gateway	Incremental top-of-book change
`TRADE`	Gateway	Executed trade print
`STATE`	Gateway	Session or instrument state transition
`HB`	Gateway	Heartbeat when no data is flowing
`ERR`	Gateway	Error notification

SNAP is a message type, not a channel

You subscribe to a channel (`TOP` or `STATE`). The gateway then automatically sends a `SNAP` baseline for that channel. Clients never subscribe to `SNAP` directly.

10.4 How Participants Subscribe

A participant opens a TCP connection and follows a small, generic flow:

1. Send `HELLO`
2. Receive `WELCOME`
3. Send one or more `SUB` commands
4. Process `SNAP` baseline(s)
5. Process live incrementals (`MD`, `TRADE`, `STATE`)

Annotated example — each line is one complete CALF message:

```
# Client identifies itself and requests CALF version 1
HELLO|CLIENT=bot01|PROTO=CALF1

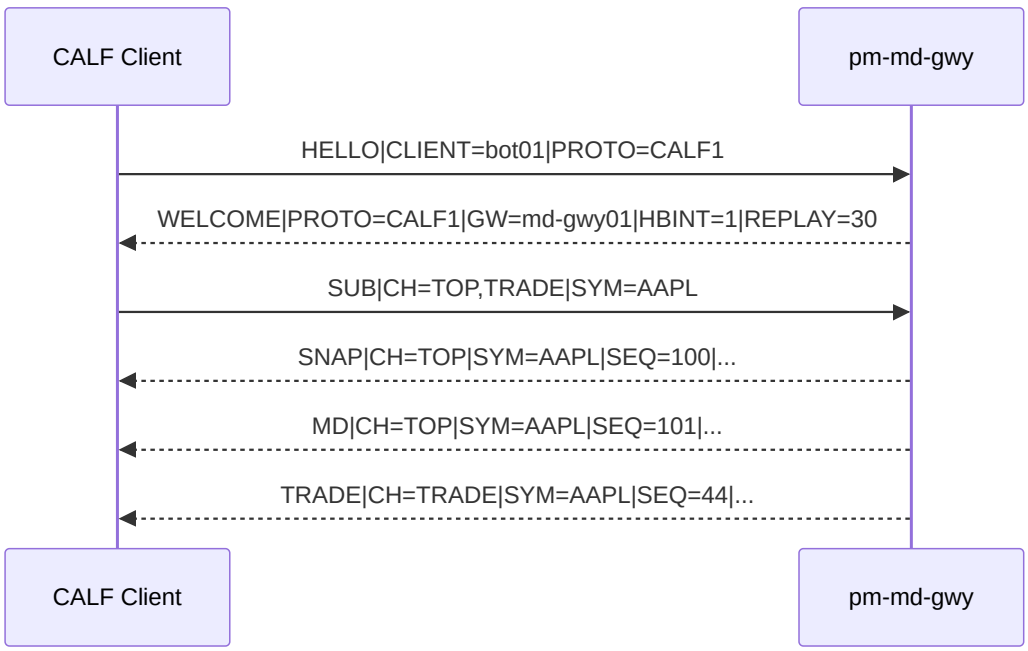
# Gateway confirms; HB every 1 s, replay window 30 s, known symbols AAPL and MSFT
WELCOME|PROTO=CALF1|GW=md-gwy01|HBINT=1|REPLAY=30|SYMBOLS=AAPL,MSFT

# Client subscribes to TOP (best bid/ask) and TRADE (fills) for AAPL
SUB|CH=TOP,TRADE|SYM=AAPL

# Gateway sends snapshot: current top-of-book at SEQ=100
# BID=150.10 x 1200, ASK=150.12 x 900, last trade was 150.11 x 300
SNAP|CH=TOP|SYM=AAPL|SEQ=100|TS=2026-06-07T10:16:00.000Z|BID=150.10|BIDSZ=1200|ASK=150.12|ASKSZ=900|LAST=150.

# The best bid just moved to 150.11 x 1400 — only changed fields are sent
MD|CH=TOP|SYM=AAPL|SEQ=101|TS=2026-06-07T10:16:00.115Z|BID=150.11|BIDSZ=1400

# A trade executed: 200 shares at 150.12, buy-side was the aggressor
TRADE|CH=TRADE|SYM=AAPL|SEQ=44|TS=2026-06-07T10:16:00.141Z|PX=150.12|QTY=200|SIDE=BUY
```

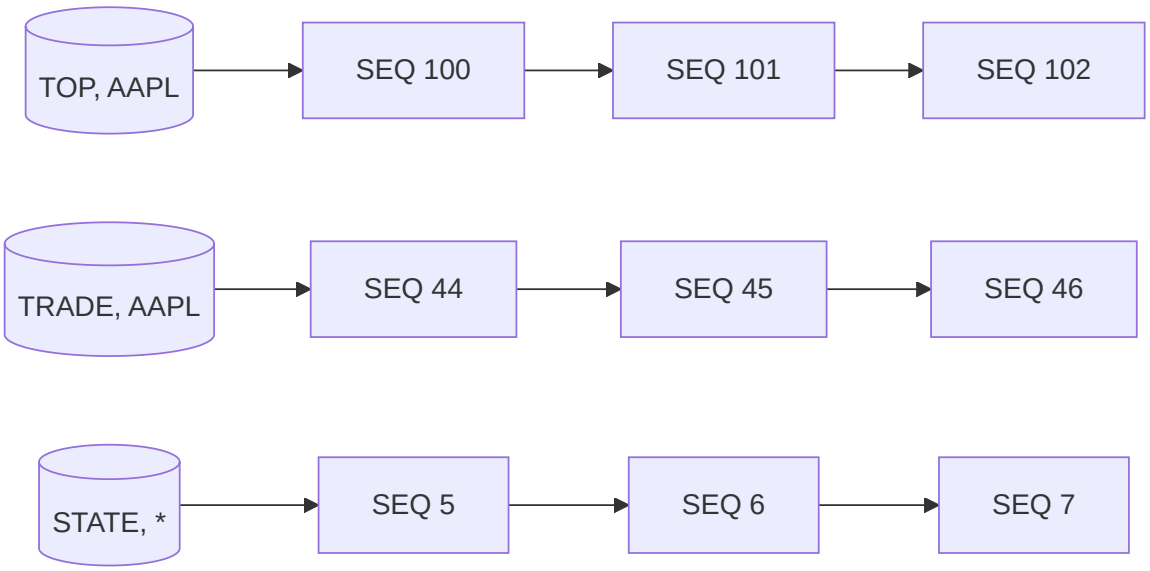


10.5 Staying Up To Date Reliably

`CALF` uses monotonic `SEQ` numbers per `(CH, SYM)` stream.

That means sequence tracking is per stream, not global:

- `(TOP, AAPL)` has one independent counter
- `(TRADE, AAPL)` has another
- `(STATE, *)` has its own



Client rule:

- If current `SEQ != previous + 1`, you detected a gap
- Attempt replay when possible
- If replay misses the window, accept fresh `SNAP` and continue

This gives deterministic behavior on reconnect without requiring full historical storage.

10.6 First Connect vs Reconnect

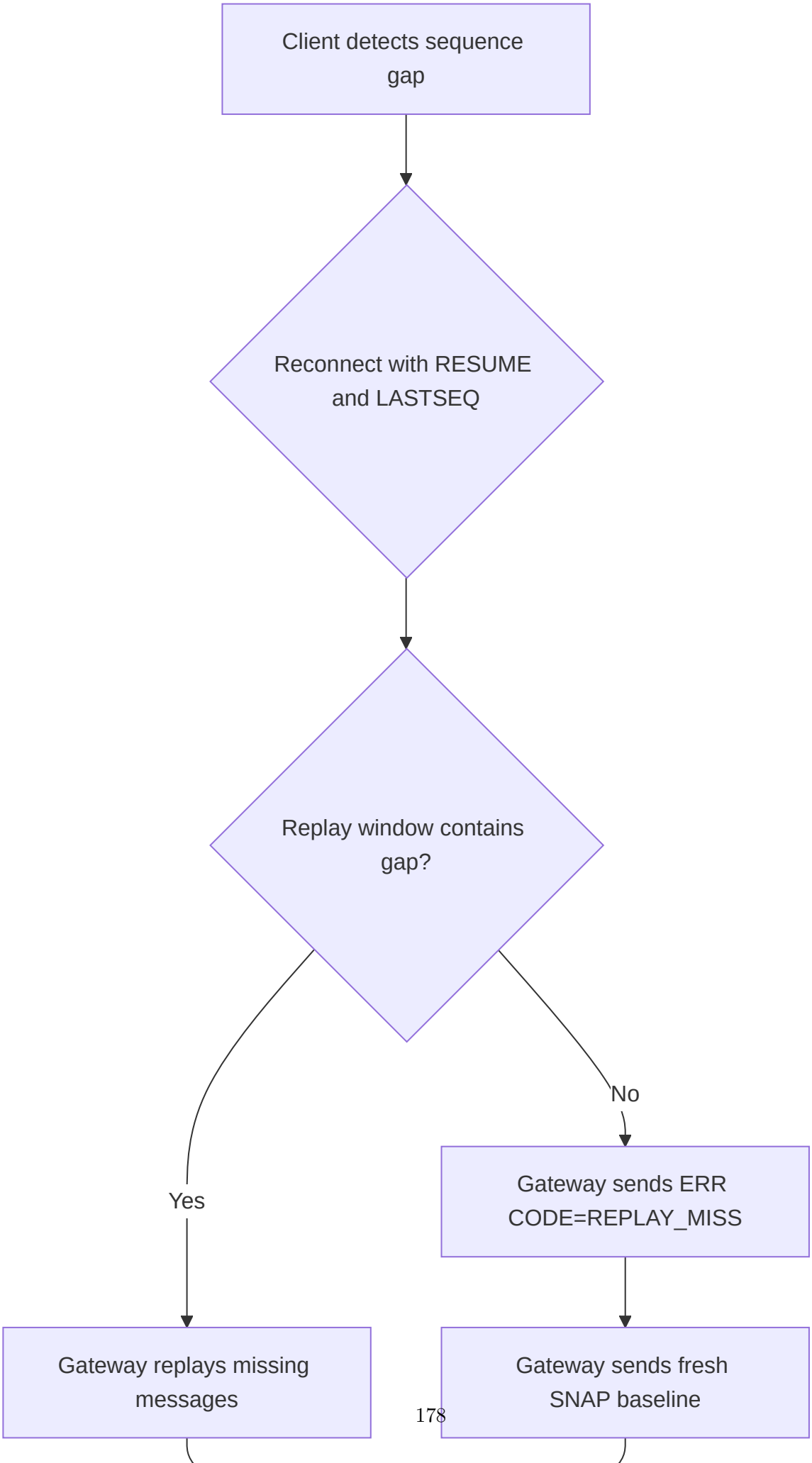
10.6.1 First connect

When you connect for the first time, the gateway has no history to replay for you. Instead it sends a `SNAP` — a complete point-in-time picture of the current state for each `(CH, SYM)` pair you subscribed to. Record the `SEQ` from that `SNAP`. Every incremental message (`MD`, `TRADE`, `STATE`) that follows will have `SEQ = last_seen + 1`. As long as the sequence is gapless you are fully in sync.

10.6.2 Reconnect

- Send a plain `HELLO`, then one `RESUME|CH=..|SYM=..|LASTSEQ=..` per stream you were following. `RESUME` is a standalone, repeatable command — as a `HELLO` flag it could only ever run once per connection.
- If the gap is within the replay window, the gateway replays the missing messages. Note it replays *everything* past `LASTSEQ`, so the reply also re-sends messages you already have: discard anything at or below the `SEQ` you last recorded.
- If not, the gateway sends `ERR|CODE=REPLAY_MISS`, then a fresh `SNAP` on `TOP`/`STATE`/`INDEX`/`DEPTH`/`CB`. On `TRADE` and `AUCTION` no `SNAP` follows — there is no snapshot of a print that already happened,

so those events are gone.
- A `SNAP` re-baselines the stream; it is never itself a gap.

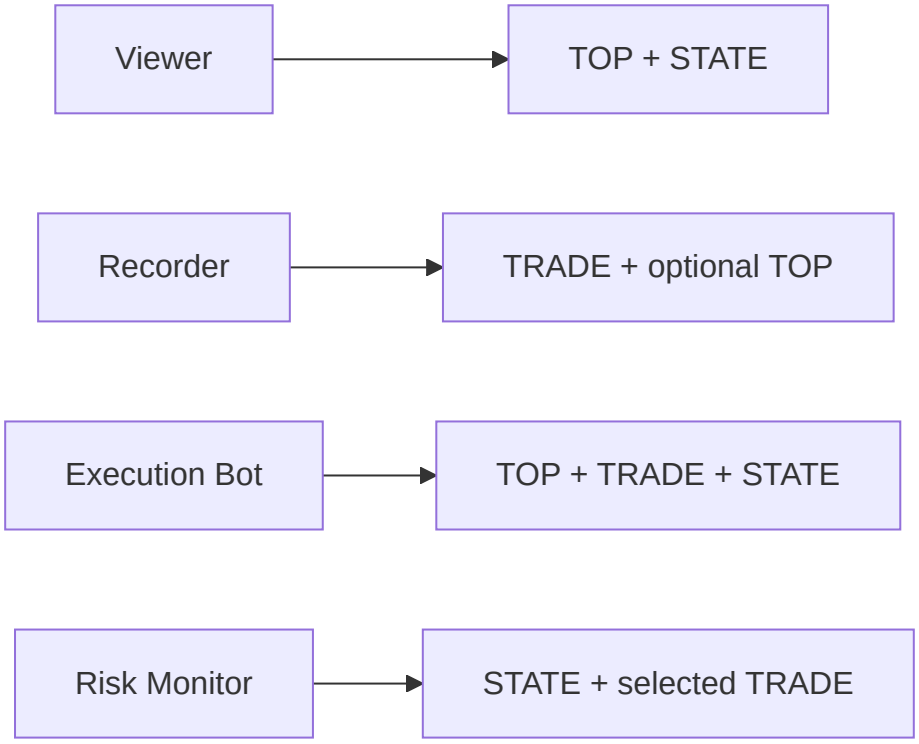

Client detects sequence gap
Reconnect with RESUME and LASTSEQ
Replay window contains gap?
Yes
No
Gateway replays missing messages
Gateway sends ERR CODE=REPLAY_MISS
Gateway sends fresh SNAP baseline

This keeps client logic straightforward: always restore a known baseline, then continue incrementally.

10.7 Why This Matters For Different Participants

Different participant types consume different CALF slices:

- Viewer: `TOP` + `STATE` for spread, best levels, and market phase
- Recorder: `TRADE` (and often `TOP`) for post-trade analysis
- Execution bot: `TOP` + `TRADE` + `STATE` for session-aware strategy decisions
- Risk monitor: `STATE` + selected `TRADE` streams for halt/session supervision



The same protocol works for all, with different subscription sets.

10.8 CALF and Index Dissemination

The `CALF` channel model is also suitable for **disseminating index calculations**.

Conceptually, an index feed is just another ordered stream of derived market facts:

- index value
- timestamp
- optional contribution metadata
- sequence for gap detection/recovery

In EduMatcher, this can be integrated in the same operational pattern as other CALF streams, so clients can consume index updates with the same subscription, sequence, and replay semantics they already use for `TOP`/`TRADE`/`STATE`.

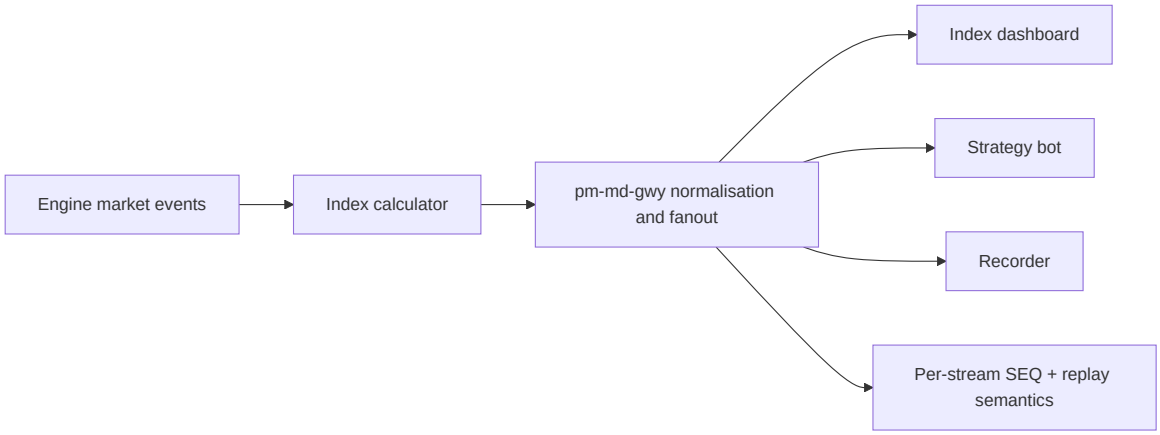
Practical options include:

- publishing index values as a dedicated CALF stream in the gateway
- mapping index updates into `STATE`-style or dedicated symbol streams for consumers

A client subscribing to index data would follow the same handshake and sequence rules as any other CALF stream. For example, an index update might look like:

```
SUB|CH=TOP|SYM=EDU-INDEX
SNAP|CH=TOP|SYM=EDU-INDEX|SEQ=1|TS=2026-06-07T10:00:00.000Z|LAST=1042.30
MD|CH=TOP|SYM=EDU-INDEX|SEQ=2|TS=2026-06-07T10:00:05.000Z|LAST=1043.10
```

The key idea: **one generic subscriber model for both raw market events and derived index events**.



10.9 Minimal Client Checklist

For a robust CALF client implementation:

1. **Parse line-by-line** — TCP is a byte stream; a single `recv()` can contain part of a line, one full line, or several lines at once. Buffer and split on `\n`.
2. **Track `last_seq` per stream** — keep a dict keyed by `(CH, SYM)` and check every incoming `SEQ`.
3. **Treat missing fields in `MD` as unchanged** — if an `MD` message has no `ASK` field, the ask price has not moved since the last `SNAP` or `MD` that carried it. Do not reset it to zero.
4. **Handle `HB`** — a heartbeat means the gateway is alive but quiet. If neither data nor `HB` arrives, the connection may be dead.
5. **On gap, attempt replay; on replay miss, reset from `SNAP`** — send `RESUME|CH=<ch>|SYM=<sym>|LASTSEQ=<n>`, and drop any replayed message at or below the `SEQ` you already recorded. If you get `ERR CODE=REPLAY_MISS` instead, accept the fresh `SNAP` that follows on the snapshot-backed channels; on `TRADE`/`AUCTION` none follows and the gap is permanent.
6. **Keep subscriptions narrow** — only subscribe to `(CH, SYM)` pairs your process actually uses.

10.10 Summary

CALF gives EduMatcher a readable but production-shaped market-data model:

- channelized information levels (`TOP`, `TRADE`, `STATE`)
- snapshot baseline + incremental updates
- deterministic sync via per-stream sequences
- bounded replay for short disconnect recovery
- reusable dissemination pattern for derived outputs such as index calculations

This makes CALF a practical bridge between educational clarity and realistic feed behavior.

Glossary →

CHAPTER 11

Drop Copy

Learning objectives

After reading this page you will understand:

- What drop copy is and why it exists in real exchange ecosystems
- How EduMatcher's drop copy publisher works
- The message format published on the drop copy channel
- What replay support exists today and what its limits are
- How to configure and monitor the drop copy stream
- Three ways to consume it: connecting directly to port 5557, relaying it through your own ALF session with `DC|STATE=ON`, or via the standalone `pm-dc-gwy` TCP gateway for non-ZeroMQ clients
- How this compares to how real exchanges deliver drop copy

Prerequisite: Processes, Environment and Ports gives an overview of the ZeroMQ topology that drop copy sits within.

11.1 What is drop copy?

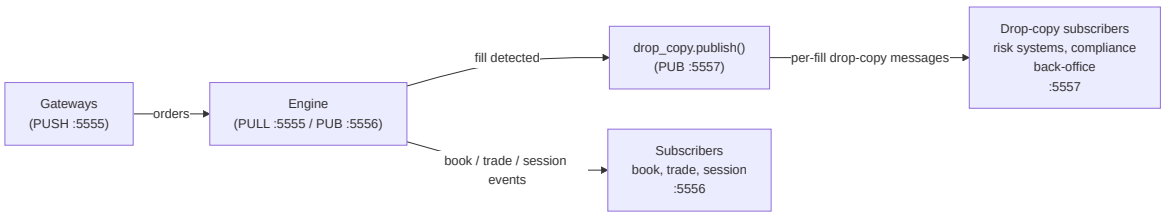
Drop copy is the name for a secondary stream of trade and fill event notifications sent to risk management systems, compliance monitors, and back-office infrastructure in parallel with the primary trade confirmation path.

In real exchange connectivity the "drop" in drop copy refers to the idea that a copy of every execution report is *dropped* to a side channel as it is processed, without the recipient needing to be part of the order flow itself. This separation is important:

- A risk system must receive fills even when the originating gateway has disconnected or failed.
- A compliance monitor should see every fill across all gateways in a single stream, not one per gateway connection.
- Back-office systems need guaranteed delivery with sequence numbers so they can detect gaps and request replays.

11.2 Architecture

EduMatcher's drop copy publisher runs inside the matching engine in `src/edumatcher/engine/drop_copy.py`. It binds a dedicated ZeroMQ PUB socket on port **5557** and publishes events as the engine processes fills.



The drop copy socket is **lazily initialised** in `Engine.run()`. It is not created during `__init__`, which means tests that instantiate `Engine` without calling `run()` do not bind a ZMQ socket.

11.2.1 Socket addresses

Socket	Address	Purpose
Engine PULL	`tcp://127.0.0.1:5555`	Engine receives orders from gateways
Engine PUB	`tcp://127.0.0.1:5556`	Engine broadcasts book snapshots, trades, etc.
Drop copy PUB	`tcp://127.0.0.1:5557`	Engine broadcasts fill events to risk systems

The engine actually **binds** this socket to `DROP_COPY_PUB_BIND_ADDR` in `src/edumatcher/config.py` (driven by `EDUMATCHER_ENGINE_BIND_HOST`). `DROP_COPY_PUB_ADDR` is a separate, connect-side constant (driven by `EDUMATCHER_ENGINE_HOST`) that consumers such as `pm-dc-gwy` and `pm-dc-spy` use to *reach* the engine — editing it does not change where the engine listens.

No authentication or entitlement checks

The drop copy PUB socket performs no authentication, and there is no per-gateway entitlement check on who may subscribe. Any process that can reach `tcp://127.0.0.1:5557` can subscribe to fills for **every** gateway, regardless of the topic prefix it filters on. This differs from the RALF gateway's `DROP_COPY` channel (see RALF DROP_COPY channel vs. this feed below), which does enforce per-connection role entitlement.

11.3 Message format

Every drop copy message is a two-frame ZeroMQ multipart message:

```
Frame 0 (topic):    b"drop_copy.event.<gateway_id>"
Frame 1 (payload):  orjson-encoded JSON object
```

Payloads are serialized with `orjson` when it is installed (the normal case); `edumatcher.models.message` falls back to the stdlib `json` module if `orjson` is unavailable, so the wire format is identical either way.

11.3.1 Payload fields

Field	Type	Description
`seq`	int	Monotonically increasing per-gateway sequence (starts at 1; resets when the engine restarts)
`timestamp`	int	Nanosecond timestamp from `now_ns()`
`gateway_id`	str	ID of the gateway that submitted the order

Field	Type	Description
`event_type`	str	Type of event (currently `"order.fill"`)
`order_id`	str	The order this execution belongs to
`trade_ids`	list[str]	Public `trade.executed` ID(s) for this execution — one today, kept as a list for parity with coalesced `order.fill` messages
`symbol`	str	Instrument ticker
`fill_qty`	int	Executed quantity
`fill_price`	float	Display money, not ticks
`liquidity_flag`	str	`"TAKER"` for the aggressor, `"MAKER"` for the resting side

The field set is fixed, not open. It used to be described as "event-specific data merged into the top-level object", because `DropCopyPublisher.publish` took a `payload` dict — but the single producer has always sent these five keys, and since phase 6.1d the publisher takes them as named arguments and the spec declares them. A second event type is a spec change, not a new key.

11.3.2 `order.fill` event

Published whenever a trade occurs. This is the only live drop-copy event type currently emitted by the engine. Every trade produces **two** drop-copy events — one per counterparty — so the example below represents one side of a matched MAKER/TAKER pair, not the whole trade.

```
{
  "seq": 42,
  "timestamp": 1700000000000000000,
  "gateway_id": "TRADER01",
  "event_type": "order.fill",
  "order_id": "ord-001",
  "trade_ids": ["000042-000000123"],
  "symbol": "MSFT",
  "fill_qty": 100,
  "fill_price": 420.0,
  "liquidity_flag": "MAKER"
}
```

Price units and fields

`fill_price` is published as a display-price float, not as raw integer ticks. `liquidity_flag` is derived from the trade's aggressor side (`"TAKER"` for the aggressor, `"MAKER"` for the resting side). The payload does not include `remaining_qty`, `side`, or `leaves_qty`.

Trade correlation

`trade_ids` connects a private fill to the public `trade.executed` tape. Drop-copy events are emitted once per execution, so the list currently has one durable trade ID. An aggressive order that sweeps several price levels instead receives one coalesced private `order.fill` with all of those IDs in execution order; concatenating its corresponding drop-copy `trade_ids` produces the same list. Consumers should treat this as a list, not a scalar.

This is fed from the engine's single trade-publication path (`engine/main.py`, `_publish_trade`), so it covers every fill-producing flow — new orders, quotes, combo legs, OCO legs, auction uncrosses, stop cascades, and amend-rematches — not just plain new-order fills.

11.4 Sequence numbers

The `seq` field is an independent counter for each `gateway_id`. It starts at 1 for each gateway and increments by 1 for every event in that gateway's `drop_copy.event.<gateway_id>` stream. Each gateway's counter restarts at 1 when the engine restarts. A subscriber to one gateway can therefore detect gaps and duplicates directly; a subscriber to multiple gateways must track a separate last-seen sequence for each gateway.

Subscriptions match by prefix

Subscribing to `drop_copy.event.TRADER1` also delivers the events of `TRADER10`, `TRADER11`, … — each with its own `seq` stream. A consumer that checks sequences must compare the topic (or the payload's `gateway_id`) exactly, or it will report false gaps.
Sequence numbers allow downstream consumers to:

1. **Detect gaps** — if the consumer receives `seq=5` after `seq=3`, it knows `seq=4` was missed and should request a replay.
2. **De-duplicate** — if a message is received twice (due to a reconnect replay), the consumer can discard the duplicate by checking whether the `seq` has already been processed.

11.5 In-memory buffer

The drop copy publisher maintains one **circular in-memory buffer** of the last 10,000 events, shared by all gateways. This buffer supports replay of a gateway's events after a gap.

The buffer is implemented as a `collections.deque` with `maxlen=10_000`. When the buffer is full, the oldest event is discarded automatically, whichever gateway it belongs to — so a busy gateway shortens a quiet gateway's history.

At a sustained rate of ~10 fills/second, 10,000 buffered messages covers roughly 16 minutes of history before the oldest events start being dropped.

11.6 Replay

A downstream system that loses its connection and reconnects can request a replay of events it missed.

In-process only — no external replay protocol

Replay is implemented as a publisher method (`DropCopyPublisher.replay(recipient_id, from_seq, gateway_id=...)`). There is currently **no** ZMQ message handler that accepts replay requests from external consumers. It is useful for tests and embedded consumers but not yet a full reconnect protocol.

11.6.1 Replay request (programmatic)

Call `DropCopyPublisher.replay(recipient_id, from_seq, gateway_id=...)` from within the engine. This publishes buffered events for that gateway with `seq >= from_seq` on a dedicated replay topic. `gateway_id` is required: `from_seq` only means something within one gateway's stream.

```
Frame 0 (topic):    b"drop_copy.replay.<recipient_id>"
Frame 1 (payload):  same format as live events
```

The call returns the number of messages replayed.

11.6.2 Replay topic design

Replay messages use a different topic prefix (`drop_copy.replay.*`) from live messages (`drop_copy.event.*`). A consumer subscribes to its own replay topic so that replay messages from one reconnecting client do not pollute the live stream received by other consumers.

11.6.3 Replay limitations

The buffer holds only the last 10,000 events across all gateways. If a consumer has been disconnected long enough that its `from_seq` predates the oldest buffered message for that gateway, it will receive only the available subset. In production systems this gap would typically be filled from a persistent audit log (see Persistence).

11.7 Topic subscription patterns

ZeroMQ PUB/SUB uses prefix-based topic filtering. Common subscription patterns for the drop copy stream:

Pattern	Receives
`b"drop_copy.event."`	All live fill events from all gateways
`b"drop_copy.event.TRADER01"`	Live fills from TRADER01 — and from any ID it prefixes, such as TRADER010
`b"drop_copy.replay.MY_RISK_SYS"`	Replay messages addressed to MY_RISK_SYS

11.8 Startup and shutdown

The drop copy publisher is created in `Engine.run()` before the main event loop starts. If the ZMQ bind fails (e.g., port 5557 is already in use), the engine logs a warning and continues without a drop copy publisher. This is a deliberate design choice: a port conflict should not prevent the matching engine from operating.

On shutdown the publisher is explicitly closed via `DropCopyPublisher.close()`, which releases the ZMQ socket cleanly.

11.9 Configuration reference

```
# engine_config.yaml — no explicit drop copy section needed
# The engine's bind address is controlled in src/edumatcher/config.py:
#   DROP_COPY_PUB_BIND_ADDR = "tcp://0.0.0.0:5557"  (via EDUMATCHER_ENGINE_BIND_HOST)
# Consumers connect via the separate DROP_COPY_PUB_ADDR constant
# (via EDUMATCHER_ENGINE_HOST), which must resolve to the same port.
```

To change the drop copy port, edit `DROP_COPY_PUB_BIND_ADDR` (and keep `DROP_COPY_PUB_ADDR` in sync for consumers) in `config.py`.

11.10 Subscribing to the drop copy feed

Quick observability: pm-dc-spy

For ad-hoc inspection you don't need to write a subscriber at all — `pm-dc-spy` is a read-only CLI that connects to port 5557 and prints every fill event, human-readable or as JSON:

```
pm-dc-spy --gateway TRADER01
```

Use the example subscriber below when you're building your own risk/clearing integration; use `pm-dc-spy` when you just want to look at the wire.

A minimal Python subscriber that prints every fill event:

```
import json
import zmq

ctx = zmq.Context()
sock = ctx.socket(zmq.SUB)
sock.connect("tcp://127.0.0.1:5557")

# Subscribe to all drop copy events (or filter by gateway ID)
sock.subscribe(b"drop_copy.event.")

print("Listening for drop-copy events on :5557 ...")
while True:
    frames = sock.recv_multipart()
    topic = frames[0].decode()
    payload = json.loads(frames[1])
    print(f"[{topic}] seq={payload['seq']}  {payload['symbol']}  "
          f"{payload['fill_qty']}@{payload['fill_price']}  "
          f"gateway={payload['gateway_id']}")
```

Example output:

```
Listening for drop-copy events on :5557 ...
[drop_copy.event.TRADER01] seq=1  AAPL  100@150.05  gateway=TRADER01
[drop_copy.event.TRADER02] seq=1  AAPL  100@150.05  gateway=TRADER02
[drop_copy.event.TRADER01] seq=2  MSFT  50@415.20   gateway=TRADER01
```

Tip

For production use, subscribe to a specific gateway prefix (e.g., `b"drop_copy.event.TRADER01"`) to limit traffic to only fills relevant to your risk system, and drop any message whose topic is not exactly `drop_copy.event.TRADER01`.

11.11 Alternative: relayed through your own ALF session

Connecting directly to port 5557 (above) is the most general option and the right choice for an external, independent recipient — a risk system, clearing broker, or compliance monitor that is not itself an ALF trading client. But if you already have an ALF session open (`pm-alf-console` or a `pm-alf-gwy` TCP client) and only want to see **your own** fills without managing a second connection, both ALF entry points can relay the feed for you:

```
# pm-alf-console: enable from the start
pm-alf-console --id TRADER01 --drop-copy

# or toggle at runtime from either pm-alf-console or an external
# pm-alf-gwy client
DC|STATE=ON
```

Once enabled, every fill for that gateway arrives asynchronously as a `DC_FILL` line — see ALF Console → DC and ALF TCP Gateway → DC for the full command reference. Internally this is exactly the same ZMQ SUB connection to `drop_copy.event.{gateway_id}` described above — `pm-alf-gwy` and `pm-alf-console` just do the subscribing for you and re-deliver it down the connection you already have, scoped to your own gateway ID only. This mirrors how a real exchange's drop copy is typically wired: relayed to a participant's own session infrastructure rather than requiring a wholly separate connection — see How real exchanges publish drop copy below for the comparison.

Both paths read the same feed and can be used simultaneously; enabling `DC` on your ALF session does not stop you from also connecting a separate `pm-dc-spy`/direct subscriber to port 5557.

11.12 Alternative: `pm-dc-gwy` for non-ZeroMQ external clients

Both options above assume the recipient either has `pyzmq` available (direct port-5557 subscriber) or is already an ALF trading client (`DC` relay). If neither is true — an external risk system, clearing broker, or compliance monitor written in a language or environment where installing ZeroMQ isn't practical — `pm-dc-gwy` exposes the same feed over a plain TCP text protocol (DC1). A client sends `HELLO|...|ID=<gateway_id>` and receives that gateway's fills as unsolicited `DC_FILL` lines, with no ZeroMQ dependency at all. See Drop-Copy TCP Gateway for the full protocol reference and setup instructions.

11.13 How real exchanges publish drop copy

Real venues typically deliver drop copy over a **dedicated FIX session per recipient**, separate from the trading session — often a second `SenderCompID`/`TargetCompID` pair, sometimes hosted by the exchange, sometimes by a clearing broker relaying fills downstream. The payload is an Execution Report (`MsgType=8`), the same message type used for live order acks/fills, delivered again on the drop-copy session. Sequence numbers are session-scoped FIX `MsgSeqNum`, with gap recovery via FIX's native `ResendRequest`/`SequenceReset` rather than a bespoke replay protocol. Entitlement is structural — the FIX session itself is provisioned per participant — rather than a filter applied after connecting.

EduMatcher's drop-copy relay through `pm-alf-gwy`/`pm-alf-console` (`DC`, above) is the closer simulation of this pattern: a per-participant, per-connection relay scoped to that participant's own `gateway_id`, layered on top of an existing session rather than requiring a wholly separate connection. The direct port-5557 connection is architecturally closer to tapping the wire between the exchange and its drop-copy relay — useful for building your own relay or observability tooling, but not how a real end-participant would typically receive their drop copy.

Still no authentication or entitlement checks

Unlike a real FIX drop-copy session (provisioned per participant at the connection level), neither the raw port-5557 socket nor the `DC` command in `pm-alf-gwy`/`pm-alf-console` enforce that a gateway can only see its own fills through some external check — `DC` scopes what a session *asks for* via `gateway_id`, but the underlying feed itself still has no authentication (see Architecture above). This is a deliberate scope decision for EduMatcher, not an oversight.

11.14 RALF DROP_COPY channel vs. this feed

The RALF gateway (`pm-ralf-gwy`) has its own concept called the `DROP_COPY` **channel** — see the RALF protocol reference. Despite the shared name, it is a different mechanism from the feed described on this page:

- The engine's drop copy feed (this page) is published by `DropCopyPublisher` on its own ZMQ PUB socket, port **5557**, with sequence numbers and a replay buffer scoped to fills only.
- RALF's `DROP_COPY` channel is one of three role-based subscription channels (`CLEARING`, `DROP_COPY`, `AUDIT`) on the RALF text protocol. Each trade is emitted **once per channel** as an `EXEC` message, sourced from the same `trade.executed` broadcast on port 5556 that feeds `CLEARING` and `AUDIT` — it is not sourced from port 5557, and the three channels carry identical trade content, differing only in which entitlement role may subscribe to them.

If you need a per-participant, sequenced, replayable fill feed for programmatic consumption, use the engine's drop copy feed on port 5557. If you need RALF's text-protocol trade feed with role-based entitlement checks, see RALF protocol reference and Post-trade.

11.15 See also

In this book:

- Drop-Copy Spy (pm-dc-spy) — read-only CLI for inspecting this feed without writing a subscriber
- Messages — all message types used in EduMatcher
- RALF protocol reference — RALF `DROP_COPY` channel wire format

Participant Guide:

- ALF Console → DC — relay this feed through your own `pm-alf-console` session

Operator's Guide:

- Drop-Copy TCP Gateway (pm-dc-gwy) — plain-TCP relay of this feed for non-ZeroMQ external clients
- ALF TCP Gateway → DC — relay this feed through your own `pm-alf-gwy` session
- Risk Controls — halt, collar, and circuit breaker events
- Persistence — durable audit trail as a complement to drop copy
- Post-trade — RALF post-trade feed and channels

Reference Manual:

- Processes — full ZeroMQ topology

CHAPTER 12

Protocol Support Library Examples

Learning objectives

After reading this chapter you will understand:

- where protocol support-library source code lives
- the Python and C APIs exposed by each parser library
- how to run the example clients for each gateway
- how to validate gateway connectivity quickly using reusable example code
- how to navigate parser-library source files when integrating external clients

12.1 Scope

This chapter covers source-code examples for all external protocol gateways and the REST API:

Protocol	Gateway	Port	Directory
ALF	`pm-alf-gwy`	5565	`docs/examples/alf`
CALF	`pm-md-gwy`	5570	`docs/examples/calf`
RALF	`pm-ralf-gwy`	5580	`docs/examples/ralf`
DC1	`pm-dc-gwy`	5590	`docs/examples/dc`
BALF	`pm-balf-gwy`	5560	`docs/examples/balf`
REST	`pm-api-gwy`	8080	`docs/examples/REST`

This chapter does not cover the ready-to-use engine configuration templates under `docs/examples/ref_data` — see Example Engine Configs for those.

See also: Protocol Overview

12.2 ALF support library and examples

ALF (`pm-alf-gwy`) is the primary order-entry gateway. External clients connect over TCP and exchange text lines using the ALF wire format.

12.2.1 Source files

File	Description
alf/README.txt	Directory overview and quick-start
alf/python/alf_parser.py	Python parser, builder, and session helper
alf/python/alf_client.py	Interactive Python client with tab-completion
alf/c/alf_parser.h	C library header
alf/c/alf_parser.c	C library implementation
alf/c/alf_client.c	Interactive C client with readline
alf/c/Makefile	Build helper

12.2.2 Wire format

```
VERB|KEY=VALUE|KEY=VALUE\n
```

Command verbs and field keys are case-insensitive (normalized to uppercase). Segments without `=` are silently skipped. Duplicate keys: last value wins. Response values preserve case: keep the complete `ORDER_ID` unchanged when sending `AMEND` or `CANCEL`. Prices on this TCP connection are decimal display prices, not the engine's internal integer ticks.

12.2.3 Python API — `alf_parser.py`

`parse_alf_line(line: str) → AlfMessage` Parse one ALF line. Returns an `AlfMessage(msg_type, fields)` frozen dataclass. Raises `AlfParseError` on malformed input.

`build_alf_line(msg_type: str, fields: dict | None) → str` Build one `\n`-terminated ALF line. Raises `AlfParseError` if the message type is invalid, a key is empty or contains `|`, `=`, CR or LF, or a value contains `|`. CR/LF in values are replaced with spaces.

`AlfSession.connect(host, port, gateway_id, client_name, timeout) → AlfSession` Open a TCP connection and complete the `HELLO`/`WELCOME` handshake. Validates `WELCOME`, bounds the handshake with `timeout` (default five seconds), and closes the socket on failure. After success reads block. It does not start a receive loop, keepalives, or automatic reconnects.

`AlfSession.send(msg_type, fields)` Build and send one ALF line over the established connection.

`AlfSession.recv_msg() → AlfMessage` Block until one complete `\n`-terminated line arrives and parse it. Buffers fragmented and coalesced TCP data. Use one reader per session.

`AlfSession.send_raw(line)` Send one line without a trailing newline; embedded CR/LF are rejected. Both send methods serialize socket writes with a lock.

`AlfSession.close()` Send `EXIT` and close the socket.

12.2.4 Python usage pattern

```
from alf_parser import parse_alf_line, build_alf_line, AlfSession, AlfMessage

# Parse one line received from the gateway
msg: AlfMessage = parse_alf_line("ACK|ORDER_ID=abc|ACCEPTED=TRUE|SYMBOL=AAPL|TAG=ORDER-001")
print(msg.msg_type)   # "ACK"
print(msg.fields)     # {"ORDER_ID": "abc", "ACCEPTED": "TRUE", "SYMBOL": "AAPL", "TAG": "ORDER-001"}

# Build a line to send
line: str = build_alf_line("NEW", {
    "SYM": "AAPL", "SIDE": "BUY", "TYPE": "LIMIT",
    "QTY": "100", "PRICE": "150.00", "TAG": "ORDER-001",
})
# → "NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TAG=ORDER-001\n"
```

For a complete query session, including interleaved events and keepalives, use the application example below.

12.2.5 Run Python interactive client

```
cd docs/examples/alf/python

# Connect to local gateway as TRADER01
python3 alf_client.py --id TRADER01

# Connect to remote gateway with a custom client name in logs
python3 alf_client.py --host 10.0.0.5 --port 5565 --id MM01 --client "my-mm-bot"
```

At the prompt, tab-completion is available for commands, field names, and enum values. Command history is persisted to `~/.alf_client_history`.

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TAG=ORDER-001
[TRADER01]> AMEND|ID=<order-id>|PRICE=151.00|RTAG=req-001
[TRADER01]> CANCEL|ID=<order-id>|RTAG=req-002
[TRADER01]> KILL
[TRADER01]> SYMBOLS
[TRADER01]> ORDERS
[TRADER01]> POS
[TRADER01]> SESSION
[TRADER01]> STATUS
[TRADER01]> HELP
[TRADER01]> EXIT
```

`AMEND` and `CANCEL` need the order's full 32-character ID, exactly as the `ACK` line or the `ORDERS` table prints it.

12.2.6 C API — `alf_parser.h`

`int alf_parse_line(char *line, alf_message_t *out)` Parse one mutable ALF line in-place. Returns 0 on success, negative on error (`-1` null args, `-2` empty line, `-3` invalid verb, `-4` too many fields, `-6` verb/key/value exceeds storage).

`const char *alf_get_field(const alf_message_t *msg, const char *key)` Return the value for `key` (case-insensitive), or `NULL` if absent.

`int alf_build_line(char *out, size_t cap, const char *msg_type, const char * const *kv)` Write one ALF line into `out`. `kv` is a `NULL`-terminated array of alternating key/value strings. Returns bytes written, including the newline, or `-1` on invalid arguments or overflow. Rejects delimiter-

bearing keys/values and replaces CR/LF in values with spaces.

12.2.7 C usage pattern

```
#include "alf_parser.h"

/* Parse */
char line[] = "ACK|ORDER_ID=abc|ACCEPTED=TRUE";
alf_message_t msg;
if (alf_parse_line(line, &msg) != 0) {
    fprintf(stderr, "Invalid ALF response\n");
    return 1;
}
printf("%s\n", msg.msg_type);                    /* "ACK" */
const char *accepted = alf_get_field(&msg, "ACCEPTED");
if (accepted) printf("%s\n", accepted);           /* "TRUE" */

/* Build */
const char *kv[] = {
    "SYM", "AAPL", "SIDE", "BUY",
    "TYPE", "LIMIT", "QTY", "100", "PRICE", "150.00", "TAG", "ORDER-001", NULL,
};
char buf[4096];
int len = alf_build_line(buf, sizeof(buf), "NEW", kv);
if (len < 0) {
    fprintf(stderr, "Cannot build ALF command\n");
    return 1;
}
/* → "NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TAG=ORDER-001\n" */
```

This fragment belongs inside your application's function; include `<stdio.h>` for its diagnostics. Send all `len` bytes using a partial-write loop, not a single unchecked `write()`.

12.2.8 Build and run C client

```
cd docs/examples/alf/c
make

# Connect to local gateway
./alf_client --id TRADER01

# Connect to remote gateway; disable ANSI colours
./alf_client --host 10.0.0.5 --port 5565 --id TRADER01 --no-color
```

The C client supports the same commands and tab-completion as the Python client.

12.2.9 Use ALF as a foundation for your own program

The examples are standalone source, not an installed `edumatcher` SDK. Download or copy the files from the table above into your project:

- **Python:** copy `alf_parser.py` beside your application and import it. The parser/session module uses only the standard library and does not require readline. Python 3.9+ is sufficient for these examples; the installed engine has its own runtime requirements.
- **C:** copy `alf_parser.h` and `alf_parser.c`, include the header, and compile the implementation with your sources, for example `cc -std=c11 -Wall -Wextra my_bot.c alf_parser.c -o my_bot`. The parser has no readline dependency. Readline is needed only for the interactive client.

- Treat the interactive clients as lifecycle/event-dispatch references. Replace their terminal display and local caches with your application's handlers and state; do not make a headless bot depend on the REPL.

Configure the participant ID in the engine before connecting. The examples below assume `TRADER01` is authorized, the engine and gateway are running, and no other client is using that ID. `QUOTE` additionally requires the `MARKET_MAKER` role.

This runnable Python query illustrates one socket reader, periodic inbound activity, a bounded response wait, and matching multiline termination. Save it beside `alf_parser.py` and run `python3 my_bot.py`:

```
import time

from alf_parser import AlfSession

session = AlfSession.connect("127.0.0.1", 5565, "TRADER01", timeout=5.0)
try:
    interval = session.welcome.idle_timeout / 3
    next_ping = time.monotonic() + interval
    deadline = time.monotonic() + 10.0
    session.send("SYMBOLS")
    while True:
        now = time.monotonic()
        if now >= deadline:
            raise TimeoutError("SYMBOLS response did not complete")
        if now >= next_ping:
            session.send("PING")
            next_ping = now + interval
        session.sock.settimeout(min(next_ping - now, deadline - now))
        try:
            msg = session.recv_msg()
        except TimeoutError:
            continue
        print(msg.msg_type, msg.fields)
        if msg.msg_type == "ERR":
            raise RuntimeError(f"Gateway error: {msg.fields}")
        if msg.msg_type == "END" and msg.fields.get("TYPE") == "SYMBOLS":
            break
finally:
    session.close()
```

The timeout here is an application policy, not the gateway's idle timeout. The helper retains incomplete lines across receive timeouts. For a long-lived bot, continue receiving and dispatching after the initial query. The interactive Python client's `_recv_loop` and `_keepalive_loop` show a threaded alternative; its send lock prevents command/keepalive bytes from interleaving.

For a C bot, `alf_parser.c` only parses/builds messages: it does not connect, authenticate, buffer TCP, schedule keepalives, or reconnect. Start from `connect_gateway`, `do_handshake`, `gwy_send`, `process_socket_data`, and the `select()` loop in `alf_client.c`. These are internal example functions, not exported library APIs. Adapt them into your own transport/event loop:

1. Connect and send `HELLO`; validate `WELCOME` before sending commands.
2. Buffer bytes until newline; one read can contain part of a line or many lines. Retry interrupted I/O and handle partial writes and disconnects.
3. Check every parser/builder return code and nullable field lookup.
4. Send `PING` more frequently than `WELCOME|IDLE`, even while receiving heartbeats or broadcasts.

Receiving data does not reset gateway idle time.

5. Dispatch unsolicited events while collecting query rows. Finish only on the matching `END|TYPE=...`; serialize multiline queries because the example collectors track only one active query at a time.
6. On shutdown send `EXIT` and close the socket. On connection loss report the failure, reconnect explicitly, and reconcile engine state before resuming.

Keep the following application responsibilities explicit:

- Correlate order entry with `TAG` and amend/cancel requests with `RTAG`; handle both gateway `ERR` and engine rejection `ACK|ACCEPTED=FALSE`. Do not assume the next message after a send is its response or blindly replay an order whose outcome is unknown after disconnect.
- Preserve full order IDs. `FILL` includes `SYMBOL`/`SIDE` for orders not in your cache and may include `LIQUIDITY`/`TRADE_IDS`; stops emit `TRIGGERED`. Keep ALF decimal prices separate from internal tick representations.
- `DC_FILL` is a second report of an execution, not another position change. The demos display it without booking it again.
- Bare `POS` and its realized P&L are client-local, starting with fills seen by this connection. Use `ORDERS` and `POS|GW=TRADER01` to reconcile engine orders/positions after reconnect; the query returns `POSITION`, `POS_ENTRY`, then `END|TYPE=POSITION`. `STATUS` is also a local display, not a wire command.
- The C demo is bounded: 256 cached orders, 512 symbols/positions and query rows, 64 fields per message, 63-byte keys and 511-byte values. It reports capacity errors instead of silently truncating protocol data. Review these limits and add persistence/reconciliation appropriate to your workload.
- The examples use plain TCP, not TLS; use a trusted network or a protected transport when deploying outside a local development environment.

From a repository checkout, validate adaptations against the real gateway and engine workflows with:

```
poetry run pytest tests/test_alf_examples.py --no-cov
```

The suite includes compiled C behavior checks and interactive Python/C round trips through real engine handlers. C checks require a compiler, make, and GNU readline development files; they are skipped if unavailable.

12.2.10 Manual gateway sanity check

```
nc 127.0.0.1 5565
```

Then type:

```
HELLO|CLIENT=manual01|PROTO=ALF1|ID=TRADER01
NEW|SYM=AAPL|SIDE=BUY|TYPE=MARKET|QTY=10
SYMBOLS
ORDERS
```

See also:

- ALF Gateway
- ALF Protocol Reference

12.3 CALF support library and examples

CALF (`pm-md-gwy`) is the market data feed gateway. Subscribers connect over TCP and receive a stream of top-of-book, trade, and other market-data events.

The example subscribers are more than a single trivial subscription: they combine `TOP`, `TRADE`, `STATE` (including the session-wide `SYM=*` wildcard), and Level 2 `DEPTH` in one client, maintain a small top-of-book cache (incremental `MD` updates omit unchanged sides), pretty-print the `DEPTH` ladder, detect gaps in the per-`(CH,SYM) SEQ` counters, and check `WELCOME|CH_SUPPORTED=` before relying on channels that may not exist on an older gateway build.

12.3.1 Source files

File	Description
calf/README.txt	Directory overview and quick-start
calf/calf_parser.py	Python parser and builder
calf/calf_subscriber.py	Python subscriber example
calf/calf_parser.h	C library header
calf/calf_parser.c	C library implementation
calf/calf_subscriber.c	C subscriber example
calf/Makefile	Build helper

12.3.2 Wire format

```
MSGTYPE|KEY=VALUE|KEY=VALUE\n
```

Fields missing `=` raise `CalfParseError`. Duplicate keys: last value wins (same as ALF), the parser never raises for this case.

12.3.3 Python API — `calf_parser.py`

`parse_calf_line(line: str) → CalfMessage` Parse one CALF line. Returns a `CalfMessage(msg_type, fields)` frozen dataclass. Raises `CalfParseError` on malformed input.

`build_calf_line(msg_type: str, fields: dict | None) → str` Build one `\n`-terminated CALF line. Raises `CalfParseError` for invalid message types or fields containing `|`.

12.3.4 Python usage pattern

```
from calf_parser import build_calf_line, parse_calf_line

# Build a HELLO line
line = build_calf_line("HELLO", {"CLIENT": "demo01", "PROTO": "CALF1"})
# → "HELLO|CLIENT=demo01|PROTO=CALF1\n"

# Parse a WELCOME response
msg = parse_calf_line("WELCOME|PROTO=CALF1|GW=md-gwy01|HBINT=1|REPLAY=30")
print(msg.msg_type)          # "WELCOME"
print(msg.fields["HBINT"])   # "1"

# Parse a streaming TOP-OF-BOOK update
top = parse_calf_line("TOP|SYM=AAPL|BID=149.50|ASK=150.00|SEQ=1042")
print(top.fields["BID"])     # "149.50"
```

12.3.5 Run Python example

```
cd docs/examples/calf

# Subscribe to TOP/TRADE/STATE/DEPTH for AAPL and MSFT
python3 calf_subscriber.py --host 127.0.0.1 --port 5570 \
    --symbols AAPL,MSFT

# Also subscribe to an index feed (skipped if the gateway doesn't advertise
# INDEX support in WELCOME|CH_SUPPORTED=)
python3 calf_subscriber.py --host 127.0.0.1 --port 5570 \
    --symbols AAPL --index EDU100

# Resume from a known sequence number (single-stream resume)
python3 calf_subscriber.py --host 127.0.0.1 --port 5570 \
    --resume --resume-ch TOP --resume-sym AAPL --lastseq 1042
```

Channels are not selected with a flag — the client always requests every channel the connected gateway advertises via `WELCOME|CH_SUPPORTED=` (`TOP`, `TRADE`, `STATE`, `DEPTH`), plus a separate session-wide `SUB|CH=STATE|SYM=*` subscription and, if `--index` is given, `SUB|CH=INDEX`. Run `python3 calf_subscriber.py --help` for the full flag list, including `--client` and `--no-state-wildcard` (skip the session-wide `STATE` subscription).

12.3.6 C API — `calf_parser.h`

`int calf_parse_line(char *line, calf_message_t *out)` Parse one mutable CALF line in-place. Returns 0 on success, negative on error.

`const char *calf_get_field(const calf_message_t *msg, const char *key)` Return the value for `key`, or `NULL` if absent.

12.3.7 C usage pattern

```
#include "calf_parser.h"

char line[] = "TOP|SYM=AAPL|BID=149.50|ASK=150.00|SEQ=1042";
calf_message_t msg;
calf_parse_line(line, &msg);
printf("%s\n", calf_get_field(&msg, "BID"));  /* "149.50" */
```

12.3.8 Build and run C example

```
cd docs/examples/calf
make
./calf_subscriber 127.0.0.1 5570 AAPL,MSFT EDU100
```

Arguments are positional: `host [port [symbols [index_id]]]`. `symbols` is a comma-separated list (default `AAPL`); `index_id` is optional (default: skip the `INDEX` subscription).

12.3.9 Manual gateway sanity check

```
nc 127.0.0.1 5570
```

Then type:

```
HELLO|CLIENT=manual01|PROTO=CALF1
SUB|CH=TOP,TRADE|SYM=AAPL
```

See also:

- Market Data Feed (CALF)
- CALF Protocol Reference

12.4 RALF support library and examples

RALF (`pm-ralf-gwy`) is the post-trade dissemination gateway. Clients connect over TCP and receive execution reports, drop-copy events, and audit records.

The example subscribers subscribe to every channel a role is actually entitled to (`CLEARING`/`DROP_COPY` → that channel only, `AUDIT` → all three) in **one** combined `SUB`, then put the received `EXEC`/`EOD` data to use: a running per-symbol executed-volume tally that de-duplicates `EXEC` lines by `EXEC_ID` (a single executed trade is delivered once per subscribed channel, so an `AUDIT` client receives the same trade up to three times and naively counting raw lines would overstate volume). Both examples also detect gaps in the per-channel `SEQ` counters, cross-check the tally against each `EOD`'s `TRADE_COUNT`, and print a final tally on Ctrl-C shutdown.

12.4.1 Source files

File	Description
ralf/README.txt	Directory overview and quick-start
ralf/ralf_parser.py	Python parser and builder
ralf/ralf_subscriber.py	Python subscriber example
ralf/ralf_parser.h	C library header
ralf/ralf_parser.c	C library implementation
ralf/ralf_subscriber.c	C subscriber example
ralf/Makefile	Build helper

12.4.2 Wire format

```
MSGTYPE|KEY=VALUE|KEY=VALUE\n
```

Identical text-line structure to CALF; field parsing rules are the same.

12.4.3 Python API — `ralf_parser.py`

`parse_ralf_line(line: str) → RalfMessage` Parse one RALF line. Returns a `RalfMessage(msg_type, fields)` frozen dataclass. Raises `RalfParseError` on malformed input.

`build_ralf_line(msg_type: str, fields: dict | None) → str` Build one `\n`-terminated RALF line. Raises `RalfParseError` for invalid message types or fields containing `|`.

12.4.4 Python usage pattern

```
from ralf_parser import build_ralf_line, parse_ralf_line

# Build a HELLO line
line = build_ralf_line("HELLO", {
    "CLIENT": "demo01",
    "PROTO": "RALF1",
    "ROLE": "CLEARING",
    "LASTSEQ": "0",
})
# → "HELLO|CLIENT=demo01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=0\n"

# Parse a WELCOME response
msg = parse_ralf_line("WELCOME|PROTO=RALF1|GW=ralf-gwy01|ROLE=CLEARING")
print(msg.msg_type)         # "WELCOME"
print(msg.fields["ROLE"])   # "CLEARING"

# Parse an inbound execution report
fill = parse_ralf_line("FILL|SYM=AAPL|SIDE=BUY|QTY=100|PRICE=150.00|SEQ=77")
print(fill.fields["PRICE"]) # "150.00"
```

12.4.5 Run Python example

```
cd docs/examples/ralf

# Subscribe as CLEARING role — --role defaults the subscribed channel(s)
# to exactly what that role may access (here: just the CLEARING channel)
python3 ralf_subscriber.py --host 127.0.0.1 --port 5580 --role CLEARING

# Resume from a known sequence number
python3 ralf_subscriber.py --host 127.0.0.1 --port 5580 \
    --role CLEARING --lastseq 77
```

Pass `--channels` explicitly to override a role's default channel set — this is also how to trigger `ERR|CODE=ENTITLEMENT_DENIED` on purpose, by requesting a channel outside the role's entitlement:

```
# CLEARING is only entitled to the CLEARING channel, so requesting AUDIT
# gets rejected
python3 ralf_subscriber.py --role CLEARING --channels AUDIT
```

Run `python3 ralf_subscriber.py --help` for the full flag list, including `--client` and `--symbols` (defaults to `*`).

12.4.6 C API — `ralf_parser.h`

`int ralf_parse_line(char *line, ralf_message_t *out)` Parse one mutable RALF line in-place. Returns 0 on success, negative on error.

`const char *ralf_get_field(const ralf_message_t *msg, const char *key)` Return the value for `key`, or `NULL` if absent.

12.4.7 C usage pattern

```
#include "ralf_parser.h"

char line[] = "FILL|SYM=AAPL|SIDE=BUY|QTY=100|PRICE=150.00|SEQ=77";
ralf_message_t msg;
ralf_parse_line(line, &msg);
printf("%s\n", ralf_get_field(&msg, "PRICE")); /* "150.00" */
```

12.4.8 Build and run C example

```
cd docs/examples/ralf
make
./ralf_subscriber 127.0.0.1 5580 AUDIT
```

Arguments are positional: `host [port [role]]` (`role` one of `CLEARING`, `DROP_COPY`, `AUDIT`; default `CLEARING`).

12.4.9 Manual gateway sanity check

```
nc 127.0.0.1 5580
```

Then type:

```
HELLO|CLIENT=manual01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=0
SUB|CH=CLEARING|SYM=*
```

See also:

- Post-Trade Dissemination (RALF)
- RALF Protocol Reference

12.5 DC support library and examples

DC1 (`pm-dc-gwy`) relays the engine's drop-copy feed to plain TCP clients. A client names one participant's gateway ID in `HELLO` and receives that participant's fills as unsolicited `DC_FILL` lines; there is no `SUB` step.

The example subscribers keep a running per-symbol tally of fills, executed quantity, VWAP and the `MAKER`/`TAKER` split (`DC_FILL` carries no side, so it is an activity tally, not a net position), and print it on Ctrl-C shutdown. Drop-copy `SEQ` is a separate counter for each gateway ID, so a per-ID session sees `1, 2, 3, …` with no gaps. The examples report a `SEQ` that skips ahead (lost fills) and one that goes backwards (the engine restarted and the counter began again at 1).

12.5.1 Source files

File	Description
dc/README.txt	Directory overview and quick-start
dc/dc_parser.py	Python parser and builder
dc/dc_subscriber.py	Python subscriber example
dc/dc_parser.h	C library header
dc/dc_parser.c	C library implementation

File	Description
dc/dc_subscriber.c	C subscriber example
dc/Makefile	Build helper

12.5.2 Wire format

```
MSGTYPE|KEY=VALUE|KEY=VALUE\n
```

Identical text-line structure to CALF and RALF; field parsing rules are the same.

12.5.3 Python API — `dc_parser.py`

`parse_dc_line(line: str) → DcMessage` Parse one DC1 line. Returns a `DcMessage(msg_type, fields)` frozen dataclass. Raises `DcParseError` on malformed input.

`build_dc_line(msg_type: str, fields: dict | None) → str` Build one `\n`-terminated DC1 line. Raises `DcParseError` for invalid message types or fields containing `|`.

12.5.4 Python usage pattern

```
from dc_parser import build_dc_line, parse_dc_line

# Build a HELLO line
line = build_dc_line("HELLO", {"CLIENT": "risk01", "PROTO": "DC1", "ID": "TRADER01"})
# → "HELLO|CLIENT=risk01|PROTO=DC1|ID=TRADER01\n"

# Parse an inbound fill
fill = parse_dc_line(
    "DC_FILL|SEQ=42|ORDER_ID=ORD-001|SYMBOL=AAPL|FILL_QTY=100|FILL_PRICE=150.05|LIQUIDITY=TAKER"
)
print(fill.fields["FILL_PRICE"])  # "150.05"
```

12.5.5 Run Python example

```
cd docs/examples/dc
python3 dc_subscriber.py --host 127.0.0.1 --port 5590 --id TRADER01
```

Run `python3 dc_subscriber.py --help` for the full flag list, including `--client`.

12.5.6 C API — `dc_parser.h`

`int dc_parse_line(char *line, dc_message_t *out)` Parse one mutable DC1 line in-place. Returns 0 on success, negative on error.

`const char *dc_get_field(const dc_message_t *msg, const char *key)` Return the value for `key`, or `NULL` if absent.

12.5.7 C usage pattern

```
#include "dc_parser.h"

char line[] = "DC_FILL|SEQ=42|SYMBOL=AAPL|FILL_QTY=100|FILL_PRICE=150.05";
dc_message_t msg;
dc_parse_line(line, &msg);
printf("%s\n", dc_get_field(&msg, "FILL_PRICE")); /* "150.05" */
```

12.5.8 Build and run C example

```
cd docs/examples/dc
make
./dc_subscriber 127.0.0.1 5590 TRADER01
```

Arguments are positional: `host [port [gateway_id]]` (default `127.0.0.1 5590 TRADER01`).

12.5.9 Manual gateway sanity check

```
nc 127.0.0.1 5590
```

Then type:

```
HELLO|CLIENT=manual01|PROTO=DC1|ID=TRADER01
```

See also:

- Drop-Copy TCP Gateway
- Drop Copy

12.6 BALF support library and examples

BALF (`pm-balf-gwy`) is the binary order-entry gateway, intended for ultra-low-latency clients. Frames are fixed-length little-endian structs rather than text lines.

12.6.1 Source files

File	Description
balf/balf_parser.py	Python binary parser
balf/balf_parser.c	C binary parser

12.6.2 Binary frame structure

Every BALF frame begins with an 8-byte header:

Offset	Size	Field
0	1	Magic (`0xBA`)
1	1	Version (`0x01`)
2	1	Message type
3	1	Flags
4	4	Sequence number (LE uint32)

Prices are encoded as fixed-point integers scaled by `100_000_000` (8 decimal places).

12.6.3 Supported message types

Constant	Value	Description
`MSG_LOGON`	`0x01`	Session logon
`MSG_LOGON_ACK`	`0x02`	Logon acknowledgement
`MSG_NEW_ORDER`	`0x10`	New order
`MSG_ORDER_ACK`	`0x11`	Order acknowledgement
`MSG_CANCEL_ORDER`	`0x12`	Cancel request
`MSG_CANCEL_ACK`	`0x13`	Cancel acknowledgement
`MSG_AMEND_ORDER`	`0x14`	Amend request
`MSG_AMEND_ACK`	`0x15`	Amend acknowledgement
`MSG_EXECUTION_REPORT`	`0x20`	Fill / execution report
`MSG_HEARTBEAT`	`0x30`	Heartbeat
`MSG_HEARTBEAT_ACK`	`0x31`	Heartbeat acknowledgement
`MSG_LOGOUT`	`0x40`	Session logout

12.6.4 Python API — `balf_parser.py`

`parse_header(frame: bytes) → Header` Parse the 8-byte BALF header. Returns a `Header(magic, version, msg_type, flags, seq_no)` frozen dataclass. Raises `ValueError` on bad magic or unsupported version.

`split_frame(frame: bytes) → tuple[Header, bytes]` Validate frame length against the expected size for the message type, then return `(header, body_bytes)`. Raises `ValueError` for unknown message types or wrong frame sizes.

`decode_price(raw: int) → float` Convert a raw fixed-point integer to a float price value.

`parse_logon_ack(body: bytes) → dict` Decode a `LOGON_ACK` body. Returns a dict with keys `gateway_id`, `accepted`, `reject_code`, `message`.

`describe_logon_ack(ack: dict) → str` One line saying whether the logon succeeded and, if not, what to do. A rejected logon is always followed by the gateway closing the connection, so "retry" means a new TCP connection and a new `LOGON`.

`parse_order_ack(body: bytes) → dict` Decode an `ORDER_ACK` body. Returns a dict with keys `client_order_id`, `order_id`, `timestamp_ns`, `accepted`, `reject_code`, `reject_name`, `reason`.

`parse_cancel_ack(body: bytes) → dict` Decode a `CANCEL_ACK` body: `client_order_id`, `order_id`, `accepted`, `cancel_reason` (0 = you asked, 255 = the exchange decided) and `by_exchange`.

`parse_amend_ack(body: bytes) → dict` Decode an `AMEND_ACK` body: `client_order_id`, `order_id`, `new_price`, `new_quantity`, `remaining_qty`, `accepted`, `priority_reset`.

`parse_stop_triggered(body: bytes) → dict` Decode a `STOP_TRIGGERED` body: a stop, stop-limit or trailing stop has fired and its execution reports follow. Keys `client_order_id`, `order_id`, `stop_price`, `trigger_price`, `symbol`, `order_type`, `becomes`.

`parse_execution_report(body: bytes) → dict` Decode an `EXECUTION_REPORT` body. Returns a dict with keys `client_order_id`, `order_id`, `fill_price`, `fill_qty`, `remaining_qty`, `timestamp_ns`, `symbol`, `side` (`BUY`/`SELL`) and `status` (`PARTIAL`/`FILLED`). `fill_price` is already converted to a float via `decode_price()` — no further scaling is needed.

12.6.5 Python usage pattern

```
from balf_parser import (
    parse_header, split_frame, decode_price,
    parse_logon_ack, parse_order_ack, parse_execution_report,
    MSG_LOGON_ACK, MSG_ORDER_ACK, MSG_EXECUTION_REPORT,
    FRAME_SIZES,
)

# Decode a raw frame received from TCP
hdr, body = split_frame(raw_bytes)

if hdr.msg_type == MSG_EXECUTION_REPORT:
    report = parse_execution_report(body)
    price = report["fill_price"]  # already decoded to a float
    print(f"fill: {report['symbol']} qty={report['fill_qty']} price={price}")
elif hdr.msg_type == MSG_ORDER_ACK:
    ack = parse_order_ack(body)
    print(f"order_id={ack['order_id']} accepted={ack['accepted']}")
```

12.6.6 C reference decoder — `balf_parser.c`

Unlike the ALF and CALF C libraries, `balf_parser.c` has no header file and its functions are all `static` — it is a single self-contained reference program, not a linkable library. It builds a synthetic `LOGON_ACK` frame in memory, decodes it with the same `split_frame`/`parse_header` logic a real client would use, and prints the result as a self-test:

```
cd docs/examples/balf
cc -std=c11 -Wall -Wextra -pedantic -O2 -I../generated \
   balf_parser.c ../generated/edumatcher_order.c \
   ../generated/edumatcher_msg.c -o balf_parser
./balf_parser
# → LOGON_ACK gateway_id=TRADER01 accepted=1 reject_code=0 msg=ok
#   EXECUTION_REPORT ...
#   LOGON_ACK gateway_id=TRADER01 accepted=0 reject_code=255 msg=SYMBOLS_UNAVAILABLE
#   CANCEL_ACK ... / AMEND_ACK ... / STOP_TRIGGERED ...
#   balf_parser.c self-test: OK
```

`EXECUTION_REPORT` and `STOP_TRIGGERED` are decoded by the generated binding in `docs/examples/generated` (hence the extra sources); `LOGON_ACK`, `CANCEL_ACK` and `AMEND_ACK` are decoded by hand.

There is no BALF C (or Python) network client example — only the parsing logic is provided. A client written on it must handle three things beyond the happy path:

- **A held or refused `LOGON_ACK`.** The gateway withholds the accepted ack until the engine has sent the symbol list, so it can take a moment. If the engine never does, the ack is `accepted=0`, `reject_code=0xFF`, `msg=SYMBOLS_UNAVAILABLE`, and the connection closes: reconnect and log on again. `describe_logon_ack()` reports this.
- **A `LOGOUT` from the gateway.** The engine restarted and no longer holds the session; the connection closes. Log on again from scratch.
- **An `ORDER_ACK` with `reject_code 0x03`** for a symbol the engine does not list. To exercise BALF end-to-end, connect a TCP client to `pm-balf-gwy` and feed received frames through `split_frame()` (Python: `balf_parser.py`) or the C reference decoder's parsing functions.

12.6.7 Run the gateway

```
pm-balf-gwy

# Override bind port
pm-balf-gwy --port 5560

# Override engine host
pm-balf-gwy --engine-host 10.0.0.5
```

The gateway binds on port **5560** by default. BALF uses fixed-length binary frames so there is no text-mode manual sanity check; use the Python parser library or a binary client tool to verify connectivity.

See also:

- BALF Gateway
- BALF Protocol Reference

12.7 REST API support library and examples

`pm-api-gwy` exposes a JSON REST API for order submission and session queries. The support library (`api_gateway_client`) uses only the Python standard library and the C standard socket API — no third-party HTTP libraries are required. The two WebSocket examples need the `websockets` package (`pip install websockets`).

Every Python example runs against a real exchange in `make test-e2e` (`tests/test_examples_e2e.py`), so they are kept working.

12.7.1 Source files

File	Description
REST/python/api_gateway_client.py	Python REST client library
REST/python/demo_info.py	Print gateway status, symbols, and session info
REST/python/submit_market_order.py	Submit a MARKET order from the command line
REST/python/order_lifecycle.py	LIMIT with `wait=ack` and an `Idempotency-Key`, a safe retry, amend, replace, cancel, and a `TICK_VIOLATION` refusal
REST/python/book_snapshot.py	The order book over REST: top of book for every symbol, one symbol's ladder, `*_ticks`, `seq`, `UNKNOWN_SYMBOL`
REST/python/history_pages.py	Read a whole history list by following `next_cursor`
REST/python/private_stream.py	The private WebSocket: snapshot, `stream_seq` gap detection, reconnect recovery
REST/python/market_data.py	Market data: `items` subscribe, per-topic `seq`, trades `resume`
REST/python/admin_halt_reopen.py	Admin: halt a symbol, list its orders, dry-run reopen, reopen or resume
REST/c/api_gateway_client.h	C library header
REST/c/api_gateway_client.c	C library implementation

File	Description
REST/c/demo_info.c	C equivalent of demo_info.py
REST/c/Makefile	Build helper

12.7.2 Authentication

All requests require a `Bearer` token in the `Authorization` header. The default development key is `key-trader-demo`.

Configure via environment variables:

```
export EDUMATCHER_API_URL="http://127.0.0.1:8080"
export EDUMATCHER_API_KEY="key-trader-demo"
```

12.7.3 Python API — `api_gateway_client.py`

`ApiGatewayClient(base_url, api_key, timeout=5.0)` Construct a client for the given base URL and API key.

`client.get_json(path: str, params: dict | None = None) → dict` Issue a `GET` request (with `params` as the query string) and return the parsed JSON body.

`client.post_json(path: str, payload: dict | None = None, idempotency_key: str | None = None) → dict` Issue a `POST` request with a JSON body and return the parsed JSON response. With `idempotency_key`, sends it as the `Idempotency-Key` header: reuse the same key to retry a write safely (the gateway replays the first reply).

`client.patch_json(path, payload)` / `client.delete_json(path)` Amend and cancel.

`client.pages(path: str, key: str, params: dict | None = None) → Iterator[dict]` Every item of a paginated list (`key` names it: `trades`, `events`, …), following `next_cursor` until `has_more` is false.

Errors are two exception types, because a client must react differently:

- **`ApiError`**: the gateway answered with an error envelope; `status`, `code`, `reject_code`, `field` and `message` come from it. Nothing reached the engine, so the request may be fixed and sent again.
- **`ApiUnreachable`**: no answer at all. A write may or may not have been sent; retry it only with the same `Idempotency-Key`.

Both subclass `RuntimeError`.

12.7.4 Python usage pattern

```
import uuid

from api_gateway_client import ApiGatewayClient

client = ApiGatewayClient("http://127.0.0.1:8080", "key-trader-demo")

# Query endpoints
status  = client.get_json("/api/v1/status")
symbols = client.get_json("/api/v1/symbols")
session = client.get_json("/api/v1/session")

# Submit a MARKET order
result = client.post_json("/api/v1/orders", {
    "symbol":     "AAPL",
    "side":       "BUY",
    "order_type": "MARKET",
    "quantity":   100,
})
print(result["order_id"], result["status"])

# Submit and wait for the engine ACK; the key makes a retry safe
result = client.post_json("/api/v1/orders?wait=ack", {
    "symbol":     "MSFT",
    "side":       "SELL",
    "order_type": "MARKET",
    "quantity":   50,
}, idempotency_key=str(uuid.uuid4()))
if result["ack_timeout"]:
    print("sent; the ACK was late -- follow it, do not resubmit")
else:
    print(result.get("event"))   # engine acknowledgement event
```

12.7.5 Run Python examples

```
cd docs/examples/REST/python

# Print gateway status, symbols, and session information
python3 demo_info.py

# Submit a BUY MARKET order for 100 AAPL
python3 submit_market_order.py --side BUY --symbol AAPL --qty 100

# Submit and block until the engine ACKs
python3 submit_market_order.py --side SELL --symbol MSFT --qty 50 --wait-ack

# Override URL or key on the command line
python3 submit_market_order.py --url http://10.0.0.5:8080 --key mykey \
    --side BUY --symbol TSLA --qty 10

# One order's whole life; --price below the bid so it rests
python3 order_lifecycle.py --symbol AAPL --price 22.70

# The book over REST: every symbol's top of book, then AAPL's ladder
python3 book_snapshot.py --symbol AAPL --levels 5

# Today's trades, page by page
python3 history_pages.py --limit 50

# The private stream: place and cancel one order, watch both events arrive
python3 private_stream.py --symbol AAPL --price 22.70

# Market data: book for every symbol, trades for one; --trade makes a print
python3 market_data.py --symbol AAPL --trade

# Admin (an ADMIN key): halt, inspect, dry-run reopen, reopen
python3 admin_halt_reopen.py --symbol MSFT --key <admin key>
```

12.7.6 C API — `api_gateway_client.h`

`ApiGatewayClient api_gateway_client(host, port, api_key)` Construct a client struct (stack-allocated).

`char *api_gateway_get(const ApiGatewayClient *client, const char *path)` Issue a `GET` request and return the raw HTTP response body as a heap-allocated string. The caller must `free()` the result. Returns `NULL` on network or allocation failure.

12.7.7 C usage pattern

```
#include "api_gateway_client.h"
#include <stdlib.h>
#include <stdio.h>

const char *key = getenv("EDUMATCHER_API_KEY") ?: "key-trader-demo";
ApiGatewayClient client = api_gateway_client("127.0.0.1", 8080, key);

char *body = api_gateway_get(&client, "/api/v1/symbols");
if (body) {
    printf("%s\n", body);
    free(body);
}
```

12.7.8 Build and run C example

```
cd docs/examples/REST/c
make

# Print status, symbols, and session info
./demo_info
```

12.7.9 Manual API sanity check

```
curl -s -H "Authorization: Bearer key-trader-demo" \
     http://127.0.0.1:8080/api/v1/status | python3 -m json.tool

curl -s -X POST \
     -H "Authorization: Bearer key-trader-demo" \
     -H "Content-Type: application/json" \
     -d '{"symbol":"AAPL","side":"BUY","order_type":"MARKET","quantity":10}' \
     http://127.0.0.1:8080/api/v1/orders | python3 -m json.tool
```

See also:

- API Gateway

12.8 Integration guidance

Use the parser libraries as the protocol boundary in your client code:

1. Read bytes from TCP (or HTTP for REST).
2. Split by newline into protocol lines (text protocols) or read fixed-size frames (BALF).
3. Parse lines or frames using the appropriate parser library.
4. Route by `msg_type` and fields.
5. Track `SEQ` checkpoints per protocol rules to enable gap detection and replay on reconnect.

This keeps socket handling separate from protocol semantics and makes unit testing straightforward — pass raw strings or byte buffers directly into the parser functions without a network connection.

CHAPTER 13

Drop-Copy Spy (`pm-dc-spy`)

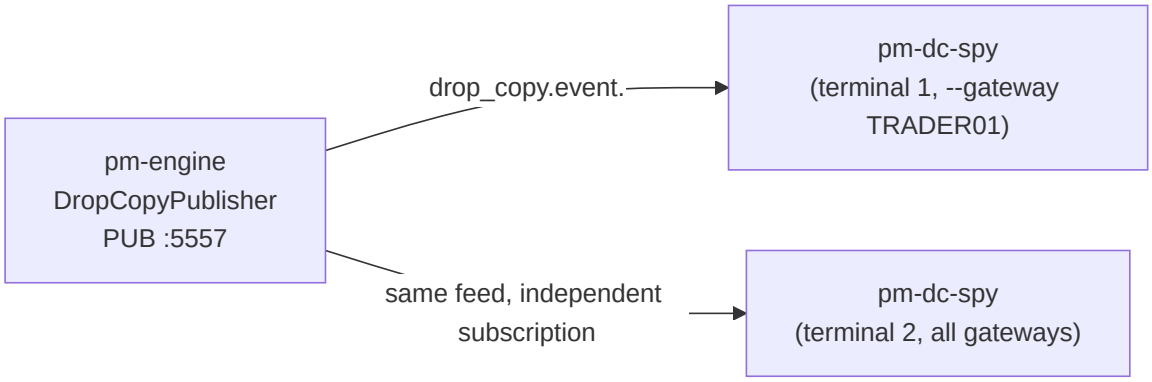
Learning objectives

After reading this page you will understand:

- What `pm-dc-spy` is and how it differs from `pm-calf-spy`/`pm-ralf-spy`
- Why the drop-copy feed needs no handshake or heartbeat, unlike CALF/RALF
- How to filter by gateway with `--gateway`
- The difference between `--format human` and `--format json`, and when to reach for each
- How `--replay-of` lets you observe `DropCopyPublisher.replay()` calls
- How to run several instances at once to watch different gateways on separate terminals

13.1 What this tool is

`pm-dc-spy` is a read-only command-line client for the matching engine's drop-copy feed (`edumatcher.engine.drop` ZMQ `PUB` socket on port **5557**). It opens a `zmq.SUB` connection, subscribes to fill events for one gateway or all gateways, and prints every message it receives — either as a colourised, human-readable log line or as one JSON object per line.



It exists purely to make the drop-copy feed observable: to answer "what does the engine actually publish on 5557 when a fill happens?" without writing a subscriber. It never publishes anything back onto the bus, and it is safe to run any number of instances at once — ZeroMQ PUB/SUB fans out independently to every connected subscriber.

13.2 Why not the example subscriber?

Subscribing to the drop copy feed shows a minimal hand-rolled subscriber meant to be read and adapted for your own risk/clearing integration. `pm-dc-spy` is a *general-purpose inspection tool*: gateway filtering, machine-readable output for piping into `jq`/`grep`/a file, a `--count` flag for scripted one-shot captures, and a `--replay-of` flag for watching `replay()` traffic. Reach for the example code when you're writing your own drop-copy consumer; reach for `pm-dc-spy` when you just want to look at the wire.

13.3 Not a protocol client — a plain ZMQ SUB

`pm-calf-spy` and `pm-ralf-spy` are TCP clients for text protocols with a `HELLO`/`WELCOME` handshake and a `PING`/`PONG` heartbeat to survive an idle timeout. The drop-copy feed is architecturally different: it is a **plain ZeroMQ PUB/SUB stream** (see Drop Copy — Architecture). There is no session to establish and no idle timeout to defend against — `zmq.SUB` sockets subscribe by topic prefix and ZMQ handles reconnection transparently. This means `pm-dc-spy`:

- Has no `--client-name`, `--ping-interval`, or `--show-heartbeats` flags — there is no handshake and no heartbeat protocol to configure.
- Prints no `WELCOME` banner — the connection line is generated locally by `pm-dc-spy` itself, not received from the engine.
- Does not fail at startup if `pm-engine` isn't running yet or hasn't bound port 5557: `zmq.SUB.connect()` succeeds immediately and ZMQ will start delivering messages as soon as a `PUB` socket appears at that address. Use `--count` with a short timeout expectation, or just watch the terminal, to confirm you're actually receiving data.

13.4 Starting point

```
pm-dc-spy
```

`pm-engine` must already be running with the drop-copy publisher bound (default `127.0.0.1:5557` — see Drop Copy). You will not see any lines until a fill occurs.

Connection options:

Flag	Default	Description
`--host`	`127.0.0.1`	Drop-copy `PUB` socket host
`--port`	`5557`	Drop-copy `PUB` socket port

Subscription filtering:

Flag	Default	Description
`--gateway`	*(none)*	Only show fills for this gateway (subscribes to `drop_copy.event.<GW_ID>`). Default subscribes to the `drop_copy.event.` prefix, i.e. every gateway

Flag	Default	Description
`--replay-of`	*(none)*	Also subscribe to `drop_copy.replay.<RECIPIENT_ID>` — the topic `DropCopyPublisher.replay()` publishes on. See Observing replay

No entitlement checks

Same caveat as the underlying feed (see Drop Copy — no authentication): any process — including `pm-dc-spy` — that can reach the drop-copy port can subscribe to **any** gateway's fills. `--gateway` is a convenience filter for readability, not an access control.

Output options:

Flag	Default	Description
`--format`	`human`	`human` (colourised log line) or `json` (one `json.dumps` object per line)
`--raw`	off	Also echo the raw topic + JSON payload under each formatted line (human format only)
`--no-color`	off	Disable ANSI colour even on a terminal
`--count N`	`0`	Exit after N messages (`0` = run until Ctrl-C)

Diagnostics: `--log-level`, `-v`/`--verbose`, `-q`/`--quiet`, `--version`, `--help` — same conventions as every other `pm-*` process (see Reference Manual → Environment variables).

13.5 Human-readable output

One line per fill: a local wall-clock timestamp, `FILL` (or `REPLAY`), the gateway ID, symbol, sequence number, `qty@price`, the liquidity flag (colour-coded — blue for `MAKER`, magenta for `TAKER`), and any remaining fields as `KEY=VALUE` pairs:

```
◆ pm-dc-spy connected to 127.0.0.1:5557, subscribed to drop_copy.event.* (all gateways) (Ctrl-C to stop)
10:02:17.512  FILL     TRADER01   AAPL       #1      100@150.05       TAKER  order_id=ord-001 trade_ids=['000
10:02:17.520  FILL     TRADER02   AAPL       #2      100@150.05       MAKER  order_id=ord-104 trade_ids=['000
```

There is no `remaining_qty` field — the drop-copy payload does not carry it (see Drop Copy — `order.fill` event). The "remaining fields" here are `order_id` and `trade_ids`, the only two payload keys left over once the envelope fields (`seq`, `timestamp`, `gateway_id`, `symbol`, `fill_qty`, `fill_price`, `liquidity_flag`) are printed positionally.

Recall that every trade produces **two** drop-copy events, one per counterparty — the pair above is a single matched trade, TAKER and MAKER side by side.

Pass `--raw` to also print the exact topic and JSON payload underneath:

```
10:02:17.512  FILL     TRADER01   AAPL       #1      100@150.05       TAKER  order_id=ord-001 trade_ids=['000042-0000
  drop_copy.event.TRADER01|{"event_type": "order.fill", "fill_price": 150.05, "fill_qty": 100, "gateway_id": "TRADER0
```

13.6 JSON output

`--format json` prints one JSON object per line — no banner, no colour, straightforward to pipe into `jq`, log to a file, or feed into another program. Every field from the drop-copy payload is preserved verbatim at the top level; `topic`, `recv_ts`, and a derived `replay` boolean are added:

```
{"recv_ts": 1784577849.634, "topic": "drop_copy.event.TRADER01", "replay": false, "seq": 1, "timestamp": 170000000000
```

Typical uses:

```
# Only TRADER01's fills, as they happen
pm-dc-spy --gateway TRADER01

# Capture the first 100 fills across all gateways for later analysis
pm-dc-spy --format json --count 100 > fills.jsonl

# Fills where AAPL was the symbol, piped through jq
pm-dc-spy --format json | jq 'select(.symbol == "AAPL")'
```

13.7 Observing replay

`DropCopyPublisher.replay(recipient_id, from_seq, gateway_id=...)` is currently an in-process method with no external wire protocol to trigger it (see Drop Copy — Replay). `pm-dc-spy` cannot request a replay itself, but `--replay-of RECIPIENT_ID` lets you *watch* one happen — useful when a test or embedded consumer calls `replay()` and you want to see exactly what gets re-published:

```
pm-dc-spy --replay-of MY_RISK_SYS
```

Replayed messages are tagged `REPLAY` instead of `FILL` (yellow instead of green in human output; `"replay": true` in JSON output), so live and replayed traffic are easy to tell apart even when watching both at once with `--gateway` and `--replay-of` together.

13.8 Running several instances at once

Since each `pm-dc-spy` process opens its own independent `SUB` socket, you can split gateways across terminals instead of filtering one firehose:

```
# Terminal 1 — one participant's fills
pm-dc-spy --gateway TRADER01

# Terminal 2 — everything, as JSON, captured to a file
pm-dc-spy --format json > all_fills.jsonl
```

None of these interfere with each other or with any other drop-copy consumer (a real risk system, the example subscriber, etc.) already connected — ZeroMQ `PUB`/`SUB` fans out independently per subscriber.

13.9 Connection and protocol errors

- If the ZMQ socket cannot be created at all (malformed `--host`), `pm-dc-spy` prints `pm-dc-spy: could not connect to HOST:PORT: ...` and exits `1`.

- A refused/unreachable connection (e.g. `pm-engine` not running yet) does **not** raise an error at startup — `zmq.SUB.connect()` succeeds immediately regardless of whether anything is listening, and ZMQ transparently reconnects once a `PUB` socket appears at that address. If you see no output, check that `pm-engine` is running and has bound port 5557 (it logs a warning and continues without drop copy if the bind fails — see Drop Copy — Startup and shutdown).
- Ctrl-C (or reaching `--count`) closes the connection cleanly and prints `pm-dc-spy: connection closed.`

13.10 See also

In this book:

- Drop Copy — operational guide, message format, replay semantics, and the Python subscriber example
- CALF Protocol Spy (pm-calf-spy) — the analogous inspection tool for the pre-trade market-data feed
- RALF Protocol Spy (pm-ralf-spy) — the analogous inspection tool for the post-trade text protocol

Reference Manual:

- Processes, Environment, and Ports — where `pm-dc-spy` sits in the process model

CHAPTER 14

CALF Protocol Spy (`pm-calf-spy`)

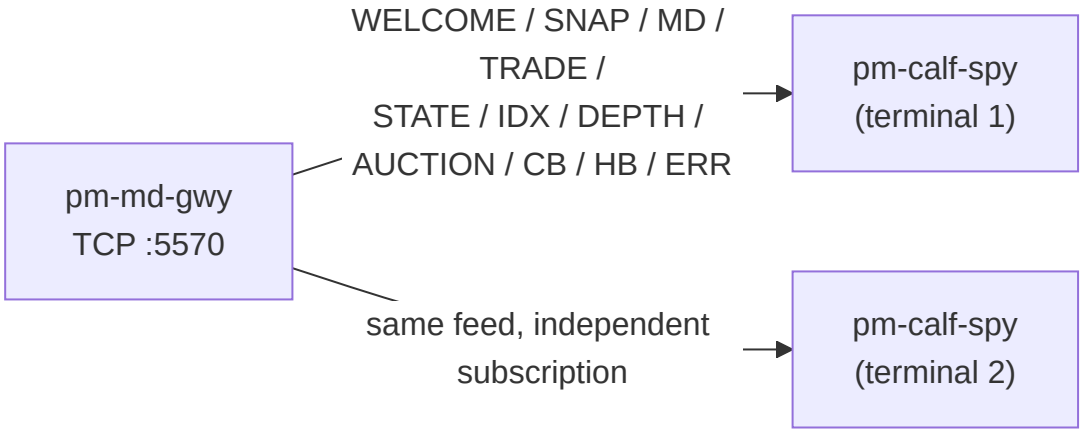
Learning objectives

After reading this page you will understand:

- What `pm-calf-spy` is and why it exists alongside the Python/C example subscribers in `docs/examples/calf/`
- How to filter by channel and symbol with `--channels`/`--symbols`
- The difference between `--format human` and `--format json`, and when to reach for each
- How to run several instances at once against the same gateway to watch different channels on separate terminals
- How `--resume` and `--count` work
- Why a listen-only session needs no keepalive, and what `--ping-interval` is still for
- How connection and protocol errors are reported

14.1 What this tool is

`pm-calf-spy` is a read-only command-line client for `pm-md-gwy`. It opens one CALF TCP session, sends `HELLO` and `SUB` on your behalf, and prints every line the gateway sends back — `WELCOME`, `SNAP`, live data (`MD`/`TRADE`/`STATE`/`IDX`/`DEPTH`/`AUCTION`/`CB`), `HB`, and `ERR` — either as a colourised, human-readable log line or as one JSON object per line.



It exists purely to make the protocol observable: to answer "what does CALF actually send when X happens?" without writing a client or reading `docs/examples/calf/calf_subscriber.py`. It never places orders, never mutates exchange state, and it is safe to run any number of instances against the

same gateway at once — pm-md-gwy accepts an arbitrary number of concurrent TCP connections (up to market_data_gateway.max_connections), and each pm-calf-spy process has its own independent subscription set.

14.2 Why not the example subscriber?

docs/examples/calf/calf_subscriber.py (see Market Data Feed — Python subscriber example) is a *library-style* example meant to be read and adapted: it hard-codes a fixed channel set (TOP, TRADE, STATE, DEPTH, optionally INDEX) and formats output for a specific demo. pm-calf-spy is a *general-purpose inspection tool*: any channel combination, any symbol filter, machine- readable output for piping into jq/grep/a file, and a --count flag for scripted one-shot captures. Reach for the example code when you're writing your own CALF client; reach for pm-calf-spy when you just want to look at the wire.

14.3 Starting point

```
pm-calf-spy --channels TOP,TRADE --symbols AAPL
```

pm-md-gwy must already be running and reachable (default 127.0.0.1:5570). pm-engine does not strictly need to be running for the handshake to succeed, but you will not see any live data until it is.

Connection options:

Flag	Default	Description
--host	127.0.0.1	pm-md-gwy TCP host
--port	5570	pm-md-gwy TCP port
--client-name	calf-spy-<pid>	HELLO|CLIENT= identifier reported in gateway logs
--ping-interval	60	Seconds between PING frames sent to the gateway; 0 disables the heartbeat. See Keeping the connection alive

Subscription filtering:

Flag	Default	Description
--channels	*	Comma-separated channels, e.g. TOP,TRADE,CB. * subscribes to every channel in WELCOME|CH_SUPPORTED= (falling back to TOP,TRADE,STATE if that field is absent — see pre-1.0.0 gateway note)

Flag	Default	Description
`--symbols`	*	Comma-separated symbols, e.g. `AAPL,MSFT`. * requests the wildcard for every channel that allows it (`TOP`, `TRADE`, `STATE`, `AUCTION`)
`--resume`	*(none)*	One-shot `CH:SYM:LASTSEQ`, e.g. `TOP:AAPL:1042` — requests single-stream replay just after the handshake (sends `RESUME|CH=..|SYM=..|LASTSEQ=..`)

`--channels`/`--symbols` are applied as a single `SUB` for the full Cartesian product. If a requested combination is invalid — `SYM=*` on a channel that rejects it (`INDEX`, `DEPTH`, `CB`), or an unknown symbol — the gateway's `ERR` line is printed like any other line rather than aborting the whole session, so you can see exactly what was rejected and why. Pass explicit symbols for `INDEX`/`DEPTH`/`CB` rather than relying on *.

Output options:

Flag	Default	Description
`--format`	`human`	`human` (colourised log line) or `json` (one `json.dumps` object per line)
`--raw`	off	Also echo the raw wire line under each formatted line (human format only)
`--no-color`	off	Disable ANSI colour even on a terminal
`--show-heartbeats`	off	Also print `HB` and `PONG` lines (suppressed by default to reduce noise)
`--count N`	`0`	Exit after N data-carrying lines (`0` = run until Ctrl-C); `HB` lines don't count, but `PONG` replies do

Diagnostics: `--log-level`, `-v`/`--verbose`, `-q`/`--quiet`, `--version`, `--help` — same conventions as every other `pm-*` process (see Reference Manual → Environment variables).

14.4 Keeping the connection alive

`pm-calf-spy` is purely a listener: after the initial `HELLO`/`SUB` handshake it has nothing more to say. **This is fine, and needs no help from you.** `pm-md-gwy`'s idle timer counts traffic in either direction, and the gateway's own heartbeats keep the session alive, so a receive-only client stays connected indefinitely without sending a thing (see Market Data Feed — Configuration).

> Earlier gateway builds advanced the idle timer only on *inbound* bytes, so a silent listener really was dropped every `idle_timeout_sec`. The `--ping-interval` thread below exists because of that; it is now a diagnostic convenience rather than a requirement.

`pm-calf-spy` still sends `PING` from a background thread every `--ping-interval` seconds (default `60`), and the gateway replies with a `PONG` (suppressed from the default view the same way `HB` is — pass `--show-heartbeats` to see both). It is a useful explicit round-trip check that the session is genuinely alive rather than merely un-errored. Pass `0` to disable it.

14.5 Human-readable output

One line per event: a local wall-clock timestamp, the message type, the channel (colour-coded so several interleaved channels stay visually distinct), the symbol, the sequence number, and the remaining fields as `KEY=VALUE` pairs, sorted for stable reading:

```
◆ pm-calf-spy connected to 127.0.0.1:5570 as calf-spy-40213 (Ctrl-C to stop)
10:02:17.041  WELCOME   CH_SUPPORTED=AUCTION,CB,DEPTH,INDEX,STATE,TOP,TRADE GW=md-gwy01 HBINT=1 PROTO=CALF1 REPLAY=3
10:02:17.048  SNAP      TOP       AAPL       #1       ASK=150.12 ASKSZ=900 BID=150.10 BIDSZ=1200 LAST=150.11 LASTSZ=300
10:02:17.512  MD        TOP       AAPL       #2       BID=150.11 BIDSZ=1400
10:02:18.203  TRADE     TRADE     AAPL       #44      PX=150.12 QTY=200 SIDE=BUY
10:02:20.001  CB        CB        AAPL       #4       CORRHI=165.11 CORRLO=135.09 EXP=0 LEVEL=L2 REFPX=150.10 RESUMEAT=
10:20:00.010  CB        CB        AAPL       #5       CORRHI=180.12 CORRLO=120.08 EXP=1 IMB=BUY INDICPX=182.00 INDICQTY=
```

Session-level messages that carry no channel/symbol of their own (`WELCOME`, `HB`, `PONG`, `EXIT`) are rendered without the channel/symbol/sequence columns. `ERR` lines highlight the `CODE` field:

```
10:02:21.010  ERR       INVALID_SYMBOL SYM=*
```

Pass `--raw` to also print the exact wire line underneath, for comparing the rendering against the actual bytes:

```
10:02:17.512  MD        TOP       AAPL       #2       BID=150.11 BIDSZ=1400
  MD|CH=TOP|SYM=AAPL|SEQ=2|TS=2026-06-30T09:30:00.500Z|BID=150.11|BIDSZ=1400
```

14.6 JSON output

`--format json` prints one JSON object per line for market-data records, straightforward to pipe into `jq`, log to a file, or feed into another program. It does **not** suppress the connect banner or the closing "connection closed" line — both still print to the same stdout stream in every `--format`, so a strict line-oriented JSON consumer (e.g. `jq -R` reading raw lines, or a `.jsonl` file written via shell redirection) will see one non-JSON line at the start and one at the end. The envelope fields (`CH`, `SYM`, `SEQ`) are lifted to top-level keys for easy filtering; every field, including the envelope ones, is also kept verbatim under `fields` so nothing is lost relative to the raw line:

```
{"recv_ts": 1784576040.335, "msg_type": "SNAP", "ch": "TOP", "sym": "AAPL", "seq": 1, "ts": "2026-07-20T10:02:17.048
```

Typical uses:

```
# Only circuit-breaker events, as they happen
pm-calf-spy --channels CB --format json | jq 'select(.fields.STATUS == "HALTED")'

# Capture the first 50 trades to a file for later analysis
pm-calf-spy --channels TRADE --symbols '*' --format json --count 50 > trades.jsonl
```

14.7 Running several instances at once

Since each `pm-calf-spy` process opens its own independent TCP connection and subscription set, you can split channels across terminals instead of filtering one firehose:

```
# Terminal 1 — order book activity for one symbol
pm-calf-spy --channels TOP,DEPTH --symbols AAPL

# Terminal 2 — market-wide trade tape
pm-calf-spy --channels TRADE --symbols '*'

# Terminal 3 — everything auction- and circuit-breaker-related
pm-calf-spy --channels AUCTION,CB --symbols AAPL,MSFT
```

None of these interfere with each other or with any other CALF client (a real trading bot, `calf_subscriber.py`, etc.) already connected to the same gateway — `pm-md-gwy` fans out independently per session.

14.8 Connection and protocol errors

- If the initial TCP connect fails (gateway not running, wrong host/port), `pm-calf-spy` prints `pm-calf-spy: could not connect to HOST:PORT: ...` and exits `1` — no retry loop.
- If the gateway rejects the handshake itself (`ERR|CODE=AUTH_REQUIRED` in reply to `HELLO` — e.g. a `--client-name` over 32 characters), the spy prints `pm-calf-spy: gateway rejected HELLO: AUTH_REQUIRED ...` and exits `1`.
- If the connection closes before any reply to `HELLO`, the spy reports `connection closed before WELCOME (is the gateway at max_connections?)` — a gateway at `market_data_gateway.max_connections` accepts a connection and closes it without a line.
- Once connected, any `ERR` the gateway sends in response to `SUB` (bad channel, bad symbol, wildcard misuse, subscription limit) is printed like any other line — the session stays open so you can see the rejection and keep watching whatever subscriptions *did* succeed.
- If the gateway ends the session (`EXIT|REASON=idle_timeout`, after writes to the spy stopped succeeding; or `ERR|CODE=SLOW_CLIENT`), the line is printed and the spy exits once the gateway closes the socket.
- Ctrl-C (or reaching `--count`) ends the session with `EXIT` — the gateway flushes and closes — and prints `pm-calf-spy: connection closed.`

14.9 See also

In this book:

- Appendix — CALF Protocol — normative wire format, full field tables, sequencing rules
- Drop-Copy Spy (pm-dc-spy) — the analogous inspection tool for the per-participant fill feed

Operator's Guide:

- Market Data Feed (CALF) — operational guide, wire examples, and the Python/C example subscribers

Reference Manual:

- Processes, Environment, and Ports — where `pm-calf-spy` sits in the process model

Training Guide:

- CALF Market-Data Gateway Protocol Training — hands-on exercises using `nc` and the example subscribers

CHAPTER 15

RALF Protocol Spy (`pm-ralf-spy`)

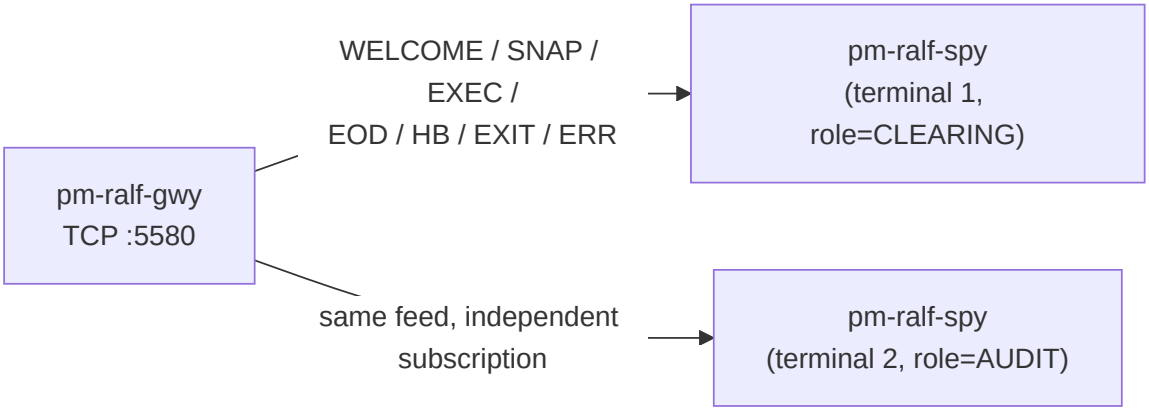
Learning objectives

After reading this page you will understand:

- What `pm-ralf-spy` is and why it exists alongside `ralf_subscriber.py` in `docs/examples/ralf/`
- How `--role` drives entitlement, and why it is required (unlike CALF, which has no per-client role)
- How to filter by channel and symbol with `--channels`/`--symbols`
- The difference between `--format human` and `--format json`, and when to reach for each
- How `--lastseq` differs from CALF's `RESUME`
- How to run several instances at once, each with a different role, to watch different post-trade streams on separate terminals
- Why a listen-only session needs no keepalive, and what `--ping-interval` is still for

15.1 What this tool is

`pm-ralf-spy` is a read-only command-line client for `pm-ralf-gwy`. It opens one RALF TCP session, sends `HELLO` (with a chosen `ROLE`) and `SUB` on your behalf, and prints every line the gateway sends back — `WELCOME`, `SNAP`, live data (`EXEC`/`EOD`), `HB`, `EXIT`, and `ERR` — either as a colourised, human-readable log line or as one JSON object per line.



It exists purely to make the protocol observable: to answer "what does RALF actually send when a trade executes, or at end-of-day?" without writing a client. It never places orders, never mutates exchange state, and it is safe to run any number of instances against the same gateway at once — `pm-ralf-gwy` accepts an arbitrary number of concurrent TCP connections, and each `pm-ralf-spy`

process has its own independent role and subscription set.

15.2 Why not `ralf_subscriber.py`?

`docs/examples/ralf/ralf_subscriber.py` (see Post-Trade Dissemination — Python subscriber example) is a *library-style* example meant to be read and adapted: it demonstrates gap detection, dedup, and a specific role/channel selection for a tutorial walkthrough. `pm-ralf-spy` is a *general-purpose inspection tool*: any role, any channel/symbol combination, machine-readable output for piping into `jq`/`grep`/a file, and a `--count` flag for scripted one-shot captures. Reach for the example code when you're writing your own RALF client; reach for `pm-ralf-spy` when you just want to look at the wire.

15.3 RALF's role model, briefly

Unlike CALF, every RALF client authenticates as one of three roles — `CLEARING`, `DROP_COPY`, or `AUDIT` — sent once in `HELLO|ROLE=`. The role determines which channels you may subscribe to: `CLEARING` and `DROP_COPY` may only subscribe to their own same-named channel; `AUDIT` may subscribe to any of the three. There is no `CH_SUPPORTED`-equivalent capability field in `WELCOME` — the channel set is fixed and known ahead of time (`CLEARING`, `DROP_COPY`, `AUDIT`), so there is nothing to discover.

15.4 Starting point

```
pm-ralf-spy --role AUDIT --symbols AAPL
```

`pm-ralf-gwy` must already be running and reachable (default `127.0.0.1:5580`). `pm-engine` does not strictly need to be running for the handshake to succeed, but you will not see any live data until trades occur (and until the next end-of-day cycle for `EOD`).

Connection options:

Flag	Default	Description
`--host`	`127.0.0.1`	`pm-ralf-gwy` TCP host
`--port`	`5580`	`pm-ralf-gwy` TCP port
`--client-name`	`ralf-spy-<pid>`	`HELLO|CLIENT=` identifier reported in gateway logs
`--ping-interval`	`60`	Seconds between `PING` frames sent to the gateway; `0` disables the heartbeat. See Keeping the connection alive

Subscription filtering:

Flag	Default	Description
`--role`	`AUDIT`	`HELLO|ROLE=` to authenticate as: `CLEARING`, `DROP_COPY`, or `AUDIT`. `AUDIT` is the most useful default for a spy tool since it is entitled to every channel

Flag	Default	Description
`--channels`	*	Comma-separated channels, e.g. `CLEARING,DROP_COPY`. * subscribes to every channel `--role` is entitled to (all three for `AUDIT`; just its own name for `CLEARING`/`DROP_COPY`)
`--symbols`	*	Comma-separated symbols, e.g. `AAPL,MSFT`. * subscribes to every symbol — RALF has no per-channel wildcard restriction the way CALF's `INDEX`/`DEPTH`/`CB` do
`--lastseq`	`0`	Requests replay on connect via `HELLO|LASTSEQ=N` for every channel `--role` is entitled to (`0` = no replay)

`--channels`/`--symbols` are applied as a single `SUB` for the full Cartesian product. If a requested channel is outside your role's entitlement (e.g. `--role CLEARING --channels DROP_COPY`), the gateway's `ERR|CODE=ENTITLEMENT_DENIED` is printed like any other line rather than aborting the whole session, so you can see exactly what was rejected.

-lastseq vs. CALF's RESUME

RALF has no separate resume message: a non-zero `LASTSEQ` goes directly on `HELLO` and requests replay across **every** channel your role is entitled to at once (not scoped to one channel/symbol pair the way CALF's `RESUME|CH=..|SYM=..` is). If the requested sequence is older than `replay_retention_sec`, the gateway sends `ERR|CODE=REPLAY_MISS` followed by a fresh `SNAP` — accept it and reset any local state.

Output options:

Flag	Default	Description
`--format`	`human`	`human` (colourised log line) or `json` (one `json.dumps` object per line)
`--raw`	off	Also echo the raw wire line under each formatted line (human format only)
`--no-color`	off	Disable ANSI colour even on a terminal
`--show-heartbeats`	off	Also print `HB` and `PONG` lines (suppressed by default to reduce noise)

`--count N`	`0`	Exit after N data-carrying lines (`0` = run until Ctrl-C); `HB` lines don't count, but `PONG` replies do

Diagnostics: `--log-level`, `-v`/`--verbose`, `-q`/`--quiet`, `--version`, `--help` — same conventions as every other `pm-*` process.

15.5 Keeping the connection alive

`pm-ralf-spy` is purely a listener: after the initial `HELLO`/`SUB` handshake it has nothing more to say. **This is fine, and needs no help from you.** `pm-ralf-gwy`'s idle timer counts traffic in either direction, and the gateway sends `HB` every `HBINT` seconds, so a client that keeps reading stays connected indefinitely without sending a thing. The idle timeout (`EXIT|REASON=idle_timeout`) fires only once writes to a client have stopped succeeding — a client that has stopped reading.

`pm-ralf-spy` still sends `PING` from a background thread every `--ping-interval` seconds (default `60`), and the gateway replies with a `PONG` (suppressed from the default view the same way `HB` is — pass `--show-heartbeats` to see both). It is an explicit round-trip check that the session is genuinely alive rather than merely un-errored. Pass `0` to disable it.

15.6 Human-readable output

One line per event: a local wall-clock timestamp, the message type, the channel (colour-coded so several interleaved channels stay visually distinct), the symbol, the sequence number, and the remaining fields as `KEY=VALUE` pairs, sorted for stable reading:

```
◆ pm-ralf-spy connected to 127.0.0.1:5580 as ralf-spy-40213 role=AUDIT (Ctrl-C to stop)
10:02:17.041  WELCOME   GW=ralf-gwy01 HBINT=1 PROTO=RALF1 REPLAY=86400 ROLE=AUDIT
10:02:17.048  SNAP      CLEARING,DROP_COPY,AUDIT AAPL       #0
10:02:18.203  EXEC      AUDIT       AAPL        #1       BUY_GW=TRADER01 BUY_ORDER_ID=1001 EXEC_ID=44 MATCH_ID=44 PX=150
16:00:00.500  EOD       AUDIT       AAPL        #2       EXEC_COUNT=37 TRADE_COUNT=37
```

Session-level messages that carry no channel/symbol of their own (`WELCOME`, `HB`, `PONG`, `EXIT`) are rendered without the channel/symbol/sequence columns. `ERR` lines highlight the `CODE` field:

```
10:02:21.010  ERR       ENTITLEMENT_DENIED  DETAIL=role 'CLEARING' cannot access CH=DROP_COPY
```

An idle-timeout disconnect from the gateway looks like:

```
10:05:00.000  EXIT      REASON=idle_timeout TS=2026-07-20T10:05:00.000Z
```

Pass `--raw` to also print the exact wire line underneath, for comparing the rendering against the actual bytes.

15.7 JSON output

`--format json` prints one JSON object per line — no banner, no colour, straightforward to pipe into `jq`, log to a file, or feed into another program. The envelope fields (`CH`, `SYM`, `SEQ`) are lifted to top-level keys for easy filtering; every field, including the envelope ones, is also kept verbatim under `fields` so nothing is lost relative to the raw line:

```
{"recv_ts": 1784577849.634, "msg_type": "EXEC", "ch": "AUDIT", "sym": "AAPL", "seq": 1, "ts": "2026-07-20T10:02:18.20
```

Typical uses:

```
# Only fills, as they happen, for one symbol
pm-ralf-spy --role AUDIT --symbols AAPL --format json | jq 'select(.msg_type == "EXEC")'

# Capture the clearing feed's end-of-day summaries to a file
pm-ralf-spy --role CLEARING --format json | jq 'select(.msg_type == "EOD")' > eod.jsonl
```

15.8 Running several instances at once

Since each `pm-ralf-spy` process opens its own independent TCP connection, role, and subscription set, you can split roles/channels across terminals instead of filtering one firehose:

```
# Terminal 1 — clearing view only
pm-ralf-spy --role CLEARING

# Terminal 2 — drop-copy view only
pm-ralf-spy --role DROP_COPY

# Terminal 3 — audit view of everything, one symbol
pm-ralf-spy --role AUDIT --symbols AAPL
```

None of these interfere with each other or with any other RALF client already connected to the same gateway — `pm-ralf-gwy` fans out independently per session.

15.9 Connection and protocol errors

- If the initial TCP connect fails (gateway not running, wrong host/port), `pm-ralf-spy` prints `pm-ralf-spy: could not connect to HOST:PORT: ...` and exits `1` — no retry loop.
- If the gateway rejects the handshake itself — missing `CLIENT`/`PROTO`/ `ROLE` (`ERR|CODE=AUTH_REQUIRED`), or a role your gateway build doesn't allow (`ERR|CODE=ENTITLEMENT_DENIED`) — that is reported and the process exits `1`.
- Once connected, any `ERR` the gateway sends in response to `SUB` (unknown channel, role/channel mismatch) is printed like any other line — the session stays open so you can see the rejection and keep watching whatever subscriptions *did* succeed.
- If the connection closes before any reply to `HELLO`, the spy reports `connection closed before WELCOME (is the gateway at max_connections?)` — a gateway at `post_trade_gateway.max_connections` accepts a connection and closes it without a line.
- If the gateway sends `EXIT` (`REASON=idle_timeout`: writes to the spy had stopped succeeding, e.g. the terminal was suspended), `pm-ralf-spy` prints it and closes cleanly rather than trying to keep reading a closing socket.
- Ctrl-C (or reaching `--count`) ends the session with `EXIT` — the gateway flushes and closes — and prints `pm-ralf-spy: connection closed.`

15.10 See also

In this book:

- Appendix — RALF Protocol — normative wire format, full field tables, sequencing rules
- CALF Protocol Spy (pm-calf-spy) — the analogous inspection tool for the pre-trade market-data feed
- Drop-Copy Spy (pm-dc-spy) — the analogous inspection tool for the engine's raw drop-copy feed (distinct from RALF's `DROP_COPY` channel — see comparison)

Operator's Guide:

- Post-Trade Dissemination (RALF) — operational guide, wire examples, and the Python/C example subscribers

Reference Manual:

- Processes, Environment, and Ports — where `pm-ralf-spy` sits in the process model

CHAPTER 16

Message Reference

Learning objectives

After reading this page you will understand:

- What a message is in the context of a distributed bus system and how it differs from a function call or shared data structure
- How messages are defined in real systems (schema registries, IDL, plain JSON) and the pragmatic trade-offs EduMatcher makes
- What ZeroMQ requires of a message — frames, encoding, topic filters
- Why ZeroMQ has no broker, and what that means for reliability and operational complexity compared with broker-based systems (Kafka, RabbitMQ, NATS)
- The full catalogue of messages used in EduMatcher, their fields, and which processes produce and consume each one

Where the rest of this page comes from

Everything below the narrative sections — the topic index, the record types and one section per message — is **generated from `spec/messages/*.yaml`** by `pm-msgen generate`, and `pm-msgen check` fails in CI when the page and the spec disagree.
This file, `000-preamble.md`, is the hand-written half: bus concepts, transports and the CALF protocol, none of which the spec can state. Edit it freely. To change anything about a *message*, edit its spec file.

16.1 Background — Messages in a Bus System

16.1.1 What is a message?

A function call passes data synchronously inside a process: the caller blocks until the callee returns. A **message** is an asynchronous unit of data sent between processes. The sender does not wait for a response; it hands the message to the transport layer and moves on.

Messages carry three things:

1. **Identity** — what kind of event this is (the topic or message type)
2. **Payload** — the data describing the event
3. **Routing metadata** — information the transport needs to deliver it (addresses, topic filters, sequence numbers)

16.1.2 How message formats are defined in real systems

In production systems there are three common approaches to defining what a message looks like:

Schema registries (Avro, Protobuf, Thrift) Messages are defined in an Interface Definition Language (IDL) file. A code generator produces serialisers and deserialisers for every target language. The registry enforces compatibility: a new field may be added but existing fields cannot be removed or re-typed without a version bump. Kafka and gRPC use this model.

Canonical JSON/XML schemas (JSON Schema, OpenAPI, AsyncAPI) Message shapes are described in a human-readable document (like AsyncAPI for event-driven systems). Any process that speaks JSON can produce or consume a message without a code generator. The schema document is the contract; violations surface only at runtime unless you add a validation library.

Hardcoded structures The simplest approach — message shape is implicit in the code that creates and reads it. No IDL, no registry, no generator. Fast to build, but schema drift is invisible until something breaks at runtime.

EduMatcher uses a small, repository-local **IDL and code-generation** workflow. The YAML definitions in `spec/messages/*.yaml` are the canonical message contracts. `pm-msgen generate` produces the Python bindings, applicable C artifacts, and the generated sections of this reference; `pm-msgen check` fails when any generated output diverges from its specification. Message producers and consumers still exchange ordinary two-frame JSON over ZeroMQ, so the generated bindings remain readable alongside the wire format they enforce.

16.1.3 What ZeroMQ requires of a message

ZeroMQ does not impose a message format. It sees messages as one or more opaque **frames** — byte arrays that are sent and received atomically as a group. It is up to the application to define what those bytes mean.

EduMatcher uses exactly **two frames** for every message:

```
frame[0]  →  topic string, UTF-8 encoded
             e.g.  b"order.ack.GW01"

frame[1]  →  JSON payload, UTF-8 encoded
             e.g.  b'{"order_id": "3f2a...", "accepted": true}'
```

`frame[0]` doubles as the **PUB/SUB filter key**. A subscriber that registers for prefix `"order.ack.GW01"` will receive only messages whose first frame starts with that string — all other messages are dropped by the ZeroMQ layer before the application even sees them. This prefix-match filter is evaluated in the kernel's socket buffer, not in Python, so it adds almost no CPU overhead regardless of how many message types are on the bus.

16.1.4 ZeroMQ without a broker

Most messaging systems interpose a **broker** between producers and consumers:

```
Producer ──►  Broker  ──►  Consumer
```

The broker buffers messages, persists them to disk, routes them to the right queues, and handles consumer acknowledgements. Examples: RabbitMQ, Apache Kafka, NATS JetStream, AWS SQS.

ZeroMQ is **brokerless**. Producers connect directly to consumers (or to the engine in PUSH/PULL):

```
Gateway ──PUSH──►  Engine ──PUB──►  Subscriber A
                                ├──►  Subscriber B
                                └──►  Subscriber C
```

There is no third process in the middle. The advantages and disadvantages flow directly from that choice:

Advantages of no broker

Advantage	Detail
Lower latency	No extra network hop; messages go directly from sender to receiver
Fewer moving parts	No broker process to install, configure, monitor, or restart
No single point of failure	The engine is the bus; if the engine is up, the bus is up
Simpler deployment	`pip install pyzmq` is the entire installation

Disadvantages of no broker

Disadvantage	Detail
No persistence	If a subscriber is down when a message is published, the message is gone forever
No guaranteed delivery	PUB/SUB drops messages to slow subscribers without warning
No replay	You cannot re-consume old messages; there is no commit log
Tight coupling on addresses	Producers must know the address of the engine; adding a new engine address requires reconfiguring all clients
No backpressure	A fast publisher can overwhelm a slow subscriber; the subscriber's receive buffer fills and messages are silently discarded

For EduMatcher these trade-offs are acceptable: the system runs on localhost, sessions last hours not days, and correctness over time is handled by the GTC persistence layer rather than the message bus. For a real exchange, the audit trail would be written by a Kafka consumer (guaranteed delivery, infinite replay), and the matching engine would use a persisted queue for order intake.

16.2 Message structure

Every inter-process communication in EduMatcher is a ZeroMQ multipart message — an ordered list of byte-array frames sent and received atomically. ZeroMQ (ZMQ) is a high-performance messaging library; the application, not ZMQ, decides what the frames mean.

The first two frames are the message itself. Behind them ride the **envelope**: metadata *about* the message — a delivery sequence and a causal trace — rather than content of it:

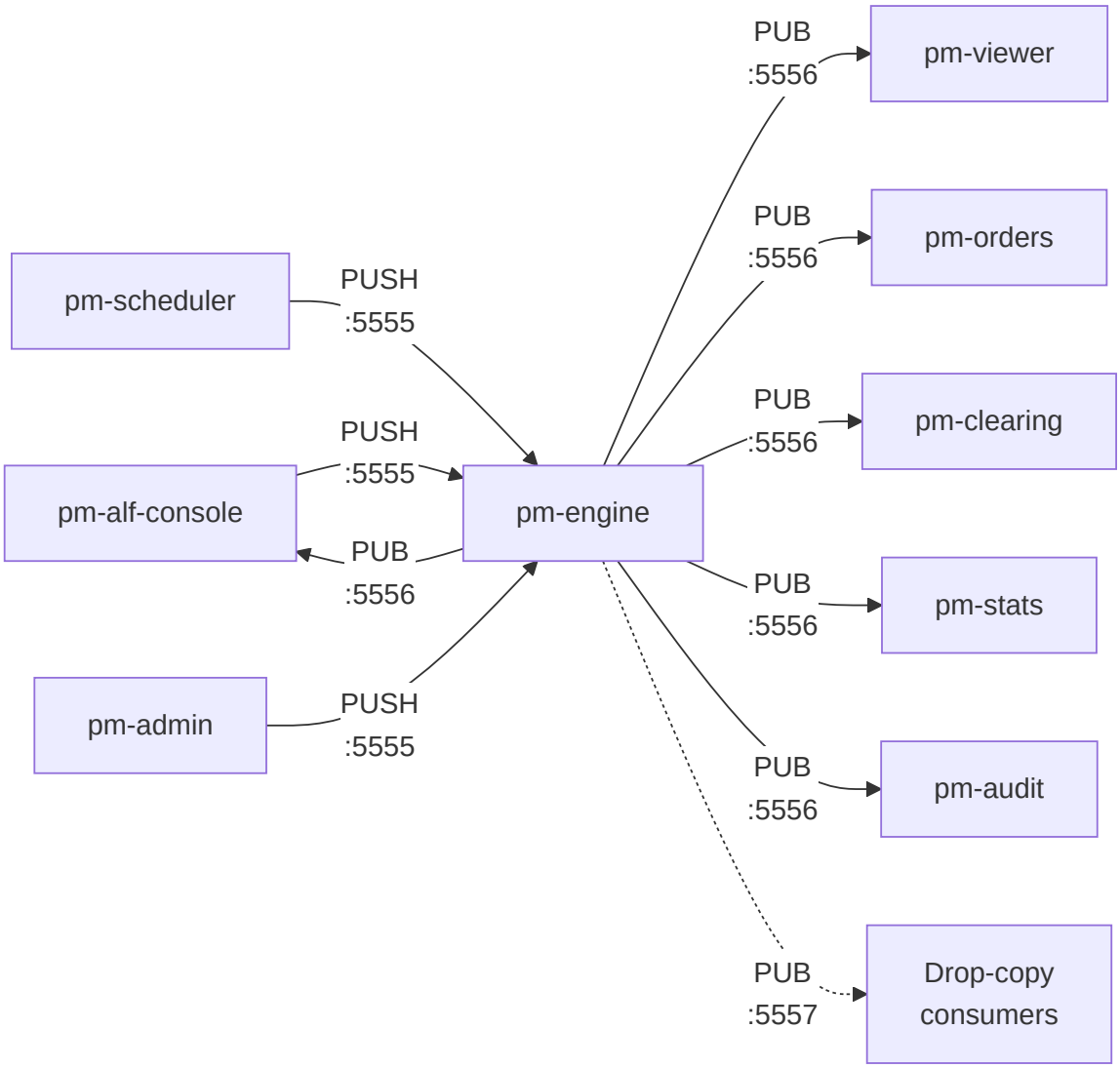
Frame	PUB (engine → subscribers)	PUSH (client → engine)
`frame[0]`	Topic string (UTF-8) — used for PUB/SUB prefix filtering	Topic string
`frame[1]`	JSON payload (UTF-8)	JSON payload
`frame[2]`	Per-topic sequence number, so a subscriber can detect a drop	Causal envelope
`frame[3]`	Causal envelope: `msg_id` / `causation_id` / `correlation_id`	—

The causal envelope is what lets `pm-audit-replay` (and `pm-audit-cli`'s `[seq=… msg=… cause=… chain=…]` log annotation) state *which submission caused which fill*, rather than inferring it from timing. `decode()` reads only `frame[0]` and `frame[1]`, so this extension cost nothing on the hot path and every existing subscriber kept working unchanged. The full wire layout, why it is frames rather than payload fields, and how to read it programmatically are covered in the architecture guide.

16.3 Transport channels

Channel	ZMQ pattern	Address	Direction
Order submission	PUSH → PULL	`tcp://127.0.0.1:5555`	Gateway → Engine
Event broadcast	PUB → SUB	`tcp://127.0.0.1:5556`	Engine → all subscribers
Drop copy	PUB → SUB	`tcp://127.0.0.1:5557`	Engine → drop-copy consumers

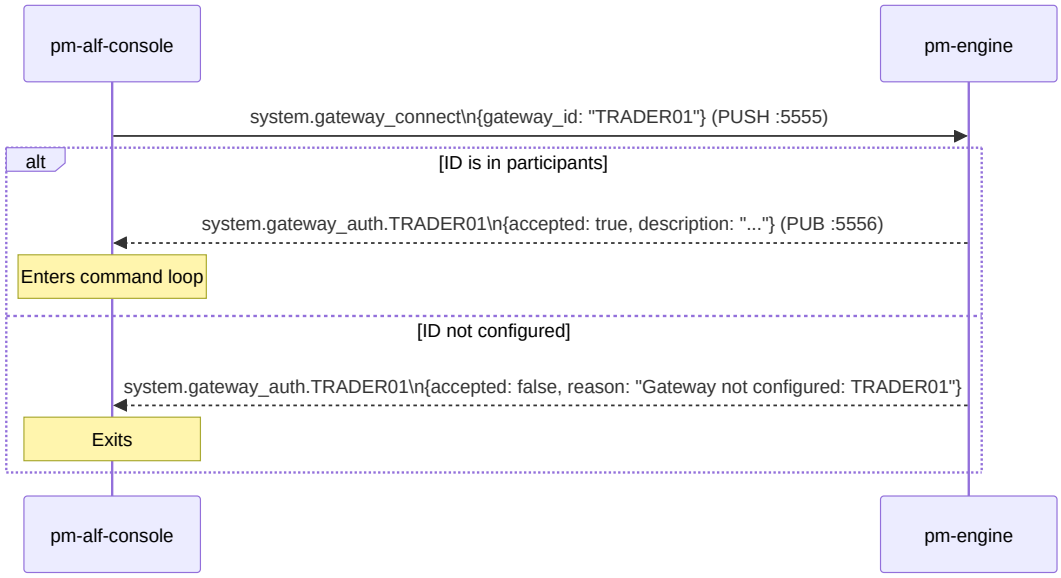
The dedicated drop-copy channel is implemented in `src/edumatcher/engine/drop_copy.py` and is documented in more detail on the Drop Copy page.



16.3.1 `system.gateway_connect`

Motivation: Enables explicit control/state synchronization so clients do not depend on timing of unsolicited events. **Published by:** Requesting client process (for example pm-alf-console, pm-admin, pm-viewer, pm-stats, bots, or API gateway) via PUSH :5555

Sent by an ALF gateway at startup to authenticate its gateway ID against `engine_config.yaml`.



Field	Type	Description
`gateway_id`	string	Gateway identifier being requested (e.g. `TRADER01`)

Reply: `system.gateway_auth.{GW_ID}`

Field	Type	Description
`gateway_id`	string	Gateway identifier
`accepted`	boolean	`true` if ID is configured in `participants`
`reason`	string	Rejection reason when `accepted=false`
`description`	string	Optional configured description for the gateway

When `accepted=false`, the gateway must terminate and MUST NOT submit orders.

Only one process may hold a given ID at a time. A connect for an ID that is already connected is refused with `reason: "Gateway already connected: TRADER01"`, and the process that was refused must not send `system.gateway_disconnect` for it, since that would tear down the live session.

16.3.2 `system.gateway_heartbeat`

Motivation: Frees a participant ID whose process died without sending `system.gateway_disconnect`.
Published by: Every process that holds an ID (`pm-alf-console`, `pm-alf-gwy`, `pm-balf-gwy`, `pm-api-gwy`, `pm-mm-bot`) via PUSH :5555

Sent every 60 seconds (`GATEWAY_HEARTBEAT_INTERVAL_SEC`), the first one immediately after the connect is accepted. It is one-way: the engine does not reply. The payload carries the sender's own `interval_sec`, so the engine needs no per-client setting. After **3 consecutive missed beats** the engine disconnects the session exactly as if `system.gateway_disconnect` had arrived, applying

the participant's `disconnect_behaviour` and publishing `system.gateway_bye.{GW_ID}` with `reason: "heartbeat_timeout"`.

Liveness is opt-in: a session that has never sent a heartbeat is never timed out, and a heartbeat for an ID that is not connected is ignored. Heartbeats are not re-published on the audit feed.

Field	Type	Description
`gateway_id`	string	The connected participant ID
`interval_sec`	integer	Seconds between this sender's beats (1 to 3600)

16.3.3 `order.new`

Motivation: Keeps order/quote lifecycle state synchronized between the initiating client and all interested subscribers. **Published by:** Requesting client process (for example pm-alf-console, pm-admin, pm-viewer, pm-stats, bots, or API gateway) via PUSH :5555

Sent by a gateway to submit a new order for matching.

Field	Type	Description
`id`	string (UUID)	Unique order identifier
`symbol`	string	Instrument ticker, e.g. `MSFT`
`side`	`"BUY"` | `"SELL"`	Order side
`order_type`	string	`MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`
`tif`	`"DAY"` | `"GTC"` | `"ATO"` | `"ATC"`	Time-in-force. `FOK`/`IOC` are `order_type` values, not `tif` values — there is no `FOK` time-in-force
`quantity`	integer	Total order quantity
`remaining_qty`	integer	Unfilled quantity (equals `quantity` on submission)
`gateway_id`	string	Originating gateway identifier, e.g. `TRADER01`
`ts_ns`	integer	Client-supplied submission time, Unix epoch **nanoseconds**. **Not** what the book uses for time priority — see `arrival_seq` — so a back-dated value cannot jump the queue

Field	Type	Description
`arrival_seq`	integer | absent	Engine-assigned monotonic arrival sequence, and the actual time-priority key. `0` on submission, meaning unassigned
`status`	string	Initial status, always `"NEW"`
`price_ticks`	integer (ticks) | null	Limit price **in ticks** (LIMIT, STOP_LIMIT, FOK, ICEBERG); `null` for MARKET, which has none
`stop_price_ticks`	integer (ticks) | null	Trigger price **in ticks** (STOP, STOP_LIMIT, TRAILING_STOP)
`visible_qty`	integer | null	Peak size for ICEBERG orders
`displayed_qty`	integer | null	Current visible slice (ICEBERG)
`trail_offset_ticks`	integer (ticks) | null	Distance **in ticks** to trail the market price (`TRAILING_STOP`)
`smp_action`	string | null	Self-match prevention: `NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`. `null` when the client omitted `SMP=`, in which case the engine resolves it to the gateway's configured `participants[].smp_action` default (else `"NONE"`) before the order reaches the book — see Configuration — Participant Fields
`client_tag`	string | absent	Optional client-supplied tag echoed back on every lifecycle event for this order (ack, fill, cancelled, expired). When present, subscribers can map events back to their submission without a FIFO scheme.
`request_tag`	string | absent	Optional amend/cancel request tag echoed on the resulting `order.amended`, `order.cancelled`, or rejected `order.ack`. Unlike `client_tag`, it identifies one request against an order, not the order itself.

Field	Type	Description
`oco_group_id`	string | null	Set once this order is linked into an OCO pair via `order.oco`; `null` on a plain submission
`combo_parent_id`	string | null	Parent `ComboOrder.id` when this order is a combo child leg; `null` for a standalone order
`leg_index`	integer | null	0-based position within the parent combo's leg list; `null` for a standalone order
`origin`	`"ORDER"` | `"QUOTE"` | `"IMPLIED"`	How this order entered the book: a direct order submission, a market-maker quote leg, or an engine-implied order
`quote_id`	string | null	Set when `origin` is `"QUOTE"`, echoing the originating `quote.new`'s identifier; `null` otherwise

[!NOTE] Prices on `order.new` and `quote.new` are **integer ticks**, not display money: 1 tick is `10^-tick_decimals` for the instrument. The replies are the other way round — `order.ack.price`, `order.fill.fill_price` and `order.amended.price` are display floats — which is why every message field carries a `unit:` declaration in `spec/messages/`, and why the generated tables below are authoritative when this narrative and the spec disagree.

Valid field combinations by order type:

`order_type`	`price_ticks`	`stop_price_ticks`	`visible_qty`	`trail_offset_ticks`	Notes
`MARKET`	—	—	—	—	Fills at best available; rejected if symbol halted
`LIMIT`	Required	—	—	—	Rests if no match; subject to collar check
`STOP`	—	Required	—	—	Triggers a market order when stop price touched
`STOP_LIMIT`	Required	Required	—	—	Triggers a limit order when stop price touched

order_type	price_ticks	stop_price_ticks	visible_qty	trail_offset_ticks	Notes
FOK	Required	—	—	—	Fill fully immediately or cancel entirely
IOC	Optional	—	—	—	Fill as much as possible immediately, cancel remainder
ICEBERG	Required	—	Required	—	Shows only visible_qty; replenishes from hidden reserve
TRAILING_STOP	—	—	—	Required	Stop price follows best opposite-side price by trail_offset_ticks

16.3.4 order.cancel

Motivation: Keeps order/quote lifecycle state synchronized between the initiating client and all interested subscribers. **Published by:** Requesting client process (for example pm-alf-console, pm-admin, pm-viewer, pm-stats, bots, or API gateway) via PUSH :5555

Sent by a gateway to cancel a resting order.

Field	Type	Description
order_id	string (UUID)	ID of the order to cancel
gateway_id	string	Gateway that owns the order

16.3.5 order.amend

Motivation: Keeps order/quote lifecycle state synchronized between the initiating client and all interested subscribers. **Published by:** Requesting client process (for example pm-alf-console, pm-admin, pm-viewer, pm-stats, bots, or API gateway) via PUSH :5555

Sent by a gateway to amend the price and/or quantity of a resting order.

Field	Type	Description
order_id	string (UUID)	ID of the order to amend
gateway_id	string	Gateway that owns the order
price	float | absent	New limit price (omit to keep current)

Field	Type	Description
`qty`	integer | absent	New total quantity (omit to keep current)

At least one of `price` or `qty` must be present.

Priority rules:

- Quantity decrease only → priority **preserved** (`arrival_seq` unchanged)
- Price change or quantity increase → priority **lost** (new `arrival_seq` assigned)

Priority is keyed on `arrival_seq`, the engine-assigned monotonic counter — **not** on the order's client-supplied `ts_ns` field (see the `ts_ns` and `arrival_seq` rows under `order.new` above). The `ts_ns` field is never used by the book to determine queue position.

Reply: `order.amended.{GW_ID}` on success, or `order.ack.{GW_ID}` with `accepted=false` on rejection.

16.3.6 `quote.new`

Motivation: Keeps order/quote lifecycle state synchronized between the initiating client and all interested subscribers. **Published by:** Requesting client process (for example pm-alf-console, pm-admin, pm-viewer, pm-stats, bots, or API gateway) via PUSH :5555

Sent by a market-maker gateway to submit or replace a two-sided quote. Role requirements and MM obligation controls are documented in Configuration - Role Privileges.

Field	Type	Description
`gateway_id`	string	Originating gateway identifier
`symbol`	string	Instrument ticker
`quote_id`	string | absent	Optional client-provided quote label
`tick_decimals`	integer	Decimal scale for `bid_price_ticks`/`ask_price_ticks`; 1 tick = `10^-tick_decimals`
`bid_price_ticks`	integer (ticks)	Bid price **in ticks**
`bid_qty`	integer	Bid quantity
`ask_price_ticks`	integer (ticks)	Ask price **in ticks**
`ask_qty`	integer	Ask quantity
`tif`	string	Quote leg time-in-force — same four values as `order.new.tif` (`DAY`, `GTC`, `ATO`, `ATC`); defaults to `DAY`

Replies:

- `quote.ack.{GW_ID}`
- `quote.status.{GW_ID}`

16.3.7 `quote.ack.{GW_ID}`

Motivation: Keeps order/quote lifecycle state synchronized between the initiating client and all interested subscribers. **Published by:** pm-engine via PUB :5556

Acknowledgement of a `quote.new` submission.

Field	Type	Description
`quote_id`	string	Client-provided quote label (echoed from request)
`accepted`	boolean	`true` = accepted; `false` = rejected
`reason`	string	Rejection reason (empty string when accepted)
`bid_order_id`	string (UUID)	Order ID of the bid leg *(present when accepted)*
`ask_order_id`	string (UUID)	Order ID of the ask leg *(present when accepted)*

16.3.8 `quote.status.{GW_ID}`

Motivation: Keeps order/quote lifecycle state synchronized between the initiating client and all interested subscribers. **Published by:** pm-engine via PUB :5556

Published when the quote's lifecycle state changes (e.g. a fill inactivates the quote or a cancel removes it).

Field	Type	Description
`quote_id`	string	Client-provided quote label
`status`	string	New quote state (see values below)
`reason`	string	Additional context (e.g. halt reason); empty when not applicable

Status values:

Value	Meaning
`ACTIVE`	Quote successfully placed on the book (both legs resting)
`INACTIVE_BID_FILLED`	Bid leg filled; ask leg auto-cancelled
`INACTIVE_ASK_FILLED`	Ask leg filled; bid leg auto-cancelled
`CANCELLED`	Quote removed (explicit cancel, kill switch, or halt)

16.3.9 `quote.cancel`

Motivation: Keeps order/quote lifecycle state synchronized between the initiating client and all interested subscribers. **Published by:** Requesting client process (for example pm-alf-console, pm-admin, pm-viewer, pm-stats, bots, or API gateway) via PUSH :5555

Cancel the active quote for one symbol.

Field	Type	Description
`gateway_id`	string	Gateway identifier
`symbol`	string	Instrument ticker

16.4 CALF TCP protocol (`pm-md-gwy`)

`pm-md-gwy` bridges the internal ZMQ bus documented above to a separate, external-facing protocol: a newline-delimited UTF-8 text feed on TCP port `5570` (`MSGTYPE|KEY=VALUE|KEY=VALUE\n` lines), independent of ZMQ topics and payload shapes. It is not just a passthrough of the messages above — it normalises, re-sequences, and reshapes them into CALF's own message types.

Channel	Message type	Wildcard (`SYM=*`)	Baseline `SNAP`?	Carries
`TOP`	`MD`	Yes	Yes	Best bid/ask/last
`TRADE`	`TRADE`	Yes	No	Individual trade prints
`STATE`	`STATE`	Yes	Yes	Session/symbol state transitions
`INDEX`	`IDX`	No	Yes	Index level updates
`DEPTH`	`DEPTH`	No	Yes	Aggregated multi-level order book (Level 2)
`AUCTION`	`AUCTION`	Yes	No	Auction uncross result (equilibrium price/qty, imbalance)
`CB`	`CB`	No	Yes	Circuit-breaker halt/resume detail beyond `STATE`'s coarse transition

Client requests are `HELLO` (authenticate), `SUB`/`UNSUB` (subscribe/cancel), `RESUME` (replay one stream from a known sequence — repeatable, one per stream), `SYMBOLS` (ask which instruments exist), `PING`, and `EXIT`. Gateway replies include `WELCOME`, a baseline `SNAP` per new stream on the five channels that have one, the seven message types above, periodic `HB` heartbeats, and `ERR` on protocol/subscription violations.

Two behaviours are easy to get wrong from the table alone:

- **`MD` is a delta, and an empty value is meaningful.** Only changed fields are sent; an omitted field means *unchanged*. An explicitly empty `BID=` or `ASK=` means that book side is now **empty** and the client must discard the price rather than keep the last one it saw.

- **`RESUME` is per stream.** `LASTSEQ` describes one `(CH, SYM)` position, so a reconnecting client sends one `RESUME` per stream it was following. It is not a flag on `HELLO`.
- **Ask for the symbol universe; do not wait for it.** `WELCOME|SYMBOLS=` is optional and omitted entirely by a gateway started without a readable engine config. Send `SYMBOLS` after the handshake, and read its `COUNT` — an empty universe omits the list rather than sending it empty.
- **Read `REF=` before you format a price.** It carries each symbol's display precision as `SYM:DEC` tuples, on both `WELCOME` and the `SYMBOLS` reply, and is the only route a market data client has to that value. A gateway that predates the field omits it entirely, in which case assume `2` decimals — knowingly, because getting this wrong rounds every price the client shows.

The full protocol — every field table, the `WELCOME`/`SNAP` handshake, sequence-gap detection and `RESUME` recovery, subscription limits, and the complete error-code table — is maintained in one place to avoid drift: Market Data Feed (CALF), with the normative wire-format specification in the CALF Protocol Reference.

16.5 See also

In this book:

- Drop Copy — the separate :5557 socket for fill-only event feeds
- CALF TCP Protocol — external market-data feed message types

Participant Guide:

- The Trader Console — how participants receive fill, book, and risk events

Operator's Guide:

- The Admin Console and Exchange Commands — `ExchangeCommandClient` methods and their underlying message topics
- Risk Controls — `risk.*` message payloads in detail

Reference Manual:

- Processes, Environment, and Ports — which process subscribes to which topic prefix

16.6 Topic index

Every topic in the system, and which process puts it on the wire.

Topic	Family	Published by
`admin.action.{gateway_id}`	`admin`	`engine`
`auction.indicative.{symbol}`	`auction`	`engine`
`auction.result.{symbol}`	`auction`	`engine`
`book.snapshot_request`	`book`	`stats`, `gateway`
`book.{symbol}`	`book`	`engine`
`circuit_breaker.extend.{symbol}`	`circuit_breaker`	`engine`
`circuit_breaker.halt.{symbol}`	`circuit_breaker`	`engine`
`circuit_breaker.resume.{symbol}`	`circuit_breaker`	`engine`

Topic	Family	Published by
combo.ack.{gateway_id}	structure	engine
combo.cancel_ack.{gateway_id}	structure	engine
combo.status.{gateway_id}	structure	engine
depth.{symbol}	book	engine
drop_copy.event.{gateway_id}	drop_copy	engine
drop_copy.replay.{recipient_id}	drop_copy	engine
engine.trace.{symbol}	order	engine
execution_report (no bus topic)	order	gateway
health.beacon	health	health_reporter
health.event	health	monitor
health.snapshot	health	monitor
index.constituent_change	index	admin
index.constituent_change_ack.{gateway_id}	index	index
index.corp_action	index	admin
index.corp_action_ack.{gateway_id}	index	index
index.error.{gateway_id}	index	index
index.history.{gateway_id}	index	index
index.history_request	index	admin, api_gateway, gateway
index.rebalance	index	api_gateway
index.rebalance_ack.{gateway_id}	index	index
index.update	index	index
log.backfill.{sub_id}	log	log_server
log.backfill_request	log	log_client
log.error.{sub_id}	log	log_server
log.event.{sub_id}	log	log_server
log.lease_expired.{sub_id}	log	log_server
log.notify.{sub_id}	log	log_server
log.renew	log	log_client
log.renew_ack.{sub_id}	log	log_server
log.server_state	log	log_server
log.status.{sub_id}	log	log_server
log.status_request	log	log_client
log.subscribe	log	log_client
log.subscribe_ack.{sub_id}	log	log_server
log.unsubscribe	log	log_client

Topic	Family	Published by
log.unsubscribe_ack.{sub_id}	log	log_server
news.event	news	market_sim
oco.ack.{gateway_id}	structure	engine
oco.cancel_ack.{gateway_id}	structure	engine
oco.cancelled.{gateway_id}	structure	engine
order.ack.{gateway_id}	order	engine
order.amend	order	gateway
order.amended.{gateway_id}	order	engine
order.cancel	order	gateway
order.cancelled.{gateway_id}	order	engine
order.combo	order	gateway
order.combo_cancel	order	gateway
order.expired.{gateway_id}	order	engine
order.fill.{gateway_id}	order	engine
order.new	order	gateway
order.oco	order	gateway
order.oco_cancel	order	gateway
order.orders.{gateway_id}	order	engine
order.orders_request	order	admin, api_gateway, gateway
order.price_level_orders.{gateway_id}	order	engine
order.price_level_orders_request	order	admin
order.triggered.{gateway_id}	order	engine
quote.ack.{gateway_id}	quote	engine
quote.cancel	quote	admin, api_gateway, gateway
quote.cancel_ack.{gateway_id}	quote	engine
quote.new	quote	gateway
quote.status.{gateway_id}	quote	engine
risk.band.{symbol}	risk	engine
risk.cancel_symbol	risk	admin, api_gateway
risk.cancel_symbol_ack.{gateway_id}	risk	engine
risk.circuit_breaker_halt_all	risk	admin
risk.circuit_breaker_halt_all_ack.{gateway_id}	risk	engine
risk.circuit_breaker_resume_all	risk	admin
risk.circuit_breaker_resume_all_ack.{gateway_id}	risk	engine
risk.force_uncross	risk	admin
risk.force_uncross_ack.{gateway_id}	risk	engine

Topic	Family	Published by
risk.kill_switch	risk	admin, api_gateway, gateway
risk.kill_switch_ack.{gateway_id}	risk	engine
risk.kill_switch_gateway	risk	api_gateway
risk.kill_switch_gateway_ack.{gateway_id}	risk	engine
risk.kill_switch_global	risk	api_gateway
risk.kill_switch_global_ack.{gateway_id}	risk	engine
risk.symbol_halt	risk	admin, api_gateway
risk.symbol_halt_ack.{gateway_id}	risk	engine
risk.symbol_resume	risk	admin, api_gateway
risk.symbol_resume_ack.{gateway_id}	risk	engine
session.next	session	scheduler
session.state	session	engine
session.transition	session	scheduler
session.transition_ack.{gateway_id}	session	engine
sim.command	sim	admin
sim.command_ack.{gateway_id}	sim	market_sim
sim.state	sim	market_sim
sim.value	sim	market_sim
stop_triggered (no bus topic)	order	gateway
system.book_checkpoint.{symbol}	system	engine
system.diagnostic	system	engine
system.eod	system	engine
system.gateway_auth.{gateway_id}	system	engine
system.gateway_bye.{gateway_id}	system	engine
system.gateway_connect	system	admin, api_gateway, gateway
system.gateway_disconnect	system	admin, api_gateway, gateway
system.gateway_heartbeat	system	api_gateway, gateway
system.gateway_reject	system	api_gateway, gateway
system.gateways.{gateway_id}	system	engine
system.gateways_request	system	admin, api_gateway
system.halt_status.{gateway_id}	system	engine
system.halt_status_request	system	api_gateway
system.position_request	system	gateway
system.position_snapshot.{gateway_id}	system	engine
system.quote_bootstrap.{gateway_id}	system	engine
system.quote_bootstrap_request	system	admin, api_gateway, gateway

Topic	Family	Published by
system.quote_legs.{gateway_id}	system	engine
system.quote_legs_request	system	api_gateway, gateway
system.quote_seed_skipped	system	engine
system.recovery_item	system	engine
system.reference.{gateway_id}	system	engine
system.reference_reload	system	api_gateway
system.reference_reload_ack.{gateway_id}	system	engine
system.reference_request	system	api_gateway
system.risk_state.{gateway_id}	system	engine
system.risk_state_request	system	api_gateway
system.session_schedule.{gateway_id}	system	engine
system.session_schedule_request	system	admin, api_gateway
system.session_state_request	system	admin, api_gateway, gateway, scheduler
system.session_status.{gateway_id}	system	engine
system.startup	system	engine
system.startup_recovery	system	engine
system.symbols.{gateway_id}	system	engine
system.symbols_request	system	admin, api_gateway, gateway, stats
system.volume.{gateway_id}	system	engine
system.volume_request	system	admin
trade.executed	trade	engine

16.7 Family `admin`

16.7.1 Record types

`AdminActionScope`

What one admin command acted on, and what it did. Every field is optional because every action uses a different subset – see the family header on why that is a stated limitation rather than a variant type. The name is inherited from the wire key rather than chosen: three of the seven fields are outcome counts rather than scope, which makes "AdminActionScope" a slightly generous reading of its own contents. It was kept because renaming the key is a wire change for every `/admin/monitor` client and buys nothing a reader can use.

Field	Type	Presence	Rules	Description
`symbol`	`string`	omitted when unset	max_len 16	The instrument, when the command named one. `kill_switch.self` used to emit this as an explicit `null` when unscoped while its siblings omitted the key – two spellings of one absence, and this is now the second.
`target_gateway_id`	`string`	omitted when unset	max_len 32	The participant acted upon, for `kill_switch.gateway`.
`quotes_only`	`bool`	omitted when unset	—	For `kill_switch.gateway`: only the target's quotes were cancelled.
`level`	`string`	omitted when unset	max_len 32	The circuit-breaker rung, for `circuit_breaker.trigger`. Not an enum: the ladder is per-symbol configuration, so the value set differs per deployment. Bounded to match `circuit_breaker.halt.leve` which carries the same name onward.

Field	Type	Presence	Rules	Description
`note`	`string`	omitted when empty	max_len 256	The operator's free-text reason, when one was supplied. Regime 4 to match `risk.kill_switch`'s own `note`, which is where this value arrives from – a field that omits on one message and emits `""` on the next would be two answers to one question.
`cancelled_orders`	`int`	omitted when unset	ge 0, unit `dimensionless`	Outcome, on accepted kill-switch actions only.
`cancelled_order_ids`	list of `string`	omitted when empty	—	AR-0.5: the order ids counted in cancelled_orders, mirroring risk.kill_switch_ack (and friends) — see risk.yaml for why this is the right key even for the quote-cancellation counterpart below.
`cancelled_quotes`	`int`	omitted when unset	ge 0, unit `dimensionless`	Outcome, on accepted kill-switch actions only.
`cancelled_quote_order_ids`	list of `string`	omitted when empty	—	AR-0.5: the order ids counted in cancelled_quotes.
`affected_gateways`	`int`	omitted when unset	ge 0, unit `dimensionless`	Outcome, on an accepted `kill_switch.global` only.

Field	Type	Presence	Rules	Description
`affected_gateway_ids`	list of `string`	omitted when empty	—	AR-0.5: the gateway ids counted in affected_gateways.

16.7.2 `admin.action.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to the admin monitor: one admin-gated command ran, who ran it, what it acted on, and whether it was accepted.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	The ADMIN caller. Topic-only; the body says the same thing as `initiator_gateway_id`.
`command_id`	`string`	required	max_len 64	
`initiator_gateway_id`	`string`	required	max_len 32	

Field	Type	Presence	Rules	Description
`action`	enum: `kill_switch.self`, `kill_switch.gateway`, `kill_switch.global`, `kill_switch.symbol`, `circuit_breaker.trigger`, `circuit_breaker.resume`, `auction.reopen`	required	—	Which command ran. The seven the engine publishes, enumerated rather than left a free string: an eighth admin command that forgets to declare itself here fails loudly at its first invocation, which is better than appearing in the monitor as a value no client renders. The values look like topics and are not: `circuit_breaker.trigger` is the action behind `risk.symbol_halt`, and there is no `circuit_breaker.trigger` topic anywhere. `auction.reopen` is the action behind `risk.force_uncross`.
`scope`	`AdminActionScope`	required	—	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	Why it was rejected; “ ” on an accepted action.

Note

This is the one topic in the system addressed to a gateway that is **not** for that gateway.
The suffix names the ADMIN caller so a monitor can filter by operator, but the event must never reach that caller's own private trading stream – `EngineClient._handle_event` checks the prefix before the private/market-data split for exactly that reason, and `ADMIN_ACTION_PREFIX` is deliberately absent from `PRIVATE_PREFIXES`.
`initiator_gateway_id` repeats the topic suffix in the body.
That is redundant on the live wire and load-bearing off it: an event stored, forwarded or rendered without its topic still says who ran the command.

Publishing is a no-op without a `command_id` — with nothing to correlate against, a monitor record is an entry no client can tie to a request.
See also: `risk.kill_switch`, `risk.symbol_halt`, `circuit_breaker.halt.{SYMBOL}`

16.8 Family `auction`

16.8.1 `auction.indicative.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all: where one symbol would uncross if the call phase ended now. Published repeatedly while an opening or closing auction collects orders.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`phase`	enum: `OPENING_AUCTION`, `CLOSING_AUCTION`	required	—	Which call phase is running. These are the only two states `models/session.py::is_auc` admits, and the producer returns early for every other one.
`eq_price`	`float`	`null` when unset	gt 0, unit `display_price`	Indicative equilibrium price, or null if the book would not cross.
`eq_qty`	`int`	required	ge 0, unit `shares`	Quantity that would execute. Zero is a true reading and is always emitted, unlike `eq_price`, which has no zero.
`imbalance_side`	enum: `BUY`, `SELL`	omitted when unset	—	Which side would be left unfilled. Absent when the book is balanced at the indicative price.

Field	Type	Presence	Rules	Description
`imbalance_qty`	`int`	required	ge 0, unit `shares`	Surplus on `imbalance_side`; zero when balanced.
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line can turn it into money without guessing.
`eq_price_ticks`	`ticks`	omitted when unset	unit `ticks`	`eq_price` in engine ticks. Canonical; the float beside it is for display. Scale: the symbol's tick_decimals.

Note

The difference from `auction.result` is tense.
That one reports what happened; this one reports what would happen if the phase ended now, and a client must not mistake the second for the first – which is why `md_gateway` projects it to a CALF `INDIC` rather than an `AUCTION`.
`eq_price` is null when the book would not cross at all.
That is a real and informative state during a call phase – nothing would trade yet – and is not the same as a price of zero, so it is a null rather than an omission.
Field names are shared with `circuit_breaker.extend`'s indicative deliberately.
A reopening auction and a scheduled one are the same mechanism, and a client that learned to read one should not have to learn the other.
See also: `auction.result.{SYMBOL}`, `circuit_breaker.extend.{SYMBOL}`

16.8.2 `auction.result.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all: one symbol's uncross has completed. Published for every uncross, including the ones that printed nothing.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`eq_price`	`float`	`null` when unset	gt 0, unit `display_price`	Uncross price, or null when there was no crossable interest.
`eq_qty`	`int`	required	ge 0, unit `shares`	Quantity executed; zero when nothing crossed.
`trades_count`	`int`	required	ge 0, unit `dimensionless`	How many trades the uncross printed.
`imbalance_side`	enum: `BUY`, `SELL`	omitted when unset	—	Which side was left unfilled. Absent when the book was balanced at the uncross price.
`imbalance_qty`	`int`	required	ge 0, unit `shares`	Surplus on `imbalance_side`; zero when balanced.
`reason`	enum: `SCHEDULED`, `REOPEN`, `RECOVERY`, `BACKSTOP`, `ADMIN_MANUAL`	required	—	Which of the five uncross paths produced this event.
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line can turn it into money without guessing.

Field	Type	Presence	Rules	Description
`eq_price_ticks`	`ticks`	omitted when unset	unit `ticks`	`eq_price` in engine ticks. Canonical; the float beside it is for display. Scale: the symbol's tick_decimals.
`auction_id`	`string`	omitted when unset	max_len 32	This uncross, `AUC-<run_seq>-<n>`: every print it made carries it as `trade.executed.auction_id`.

Note

`reason` says which uncross this was, because the five are otherwise indistinguishable to a consumer and a client cannot tell a circuit breaker reopening from the closing one: SCHEDULED - leaving an auction or other non-matching session phase REOPEN - a halted symbol reopening at the end of its halt RECOVERY - restored GTC orders uncrossed at engine startup BACKSTOP - the closing backstop forcing a still-halted symbol to reopen, printing at the corridor boundary rather than at the outlying equilibrium ADMIN_MANUAL - an operator force-uncrossed one symbol via `pm-admin-cli reopen`, optionally asserting the print price when no natural equilibrium exists There is no persistent state to snapshot here, unlike TOP or DEPTH: every event is forwarded as its own independent CALF event.
See also: `auction.indicative.{SYMBOL}`, `trade.executed`

16.9 Family `book`

16.9.1 Record types

`BookLevel`

One aggregated price level. Iceberg orders contribute only their displayed quantity, so `qty` is what a viewer should show rather than what is actually resting.

Field	Type	Presence	Rules	Description
`price`	`float`	required	unit `display_price`	
`qty`	`int`	required	unit `shares`	Aggregate visible size at this level.
`count`	`int`	required	unit `dimensionless`	How many orders make up the level.

`RecentTrade`

One entry of the book's trade tape. A trimmed view of the public trade.executed print - the last five, carried with the snapshot so a viewer that has just subscribed has some history to draw.

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 64	
`symbol`	`string`	required	max_len 16	
`buy_order_id`	`string`	required	max_len 64	
`sell_order_id`	`string`	required	max_len 64	
`buy_gateway_id`	`string`	required	max_len 32	
`sell_gateway_id`	`string`	required	max_len 32	
`price`	`float`	required	unit `display_price`	
`quantity`	`int`	required	unit `shares`	
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	Epoch nanoseconds, carried through unscaled – identical to the trade.executed print this row mirrors.

16.9.2 `book.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Broadcast an aggregated view of one instrument's order book, on a timer. What every viewer and terminal renders.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`tick_decimals`	`int`	required	unit `dimensionless`	The tick scale the display prices here were produced at. Subscribers that store prices exactly need it to convert back to integer ticks; without it they must guess, and guessing 2 for a 4-decimal symbol rounds the price away.

Field	Type	Presence	Rules	Description
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	When this snapshot was produced (AR-0.4). Previously book.{symbol} carried no time field at all, so nothing could establish whether a snapshot reflects a given trade; ordering against trade.executed.ts_ns (and recent_trades[].ts_ns, which this mirrors) was impossible. Monotonic per edumatcher.models.clock.now_ns().
`bids`	list of `BookLevel`	required	—	Descending by price.
`asks`	list of `BookLevel`	required	—	Ascending by price.
`last_price`	`float`	`null` when unset	unit `display_price`	Null until the instrument has traded.
`last_qty`	`int`	`null` when unset	unit `shares`	
`last_buy_price`	`float`	`null` when unset	unit `display_price`	
`last_sell_price`	`float`	`null` when unset	unit `display_price`	
`recent_trades`	list of `RecentTrade`	required	—	The last five prints, oldest first.

Note

Every key is always present; the four last_* fields carry null on a book that has not traded. The payload is OrderBook.snapshot() exactly.

See also: `depth.{SYMBOL}`, `book.snapshot_request`

16.9.3 `book.snapshot_request`

Published by: `stats`, `gateway`

Transport: `engine_push`

Since: 1.0

Ask the engine to publish one symbol's book immediately.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	

See also: `book.{SYMBOL}`

16.9.4 `depth.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Book-depth metrics within a tolerance band of the last trade: how much size sits nearby, which way it leans, and what it costs to move.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	When this depth snapshot was produced (AR-0.4). Previously depth.{symbol} carried no time field at all. Monotonic per edu-matcher.models.clock.now_
`tick_decimals`	`int`	required	ge 0, le 8, unit `dimensionless`	Decimal scale for the `_ticks` fields below; 1 tick = 10^-tick_decimals. Same field, same name and same meaning as `book.tick_decimals`.

Field	Type	Presence	Rules	Description
`mid_price_ticks`	`ticks`	required	unit `ticks`	The last trade price, in ticks; the band is centred here.
`mid_price`	`float`	required	unit `display_price`	
`tolerance_ticks`	`ticks`	required	unit `ticks`	Half-width of the band, in ticks.
`bid_depth`	`int`	required	unit `shares`	Total resting size between mid - tolerance and mid.
`ask_depth`	`int`	required	unit `shares`	Total resting size between mid and mid + tolerance.
`imbalance`	`float`	required	ge -1, le 1, unit `dimensionless`	(bid - ask) / total, in [-1, 1]; positive means more bids.
`microprice`	`float`	required	unit `display_price`	Imbalance-weighted mid; falls back to mid_price.
`cost_to_move`	`float`	required	unit `money`	Display notional a buyer must spend to sweep every ask in the band. Summed in ticks and converted once, not per level.

Note

Not published at all for a book with no last trade - depth_snapshot returns an empty dict and the engine skips it - so every field here is required rather than nullable.
See also: `book.{SYMBOL}`

16.10 Family `circuit_breaker`

16.10.1 `circuit_breaker.halt.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all: one symbol has stopped trading. New orders rest rather than match until the halt ends, and every resting quote on the symbol has already been cancelled by the time this is published.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`trigger_price`	`float`	`null` when unset	gt 0, unit `display_price`	The trade price that breached the level. Null on an ADMIN halt.
`reference_price`	`float`	`null` when unset	gt 0, unit `display_price`	The price the breach was measured against. Null on an ADMIN halt.
`resume_at_ns`	`int`	`null` when unset	unit `epoch_nanos`	When the current call phase ends. Null means indefinite: the halt lasts until an operator resumes it. ACE moves this on every extension, so a consumer that ignores `circuit_breaker.extend` will hold a value that has already passed.
`halt_source`	enum: `CB`, `ADMIN`	omitted when unset	—	What put the symbol into the halt. Mirrors `CircuitBreakerState.halt_` which is `None` only while no halt is in effect – so the key is present on every halt event.

Field	Type	Presence	Rules	Description
`level`	`string`	omitted when unset	max_len 32	Which rung of the ladder fired – a name from the symbol's `circuit_breaker.levels` config – or `ADMIN_ALL` / `ADMIN_SYMBOL` for an operator halt. Not an enum: the ladder is configuration, so the value set differs per deployment.
`corridor_low`	`float`	omitted when unset	gt 0, unit `display_price`	Lower bound of the ACE reopening corridor. Absent when the halt has no corridor: either ACE is disabled or the halt began with no reference price to centre one on.
`corridor_high`	`float`	omitted when unset	gt 0, unit `display_price`	Upper bound of the ACE reopening corridor. Absent with the low.
`expansion`	`int`	omitted when unset	ge 0, unit `dimensionless`	Rungs of the expansion ladder consumed so far; 0 in the initial call phase. Absent whenever the corridor is, because a halt that cannot have a corridor can never widen one.

Note

`halt_source` says what caused the halt, not how it will end: every halt ends in a reopening auction call, because LIMIT orders accumulate freely while a symbol is halted and resuming without an

uncross would start continuous trading on a crossed book.
`trigger_price` and `reference_price` are the price that fired the breaker and the price it was measured against.
Both are null on an ADMIN halt, which fires on an operator's decision rather than on a price, and both may be null on a price-triggered halt that had no reference to latch.
`resume_at_ns` is null for an indefinite halt – `halt_all` and a per-symbol halt named without a level both produce one, and it lasts until an explicit resume.
See also: `circuit_breaker.extend.{SYMBOL}`, `circuit_breaker.resume.{SYMBOL}`

16.10.2 `circuit_breaker.extend.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all: the call phase ended with the indicative price outside the corridor, so the symbol stays halted, the corridor widens by one rung and a fresh call phase begins.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`indicative_price`	`float`	required	gt 0, unit `display_price`	Where the symbol would have reopened, outside the corridor.
`indicative_qty`	`int`	required	gt 0, unit `shares`	Quantity that would have executed. Always above zero: the producer only extends when the indicative uncross would have traded.
`imbalance_side`	enum: `BUY`, `SELL`	omitted when unset	—	Which side is unfilled at the indicative price. Absent when the book is balanced there.
`resume_at_ns`	`int`	required	unit `epoch_nanos`	End of the new call phase. Always set; `extend()` computes it.

Field	Type	Presence	Rules	Description
`corridor_low`	`float`	required	gt 0, unit `display_price`	Lower bound of the widened corridor.
`corridor_high`	`float`	required	gt 0, unit `display_price`	Upper bound of the widened corridor.
`expansion`	`int`	required	gt 0, unit `dimensionless`	Rungs consumed after this widening – at least 1, since the event is published by the widening itself.

Note

The symbol's state does not change here – an extension is a continuation of the same halt – so `md_gateway` deliberately does not re-emit a STATE event for it, only the moved corridor and resume time.
The corridor fields are the corridor *after* widening, and unlike on `circuit_breaker.halt` they are always present: this event can only be produced on a path that has just asserted a corridor exists. `indicative_price` and `indicative_qty` are the imbalance indicator a real venue disseminates during a reopening.
They are what lets a participant supply the offsetting interest that resolves the halt, which only works while there is still time to act.
See also: `circuit_breaker.halt.{SYMBOL}`

16.10.3 `circuit_breaker.resume.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all: the symbol is trading again. It rejoins whatever the exchange is currently doing rather than returning to continuous trading – a halt that expires near the close resumes into CLOSING_AUCTION or CLOSED.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`halt_source`	enum: `CB`, `ADMIN`	omitted when unset	—	What had put the symbol into the halt that just ended.

Field	Type	Presence	Rules	Description
`reason`	`string`	omitted when empty	max_len 64	Why the halt ended, when that is not simply "its call phase expired". Only the closing backstop sets it, to CLOSING_BACKSTOP.
`clamped`	`bool`	omitted when unset	—	True when the print price was forced to the corridor boundary instead of the equilibrium. A client showing a clamped price as a discovered one would mislead. Absent on the three ordinary resumes, where no price is imposed at all – which is a different statement from `false`, and worth keeping distinct.
`print_price`	`float`	omitted when unset	gt 0, unit `display_price`	The price the backstop uncross printed at. Absent when there was no crossing interest to print, and on the three ordinary resumes.

Note

Four producers, two shapes.
Three of them – ACE expiry, ADMIN resume-all and ADMIN per-symbol resume – send `symbol` and `halt_source` alone.
The closing backstop sends three fields more, because it is the one resume where the reopening price was imposed rather than discovered: it prints *at* the corridor boundary for a symbol that

could not reopen inside it, which can leave the book crossed by design.
The three extra fields are regime 3 and 4 rather than always-present nulls, so the three ordinary producers keep the two-key payload they have always sent.
`normalise_cb_resume` reads each through a falsy guard, so an absent key and an empty one are the same event to it.
See also: `circuit_breaker.halt.{SYMBOL}`

16.11 Family `drop_copy`

16.11.1 `drop_copy.event.{gateway_id}`

Published by: `engine`

Transport: `drop_copy_pub`

Since: 1.0

Engine to a participant's clearing broker, prime broker or in-house risk system: one execution, as it happens.

Field	Type	Presence	Rules	Description
`seq`	`int`	required	gt 0, unit `dimensionless`	Monotone counter for this gateway's stream, starting at 1 so that 0 can mean "no events yet". Independent of other gateways and resets when the engine restarts. A recipient detects loss from a gap and a duplicate from a repeat within this stream.
`timestamp`	`int`	required	unit `epoch_nanos`	When the engine published the event (`models/clock.py::now_ns`).

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	The participant whose order executed. Carried in the body as well as in the topic, because `drop_copy.replay` names the *recipient* in its topic instead and a replayed event would otherwise not say whose fill it was.
`event_type`	enum: `order.fill`	required	—	One value today. An enum rather than a free string so that a second event type is a spec change with a regenerated binding, rather than a new dict key no reader knows about – `DropCopyPublisher`'s own docstring promised "every fill and cancel" while only fills existed, which is how the gap went unnoticed. Section 27.3.
`order_id`	`string`	required	max_len 64	The resting or aggressing order this execution belongs to.

Field	Type	Presence	Rules	Description
`trade_ids`	list of `string`	required	min_items 1	Public trade.executed id(s) for this execution. One ID today; kept as a list so consumers can use the same representation as coalesced order.fill messages when an order sweeps multiple price levels.
`symbol`	`string`	required	max_len 16	
`fill_qty`	`int`	required	gt 0, unit `shares`	
`fill_price`	`float`	required	gt 0, unit `display_price`	Display money, not ticks – converted once in `_publish_trade`.
`liquidity_flag`	enum: `MAKER`, `TAKER`	required	—	Derived from the trade's aggressor side: the aggressor is the TAKER and the resting side the MAKER. Exactly one of the two events a trade produces is TAKER.

Note

Fed from the engine's single trade-publication path, so it covers every fill-producing flow – new orders, quotes, combo legs, OCO legs, auction uncrosses, stop cascades and amend-rematches. It was once wired only into the new-order loop, and quote and auction fills were invisible to clearing as a result.

This is a *derived copy* of `order.fill.{GW_ID}`, not the same message: it is sequenced, buffered for replay, carries the liquidity flag, and travels on a socket the trading gateway does not subscribe to.

The two are deliberately allowed to differ.

Every trade produces two of these, one per counterparty, so a recipient watching both sides of a matched pair sees the same execution twice under different `gateway_id`s.

See also: `drop_copy.replay.{RECIPIENT_ID}`, `order.fill.{GW_ID}`

16.11.2 `drop_copy.replay.{recipient_id}`

Published by: `engine`

Transport: `drop_copy_pub`

Since: 1.0

Engine to one named recipient: buffered events re-published on request, so a participant that reconnects mid-session can close its sequence gap.

Field	Type	Presence	Rules	Description
`recipient_id`	`string`	required	max_len 32	Who asked for the replay. Topic-only — deliberately not in the body, which is byte-identical to the live event.
`seq`	`int`	required	gt 0, unit `dimensionless`	Original sequence number in this gateway's live stream.
`timestamp`	`int`	required	unit `epoch_nanos`	
`gateway_id`	`string`	required	max_len 32	Whose fill this was — not the recipient the topic names.
`event_type`	enum: `order.fill`	required	—	
`order_id`	`string`	required	max_len 64	
`trade_ids`	list of `string`	required	min_items 1	
`symbol`	`string`	required	max_len 16	
`fill_qty`	`int`	required	gt 0, unit `shares`	
`fill_price`	`float`	required	gt 0, unit `display_price`	
`liquidity_flag`	enum: `MAKER`, `TAKER`	required	—	

Note

The body is byte-identical to the live event, including the original `seq` and `timestamp` – a replayed fill is the same fill, not a new one.
Only the topic differs, and it names the *recipient* rather than the gateway, so two simultaneous replays do not interleave.
`recipient_id` is therefore not the same thing as `gateway_id`, which is why the body keeps carrying the latter.
The in-process replay method takes `gateway_id` as a required argument and replays only that gateway's events.

There is no request message.
`DropCopyPublisher.replay()` is in-process only, callable from the engine and reachable by no protocol – the module docstring described a `drop_copy.replay_request` that was never built. Section 27.3.
See also: `drop_copy.event.{GW_ID}`

16.12 Family `health`

16.12.1 Record types

`MetricSample`

One application metric of a process at the time of its beacon. A list entry rather than a map value, so the dimension (`label`) is a field and a metric with no dimension simply carries the empty label.

Field	Type	Presence	Rules	Description
`name`	`string`	required	max_len 64, min_len 1	Dotted name, `<family>.<metric>`, e.g. `engine.orders_in`.
`kind`	enum: `COUNTER`, `GAUGE`	required	—	A COUNTER is monotonic since process start; a GAUGE is a level.
`label`	`string`	defaults to `''`	max_len 32	The single dimension: a gateway id, a symbol, a component, a reason. Bounded by configuration or by a closed set in code, never by what a client sends.
`value`	`float`	required	unit `dimensionless`	

`FileSize`

The size of one data file pm-monitor watches under the data directory.

Field	Type	Presence	Rules	Description
`name`	`string`	required	max_len 32	Logical file name, e.g. `log.db`, `audit`, `metrics.db`.

Field	Type	Presence	Rules	Description
`size_bytes`	`int`	required	ge 0, unit `bytes`	0 when the file does not exist.

`ExchangeStatus`

The one-line answer to "is the exchange OK?" (design section 10).

Field	Type	Presence	Rules	Description
`status`	enum: `OPERATIONAL`, `DEGRADED`, `IMPAIRED`, `DOWN`	required	—	DOWN when no pm-engine is running: it is DEAD, SILENT, HUNG, MISSING or STOPPED, or pm-monitor has never heard of one. IMPAIRED when any other expected exchange process is DEAD, SILENT, HUNG or MISSING (or STOPPED while the active emo profile expects it). DEGRADED when any exchange process is DEGRADED. Otherwise OPERATIONAL. Participant bots never count.
`session_state`	`string`	required	max_len 32	The engine's session state name; empty before the engine reports one.
`trading_date`	`string`	required	max_len 10	YYYY-MM-DD in the session timezone; what "today" means below.

Field	Type	Presence	Rules	Description
`session_timezone`	`string`	required	max_len 64	IANA name from engine.locale.session_timezone.

`HostStats`

The host pm-monitor runs on (design section 8.1).

Field	Type	Presence	Rules	Description
`cpu_pct`	`float`	required	ge 0, unit `percent`	
`cpu_per_core`	list of `float`	required	—	Percent per logical CPU, in psutil order.
`load1`	`float`	required	unit `dimensionless`	
`load5`	`float`	required	unit `dimensionless`	
`load15`	`float`	required	unit `dimensionless`	
`mem_total_bytes`	`int`	required	ge 0, unit `bytes`	
`mem_used_bytes`	`int`	required	ge 0, unit `bytes`	
`mem_available_bytes`	`int`	required	ge 0, unit `bytes`	
`swap_used_bytes`	`int`	required	ge 0, unit `bytes`	
`disk_free_bytes`	`int`	required	ge 0, unit `bytes`	On the filesystem holding the data directory.
`disk_total_bytes`	`int`	required	ge 0, unit `bytes`	
`files`	list of `FileSize`	required	—	

`Sparkline`

The last samples of a process, oldest first, for inline sparklines.

Field	Type	Presence	Rules	Description
`rss`	list of `int`	required	max_items 60	RSS in bytes, one per beacon.
`cpu`	list of `float`	required	max_items 60	CPU percent of one core, one per beacon.

`ProcessHealth`

One process as pm-monitor sees it: a process that beacons, or one the active emo profile expects and that has never beaconed (MISSING, with zero figures and an empty host).

Field	Type	Presence	Rules	Description
`key`	`string`	required	max_len 200, min_len 1	`process/instance@host`, or `process@host` without an instance; a MISSING process has no host and no `@`. Stable identity for routes and history.
`process`	`string`	required	max_len 32	
`instance`	`string`	`null` when unset	max_len 32	
`host`	`string`	required	max_len 128	
`group`	enum: `EXCHANGE`, `BOT`	required	—	BOT for participant bots (pm-mm-bot, pm-ai-swarm), which never affect the exchange status.
`pid`	`int`	required	ge 0, unit `dimensionless`	0 when the process has never beaconed.
`state`	enum: `HEALTHY`, `DEGRADED`, `HUNG`, `SILENT`, `DEAD`, `MISSING`, `STOPPED`	required	—	Design section 10; first matching rule wins.
`restarted_recently`	`bool`	required	—	boot_id changed within the last 10 minutes.
`starting`	`bool`	defaults to `False`	—	The latest beacon said the main loop has not ticked yet. The state stays HEALTHY through a start-up grace, so this is what tells a process still starting from one that is running.
`uptime_sec`	`float`	required	ge 0, unit `duration_seconds`	
`rss_bytes`	`int`	required	ge 0, unit `bytes`	

Field	Type	Presence	Rules	Description
`cpu_pct`	`float`	required	ge 0, unit `percent`	Percent of one core; may exceed 100 for a multi-threaded process.
`threads`	`int`	required	ge 0, unit `dimensionless`	
`fds`	`int`	`null` when unset	unit `dimensionless`	null where the OS has no file-descriptor count.
`loop_age_ms`	`float`	required	ge 0, unit `duration_millis`	
`loop_lag_ms`	`float`	required	ge 0, unit `duration_millis`	
`last_beacon_age_ms`	`float`	required	ge 0, unit `duration_millis`	Measured with pm-monitor's receive clock.
`spark`	`Sparkline`	required	—	
`metrics`	list of `MetricSample`	required	—	The application metrics of the latest beacon, as sent – except pm-engine's labelled samples (one per participant or symbol, thousands at 300 symbols): the trading, participants and books blocks carry those derived, and metrics.db keeps them. Its unlabelled samples and engine.diagnostics are kept.

`RateStats`

A counter turned into the figures of design section 9.4. Every increment is reset-corrected, so a process restart neither loses nor double-counts.

Field	Type	Presence	Rules	Description
`today`	`int`	required	ge 0, unit `dimensionless`	Sum of increments since the start of the trading date.
`per_sec_now`	`float`	required	ge 0, unit `dimensionless`	Increments over the last 60 s divided by 60.
`avg_per_min`	`float`	required	ge 0, unit `dimensionless`	Increments over the last 60 min divided by 60.
`avg_per_hour`	`float`	required	ge 0, unit `dimensionless`	today divided by hours since the session opened (else midnight).
`avg_per_day`	`float`	`null` when unset	unit `dimensionless`	Mean daily total over the last 7 trading days with data; null before the first full day.

`RejectCount`

Rejects of one reason today: a RejectCode, or a label the engine gives a reject known only by its text (MM_MAX_SPREAD, MM_MIN_QTY, UNKNOWN).

Field	Type	Presence	Rules	Description
`code`	`string`	required	max_len 32, min_len 1	
`count`	`int`	required	ge 0, unit `dimensionless`	

`TradingStats`

Exchange-wide trading activity, summed over gateways and symbols.

Field	Type	Presence	Rules	Description
`orders`	`RateStats`	required	—	engine.orders_in: inbound new orders, including those then rejected.
`trades`	`RateStats`	required	—	
`traded_value_today`	`float`	required	ge 0, unit `money`	

Field	Type	Presence	Rules	Description
`reject_pct_5m`	`float`	required	ge 0, unit `percent`	Rejected acks over inbound requests (new orders, combos, OCOs, cancels, amends, quotes) in the last 5 minutes, the same set engine.rejects counts; 0 when none.
`handle_p50_us`	`float`	required	ge 0, unit `duration_micros`	engine.handle_us_p50: order-entry messages, sampled 1 in 16, over the last minute.
`handle_p99_us`	`float`	required	ge 0, unit `duration_micros`	engine.handle_us_p99, over the same samples.
`handle_p99_ex_gc_us`	`float`	required	ge 0, unit `duration_micros`	engine.handle_us_p99_ex_gc: the same p99 with each sample's garbage-collection time taken out.
`handle_samples`	`int`	required	ge 0, unit `dimensionless`	engine.handle_samples: how many samples the percentiles stand on; below 100 the p99 is their maximum.
`other_p99_us`	`float`	required	ge 0, unit `duration_micros`	engine.other_us_p99: every message but order entry – heartbeats, connects, queries, admin.
`gc_pause_max_us`	`float`	required	ge 0, unit `duration_micros`	engine.gc_pause_us_max: the engine's longest garbage collection in the last minute, sampled or not.

Field	Type	Presence	Rules	Description
`gc_time_pct`	`float`	required	ge 0, unit `percent`	engine.gc_time_pct: share of the engine's last beacon interval spent collecting garbage.
`busy_pct`	`float`	required	ge 0, unit `percent`	Engine share of wall time spent handling; near 100 means saturated.
`rejects_by_code`	list of `RejectCount`	required	—	Today's rejects (engine.rejects_by_code) by reason, most first: what makes reject_pct_5m actionable.

`ParticipantStats`

One configured participant (gateway id) as the engine counts it.

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 32	
`role`	`string`	required	max_len 32	
`connected`	`bool`	required	—	
`connected_since`	`string`	`null` when unset	max_len 32	ISO-8601 UTC of the current session's connect; null when not connected or not seen.
`orders`	`RateStats`	required	—	
`reject_pct`	`float`	required	ge 0, unit `percent`	Rejected acks over inbound requests today (as TradingStats.reject_pct_5m); 0 when none.
`fills_today`	`int`	required	ge 0, unit `dimensionless`	
`resting_orders`	`int`	required	ge 0, unit `dimensionless`	

Field	Type	Presence	Rules	Description
`connects_today`	`int`	required	ge 0, unit `dimensionless`	
`disconnects_today`	`int`	required	ge 0, unit `dimensionless`	
`heartbeat_timeouts_today`	`int`	required	ge 0, unit `dimensionless`	

`BookStats`

One order book (design section 8.3); refreshed every book_metrics_interval_sec.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`resting_orders`	`int`	required	ge 0, unit `dimensionless`	
`gtc_orders`	`int`	required	ge 0, unit `dimensionless`	Resting orders with TIF=GTC: the long orders that survive the close. A subset of resting_orders.
`stop_orders`	`int`	required	ge 0, unit `dimensionless`	
`bid_levels`	`int`	required	ge 0, unit `dimensionless`	
`ask_levels`	`int`	required	ge 0, unit `dimensionless`	
`est_bytes`	`int`	required	ge 0, unit `bytes`	An estimate from measured per-object constants, never an exact figure.
`heap_bloat_ratio`	`float`	required	ge 0, unit `dimensionless`	Heap entries over the size compaction tolerates; at most 1.0 in a healthy book.
`trades_today`	`int`	required	ge 0, unit `dimensionless`	
`halted`	`bool`	required	—	

GatewayStats

One process that holds client connections: the ALF, BALF, MD, RALF and drop-copy gateways, the API gateway (WebSocket sessions) and pm-log-srv (LALF sessions). One shape for all, so row 6 of the dashboard is a loop.

Field	Type	Presence	Rules	Description
`key`	`string`	required	max_len 200, min_len 1	The ProcessHealth key of the same process.
`kind`	enum: `ALF`, `BALF`, `MD`, `RALF`, `DC`, `API`, `LOG`	required	—	
`clients`	`int`	required	ge 0, unit `dimensionless`	
`clients_max`	`int`	`null` when unset	unit `dimensionless`	The configured connection limit; null where none exists.
`msgs_per_sec`	`float`	required	ge 0, unit `dimensionless`	Main traffic counter of the kind (commands in, messages out, log events in) over the last 60 s.
`disconnects_today`	`int`	required	ge 0, unit `dimensionless`	
`slow_disconnects_today`	`int`	required	ge 0, unit `dimensionless`	Clients cut off for not reading; the back-pressure signal.

ActiveBreach

One threshold currently in breach (design section 9.6).

Field	Type	Presence	Rules	Description
`rule`	`string`	required	max_len 64	The configuration key, e.g. `engine_handle_p99_us`.
`subject`	`string`	required	max_len 200	What breached: a process key, `host`, a symbol or `exchange`.
`level`	enum: `WARN`, `CRIT`	required	—	

Field	Type	Presence	Rules	Description
`value`	`float`	required	unit `dimensionless`	
`threshold`	`float`	required	unit `dimensionless`	
`since`	`string`	required	max_len 32	ISO-8601 UTC of the breach.

16.12.2 `health.beacon`

Published by: `health_reporter`

Transport: `monitor_push`

Since: 1.2

Every long-running pm-* process to pm-monitor, every beacon_interval_sec: the process's own memory, CPU, threads and file descriptors, how long ago its main loop last completed an iteration, and its application counters and gauges. Sent from a daemon thread, not from the main loop. That is the point: when the main loop hangs the thread keeps reporting, with a growing loop_age_ms, so pm-monitor can tell HUNG (beacons arrive, loop stuck) from SILENT (no beacons, PID alive) from DEAD (no beacons, no PID). A beacon built on the main loop would simply stop, and silence cannot make that distinction.

Field	Type	Presence	Rules	Description
`process`	`string`	required	max_len 32, min_len 1	The name the process uses in its LALF HELLO, e.g. `pm-engine`.
`instance`	`string`	omitted when unset	max_len 32	The –instance name where a process runs more than once, e.g. `desk`.
`pid`	`int`	required	gt 0, unit `dimensionless`	
`host`	`string`	required	max_len 128, min_len 1	socket.gethostname().
`boot_id`	`string`	required	max_len 16, min_len 1	Random per process start; a change means a restart even when the PID is reused.
`seq`	`int`	required	ge 0, unit `dimensionless`	Per boot_id; a gap means lost beacons.

Field	Type	Presence	Rules	Description
`started_at_ns`	`int`	required	ge 0, unit `epoch_nanos`	Process start, for uptime only; ages use pm-monitor's receive clock.
`interval_sec`	`float`	required	gt 0, unit `duration_seconds`	The sender's beacon interval; pm-monitor's silence timers scale with it.
`rss_bytes`	`int`	required	ge 0, unit `bytes`	
`cpu_pct`	`float`	required	ge 0, unit `percent`	Percent of one core since the previous beacon.
`threads`	`int`	required	ge 0, unit `dimensionless`	
`fds`	`int`	omitted when unset	ge 0, unit `dimensionless`	Open file descriptors; absent where the OS has no such count.
`loop_age_ms`	`float`	required	ge 0, unit `duration_millis`	Time since the main loop last called tick().
`loop_lag_ms`	`float`	required	ge 0, unit `duration_millis`	Longest gap between two tick() calls since the previous beacon.
`final`	`bool`	defaults to `False`	—	True on the last beacon of a clean shutdown: STOPPED, not DEAD.

Field	Type	Presence	Rules	Description
`starting`	`bool`	defaults to `False`	—	True until the main loop has ticked once. The reporter starts before a process's loop exists (restoring state, loading config), and that time is not a hung loop; loop_age_ms still grows, so a process stuck starting stays visible.
`metrics`	list of `MetricSample`	required	—	Application metrics (design section 8); may be empty.

Note

PUSH with SNDHWM 10, LINGER 0 and a non-blocking send to pm-monitor's PULL (monitor.pull_port, default 5610).
When pm-monitor is down the beacon is dropped and counted in health.beacons_dropped; an exchange process never blocks on it.
See also: `health.snapshot`

16.12.3 `health.snapshot`

Published by: `monitor`

Transport: `monitor_pub`

Since: 1.2

pm-monitor to the dashboard bridge, once a second: everything the dashboard shows live, already derived. Rates, averages, health states and thresholds are computed once here, so every browser shows the same figures and none re-derives them.

Field	Type	Presence	Rules	Description
`ts`	`string`	required	max_len 32	ISO-8601 UTC with milliseconds; pm-monitor's clock.
`seq`	`int`	required	ge 0, unit `dimensionless`	Per pm-monitor start.
`exchange`	`ExchangeStatus`	required	—	

Field	Type	Presence	Rules	Description
`host`	`HostStats`	required	—	
`processes`	list of `ProcessHealth`	required	—	
`trading`	`TradingStats`	required	—	
`participants`	list of `ParticipantStats`	required	—	
`books`	list of `BookStats`	required	—	
`gateways`	list of `GatewayStats`	required	—	
`thresholds`	list of `ActiveBreach`	required	—	Breaches active now; empty when all is well.

Note

PUB on monitor.pub_port (default 5611).
About 120 KB at 112 participants and 300 symbols (the AI-trader simulation), most of it the participants and books blocks; pm-engine's own per-participant and per-symbol samples are left out of its process record.
A bridge that has seen no snapshot for 3 s reports pm-monitor as down.
See also: `health.beacon`, `health.event`

16.12.4 `health.event`

Published by: `monitor`

Transport: `monitor_pub`

Since: 1.2

pm-monitor to the dashboard bridge, as it happens: an engine lifecycle event (connect, disconnect, diagnostic, session transition, halt) or one pm-monitor generated itself (a process changed health state, a threshold was breached or cleared). These are the dashboard's annotations and its event log, and the rows of metrics.db `events`.

Field	Type	Presence	Rules	Description
`seq`	`int`	required	ge 0, unit `dimensionless`	The metrics.db events.seq of this event.
`ts`	`string`	required	max_len 32	ISO-8601 UTC with milliseconds.

Field	Type	Presence	Rules	Description
`kind`	enum: `CONNECT`, `AUTH_REJECT`, `DISCONNECT`, `DIAGNOSTIC`, `SESSION`, `HALT`, `RESUME`, `PROCESS_STATUS`, `THRESHOLD`	required	—	
`severity`	enum: `INFO`, `WARN`, `CRIT`	required	—	How the dashboard colours it; a heart-beat_timeout disconnect is WARN.
`subject`	`string`	required	max_len 200	A gateway id, a process key, a symbol, or a threshold rule.
`detail`	`string`	defaults to `''`	max_len 512	e.g. `heartbeat_timeout`, `HEALTHY -> HUNG`, `CONTINUOUS -> CLOSING_AUCTION`.

See also: `health.snapshot`, `system.gateway_auth.{gateway_id}`, `system.gateway_bye.{gateway_id}`, `system.diagnostic`

16.13 Family `index`

16.13.1 Record types

`DaySummary`

The session's open, high and low index level. All three arrive together or not at all: _update_day_ohlc sets open, high and low in one branch and _reset_for_new_session clears all three, so there has never been a state where one is known and another is not. As three flat keys under one guard that was a convention; as a nullable record it is unrepresentable otherwise, which is design section 16.2's whole argument.

Field	Type	Presence	Rules	Description
`open`	`float`	required	unit `dimensionless`	The first level computed this session.
`high`	`float`	required	unit `dimensionless`	

Field	Type	Presence	Rules	Description
`low`	`float`	required	unit `dimensionless`	

`HistoryRecord`

One structural audit entry, replayed verbatim from pm-index's append-only JSONL archive. This is a union of five shapes discriminated by `type`, and the IDL has no variant construct (section 20.3). Every field the five do not share is therefore optional, and the spec cannot state "a CORP_ACTION always carries action and detail" - that rule lives in _handle_corp_action. What it can state is the field set, the units and the types, which is what every consumer needs: all six read the records with `.get(key, default)` and dispatch on `type`. The optional fields omit rather than null because the archive omits: a record written before this spec existed must read back and re-emit unchanged, or specifying the family would rewrite history.

Field	Type	Presence	Rules	Description
`type`	enum: `INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST`, `REBALANCE`	required	—	The discriminator. IndexHistory.query drops any other value with a warning, so an unknown type never reaches the wire.
`ts_ns`	`int`	required	unit `epoch_nanos`	
`index_id`	`string`	required	max_len 32	
`level`	`float`	required	unit `dimensionless`	The index level immediately after the event was applied.
`symbol`	`string`	omitted when empty	max_len 16	CORP_ACTION, ADD_CONSTITUENT and DELIST.
`action`	`string`	omitted when empty	max_len 32	CORP_ACTION only.
`detail`	`string`	omitted when empty	max_len 128	CORP_ACTION only; a rendered summary such as 'shares=1000'.
`old_divisor`	`float`	omitted when unset	unit `dimensionless`	Every type but INIT.
`new_divisor`	`float`	omitted when unset	unit `dimensionless`	Every type but INIT.

Field	Type	Presence	Rules	Description
`base_value`	`float`	omitted when unset	unit `dimensionless`	INIT only.
`divisor`	`float`	omitted when unset	unit `dimensionless`	INIT only, and the odd one out: every other type reports the divisor as an old/new pair. Kept as written rather than normalised, because the archive on disk already says this.
`constituents`	list of `string`	omitted when empty	—	INIT only.
`shares_outstanding`	`int`	omitted when unset	unit `shares`	ADD_CONSTITUENT only, and absent on records written before 5.2e: the handler used the share count to weight the constituent and then dropped it from the audit entry, so what a constituent was added with was durably recorded nowhere. Optional rather than required because the archive on disk still holds records without it.
`reference_price`	`float`	omitted when unset	unit `display_price`	ADD_CONSTITUENT only.
`symbols`	list of `string`	omitted when empty	—	REBALANCE only: the symbols the batch actually applied to.

`RebalanceUpdate`

One entry of a rebalance batch. Mechanically a SHARES_ISSUANCE corporate action, applied to every named existing constituent as one batch with a single recompute and publish rather than one round-trip per symbol.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`new_shares_outstanding`	`int`	required	gt 0, unit `shares`	

16.13.2 `index.update`

Published by: `index`

Transport: `engine_pub`

Since: 1.0

pm-index to subscribers: the current level of one index, published on every constituent trade subject to a rate limit, and forced on a structural change or at end of day.

Field	Type	Presence	Rules	Description
`index_id`	`string`	required	max_len 32	
`level`	`float`	required	unit `dimensionless`	
`aggregate_cap`	`float`	required	unit `money`	Sum of constituent market capitalisations.
`divisor`	`float`	required	unit `dimensionless`	Level = aggregate_cap / divisor.
`session_state`	`string`	required	max_len 32	Mirrors session.state; a plain string there and here.
`ts_ns`	`int`	required	unit `epoch_nanos`	
`day`	DaySummary	omitted when unset	—	

Note

`day` is absent before the first level of a session is computed and after _reset_for_new_session clears it.
All three consumers - alf_console's display, pm-stats' snapshot writer and md_gateway's CALF normaliser - read it with `.get` and test `is not None`, so absent and null are the same thing to every one of them.
See also: `session.state`

16.13.3 `index.history_request`

Published by: `admin`, `api_gateway`, `gateway`

Transport: `engine_pub`

Since: 1.0

Gateway or operator to pm-index: replay the structural audit log.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`index_id`	`string`	required	max_len 32	
`from_ts_ns`	`int`	required	unit `epoch_nanos`	
`to_ts_ns`	`int`	required	unit `epoch_nanos`	
`types`	list of `string`	omitted when empty	—	Record types to include; omitted means all structural types.
`max_records`	`int`	defaults to `10000`	gt 0, unit `dimensionless`	

Note

pm-index's history is structural only - index creation, corporate actions, constituent changes, rebalances.
Level and end-of-day time-series history lives in pm-stats.
`types` is omitted when unset rather than sent as a default.
The hand-written builder defaulted it to four of the five structural types and silently dropped REBALANCE from every reply that took the default; the server's own default is the full set, and it cannot tell an omitted `types` from a deliberate one.
`max_records` keeps its default because the builder's value and the server's agree.
See also: `index.history.{GW_ID}`, `index.error.{GW_ID}`

16.13.4 `index.history.{gateway_id}`

Published by: `index`

Transport: `engine_pub`

Since: 1.0

pm-index to requestor: the matching audit records.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`index_id`	`string`	required	max_len 32	
`records`	list of `HistoryRecord`	required	—	
`warnings`	list of `string`	omitted when empty	—	

Note

`warnings` reports lines the archive could not parse and record types it did not recognise.
It is omitted when there are none, which is what the hand-written builder's `if warnings:` did.
See also: `index.history_request`

16.13.5 `index.corp_action`

Published by: `admin`

Transport: `engine_pub`

Since: 1.0

Operator to pm-index: apply a corporate action.

Field	Type	Presence	Rules	Description
`action`	enum: `SPLIT`, `CASH_DIVIDEND`, `SHARES_ISSUANCE`	required	—	
`index_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`gateway_id`	`string`	required	max_len 32	
`ratio_numerator`	`int`	omitted when unset	gt 0, unit `dimensionless`	SPLIT.
`ratio_denominator`	`int`	omitted when unset	gt 0, unit `dimensionless`	SPLIT.
`dividend_per_share`	`float`	omitted when unset	gt 0, unit `money`	CASH_DIVIDEND.
`new_shares_outstanding`	`int`	omitted when unset	gt 0, unit `shares`	SHARES_ISSUANCE.
`command_id`	`string`	omitted when empty	max_len 64	Echoed on the ack so a caller can correlate.

Note

The four parameters are action-specific and flat: SPLIT reads the two ratio fields, CASH_DIVIDEND reads dividend_per_share, SHARES_ISSUANCE reads new_shares_outstanding, and each is read with `.get(key, 0)` inside its own branch of _handle_corp_action.
A discriminated union would say that properly and the IDL has none - see design section 20.3 for why one was not built for a single family.
See also: `index.corp_action_ack.{GW_ID}`, `index.error.{GW_ID}`

16.13.6 `index.constituent_change`

Published by: `admin`

Transport: `engine_pub`

Since: 1.0

Operator to pm-index: add or delist a constituent.

Field	Type	Presence	Rules	Description
`change_type`	enum: `ADD`, `DELIST`	required	—	
`index_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`gateway_id`	`string`	required	max_len 32	
`shares_outstanding`	`int`	omitted when unset	gt 0, unit `shares`	ADD.
`initial_price`	`float`	omitted when unset	gt 0, unit `display_price`	ADD.
`command_id`	`string`	omitted when empty	max_len 64	Echoed on the ack so a caller can correlate.

Note

Both parameters belong to ADD and neither to DELIST, and the hand-written builder omitted each independently rather than as a pair - so unlike DaySummary they are two guards, not one, and stay flat.
See also: `index.constituent_change_ack.{GW_ID}`

16.13.7 `index.rebalance`

Published by: `api_gateway`

Transport: `engine_pub`

Since: 1.0

ADMIN to pm-index: set shares outstanding for several constituents in one batch.

Field	Type	Presence	Rules	Description
`index_id`	`string`	required	max_len 32	
`gateway_id`	`string`	required	max_len 32	
`updates`	list of `RebalanceUpdate`	required	min_items 1	Never empty; the handler rejects an empty batch.
`command_id`	`string`	omitted when empty	max_len 64	Echoed on the ack so a caller can correlate.

Note

The whole batch is validated before any of it is applied, so an invalid entry anywhere rejects all of it - the all-or-nothing guarantee the single-action handlers get for free by only ever doing one mutation.
See also: `index.rebalance_ack.{GW_ID}`

16.13.8 `index.corp_action_ack.{gateway_id}`

Published by: `index`

Transport: `engine_pub`

Since: 1.0

pm-index to requestor: the corporate action's outcome.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`ts_ns`	`int`	required	unit `epoch_nanos`	
`index_id`	`string`	omitted when empty	max_len 32	
`level`	`float`	omitted when unset	unit `dimensionless`	
`divisor`	`float`	omitted when unset	unit `dimensionless`	
`old_divisor`	`float`	omitted when unset	unit `dimensionless`	The divisor immediately before this action was applied. Same field name and purpose as HistoryRecord.old_divisor, so a post-mortem does not need pm-index's local JSONL archive to see what changed — the wire event is self-contained. Present only alongside divisor, i.e. on acceptance.
`command_id`	`string`	omitted when empty	max_len 64	

Note

level and divisor are the recomputed values and are present only on acceptance; index_id is absent on the paths that reject before resolving one.
reason is always emitted, as " " on success, because the hand-written builder put it in the base payload rather than under a guard.

16.13.9 `index.constituent_change_ack.{gateway_id}`

Published by: `index`

Transport: `engine_pub`

Since: 1.0

pm-index to requestor: the constituent change's outcome.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`ts_ns`	`int`	required	unit `epoch_nanos`	
`index_id`	`string`	omitted when empty	max_len 32	
`level`	`float`	omitted when unset	unit `dimensionless`	
`divisor`	`float`	omitted when unset	unit `dimensionless`	
`old_divisor`	`float`	omitted when unset	unit `dimensionless`	The divisor immediately before this action was applied. Same field name and purpose as HistoryRecord.old_divisor, so a post-mortem does not need pm-index's local JSONL archive to see what changed — the wire event is self-contained. Present only alongside divisor, i.e. on acceptance.
`command_id`	`string`	omitted when empty	max_len 64	

Note

Field for field the same payload as index.corp_action_ack, on its own topic.
Two topics rather than one because a caller waits on the specific reply to the command it sent, and commands/client.py names that topic when it registers the future.

16.13.10 `index.rebalance_ack.{gateway_id}`

Published by: `index`

Transport: `engine_pub`

Since: 1.0

pm-index to ADMIN: the batch's outcome.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`ts_ns`	`int`	required	unit `epoch_nanos`	
`updated_symbols`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`index_id`	`string`	omitted when empty	max_len 32	
`level`	`float`	omitted when unset	unit `dimensionless`	
`divisor`	`float`	omitted when unset	unit `dimensionless`	
`old_divisor`	`float`	omitted when unset	unit `dimensionless`	The divisor immediately before this action was applied. Same field name and purpose as HistoryRecord.old_divisor, so a post-mortem does not need pm-index's local JSONL archive to see what changed — the wire event is self-contained. Present only alongside divisor, i.e. on acceptance.
`command_id`	`string`	omitted when empty	max_len 64	

Note

updated_symbols is always emitted, as 0 on rejection: the builder puts it in the base payload beside accepted and reason, and a rejected batch applied nothing.

16.13.11 `index.error.{gateway_id}`

Published by: `index`

Transport: `engine_pub`

Since: 1.0

pm-index to requestor: the request could not be routed to an index at all.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	Always false; present so every reply has the same first key.
`reason`	`string`	required	max_len 512	
`ts_ns`	`int`	required	unit `epoch_nanos`	

Note

Distinct from a rejecting ack.
An unknown index_id means pm-index cannot know which ack topic the caller is waiting on, so it answers on the one topic every index caller subscribes to.
Once the index is known, a bad symbol or parameter comes back as accepted: false on the specific ack instead.

16.14 Family `log`

16.14.1 Record types

`LogFilter`

A row predicate, applied two ways by the server: evaluated in Python against a freshly persisted row on the live path, and compiled to a parameterised SQL WHERE clause for backfill. One definition with two evaluators is what guarantees a subscriber's backfill and its subsequent live stream contain the same kind of rows - a mismatch would show up as rows appearing or vanishing at the seam. Every field is optional; an empty filter matches everything.

Field	Type	Presence	Rules	Description
`min_level`	`string`	omitted when unset	max_len 16	Lowest level to include, e.g. WARNING.
`processes`	list of `string`	defaults to `[]`	—	Process names to include; empty means all.
`loggers`	list of `string`	defaults to `[]`	—	Logger names to include; empty means all.

Field	Type	Presence	Rules	Description
`sessions`	list of `string`	defaults to `[]`	—	Session ids to include; empty means all.
`contains`	`string`	omitted when unset	max_len 256	Substring the message must contain.
`exceptions_only`	`bool`	defaults to `False`	—	

`LevelCount`

How many rows of one level a NOTIFY subscription has buffered. This was a map on the wire - {“INFO”: 3, “ERROR”: 1} - which the IDL excludes deliberately (design section 15.4). The key was a value, and a list of records says so: the level is a field, not a key.

Field	Type	Presence	Rules	Description
`level`	`string`	required	max_len 16	
`count`	`int`	required	ge 0, unit `dimensionless`	

`LogRow`

One persisted log line, exactly the columns log_events stores. Carried by both log.event (live) and log.backfill (history) so a viewer sees one row shape at the seam between them.

Field	Type	Presence	Rules	Description
`seq`	`int`	required	unit `dimensionless`	Monotonic server sequence; the cursor for backfill.
`client_ts`	`string`	required	—	Producer wall-clock, ISO-8601 (e.g. 2026-07-29T10:00:00.000Z).
`server_ts`	`string`	required	—	Server receive time, ISO-8601; the log_events columns are TEXT.
`process`	`string`	required	max_len 64	
`instance`	`string`	required	max_len 64	
`pid`	`int`	required	unit `dimensionless`	
`host`	`string`	required	max_len 128	
`session`	`string`	required	max_len 64	

Field	Type	Presence	Rules	Description
`level`	`string`	required	max_len 16	
`logger`	`string`	required	max_len 128	
`module`	`string`	required	max_len 128	
`line`	`int`	required	unit `dimensionless`	
`has_exception`	`bool`	required	—	
`truncated`	`bool`	required	—	
`message`	`string`	required	—	

`SubscriptionStatus`

One subscription's live counters, reported by log.status. This is the record that motivated allowing a record inside a record: it carries the subscription's own LogFilter. Flattening that into filter_min_level and friends is the `a_b` flattening section 16.2 argued against, and forbidding depth was a rule broader than its reason - what the generators cannot survive is a cycle, not a level.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`mode`	enum: `STREAM`, `NOTIFY`	required	—	
`filter`	`LogFilter`	required	—	
`lease_sec`	`float`	required	unit `dimensionless`	
`lease_remaining_sec`	`float`	required	unit `dimensionless`	
`age_sec`	`float`	required	unit `dimensionless`	
`pending_rows`	`int`	required	unit `dimensionless`	
`pending_count`	`int`	required	unit `dimensionless`	
`sent_rows`	`int`	required	unit `dimensionless`	
`sent_messages`	`int`	required	unit `dimensionless`	
`dropped_rows`	`int`	required	unit `dimensionless`	
`renewals`	`int`	required	unit `dimensionless`	

16.14.2 `log.subscribe`

Published by: `log_client`

Transport: `engine_pub`

Since: 1.0

Subscriber to pm-log-srv: open or replace a leased stream.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	

Field	Type	Presence	Rules	Description
`mode`	enum: `STREAM`, `NOTIFY`	defaults to `'STREAM'`	—	STREAM pushes every row; NOTIFY pushes periodic counts.
`filter`	`LogFilter`	omitted when unset	—	
`backfill_minutes`	`int`	omitted when unset	unit `dimensionless`	Replay this many minutes before the live stream starts.
`lease_sec`	`int`	omitted when unset	unit `dimensionless`	How long the subscription survives without a renew.
`notify_interval_ms`	`int`	omitted when unset	unit `dimensionless`	

Note

Everything past sub_id and mode is omitted when unset rather than sent as a default, which is what the hand-written builder did - the server applies its own defaults and cannot tell an omitted lease_sec from one that happens to equal the default.
See also: `log.renew`, `log.unsubscribe`

16.14.3 `log.renew`

Published by: `log_client`

Transport: `engine_pub`

Since: 1.0

Lease keepalive. The liveness signal: a subscriber that stops renewing is dropped, which is how the server reclaims a viewer that went away without unsubscribing.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`timestamp`	`float`	required	unit `epoch_seconds`	

See also: `log.lease_expired.{SUB_ID}`

16.14.4 `log.unsubscribe`

Published by: `log_client`

Transport: `engine_pub`

Since: 1.0

Close a subscription immediately rather than letting it lapse.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`timestamp`	`float`	required	unit `epoch_seconds`	

16.14.5 `log.backfill_request`

Published by: `log_client`

Transport: `engine_pub`

Since: 1.0

Replay the last N minutes of history for one subscription.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`minutes`	`int`	required	gt 0, unit `dimensionless`	
`filter`	`LogFilter`	omitted when unset	—	Defaults to the subscription's own filter when omitted.
`max_rows`	`int`	omitted when unset	unit `dimensionless`	

See also: `log.backfill.{SUB_ID}`

16.14.6 `log.status_request`

Published by: `log_client`

Transport: `engine_pub`

Since: 1.0

Ask for subscription and server diagnostics.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`timestamp`	`float`	required	unit `epoch_seconds`	

See also: `log.status.{SUB_ID}`

16.14.7 `log.subscribe_ack.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

Confirm a subscription and echo the terms the server chose.

Field	Type	Presence	Rules	Description
`accepted`	`bool`	required	—	
`sub_id`	`string`	required	max_len 64	
`proto`	`string`	required	max_len 16	
`server`	`string`	required	max_len 64	
`mode`	enum: `STREAM`, `NOTIFY`	required	—	
`filter`	`LogFilter`	required	—	
`lease_sec`	`float`	required	unit `dimensionless`	
`renew_before_sec`	`float`	required	unit `dimensionless`	Renew sooner than this; half the lease.
`notify_interval_ms`	`int`	required	unit `dimensionless`	
`last_seq`	`int`	required	unit `dimensionless`	
`backfill_request_id`	`string`	omitted when empty	max_len 64	
`timestamp`	`float`	required	unit `epoch_seconds`	

Note

The filter comes back parsed rather than as sent, so a subscriber can see what the server actually understood - which is where a lenient filter parse would otherwise hide a typo.

16.14.8 `log.renew_ack.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

Confirm a keepalive and say how long the lease now has.

Field	Type	Presence	Rules	Description
`accepted`	`bool`	required	—	
`sub_id`	`string`	required	max_len 64	
`lease_sec`	`float`	required	unit `dimensionless`	
`expires_in_sec`	`float`	required	unit `dimensionless`	
`last_seq`	`int`	required	unit `dimensionless`	
`timestamp`	`float`	required	unit `epoch_seconds`	

16.14.9 `log.unsubscribe_ack.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

Confirm a close, or say there was nothing to close.

Field	Type	Presence	Rules	Description
`accepted`	`bool`	required	—	
`sub_id`	`string`	required	max_len 64	
`reason`	`string`	omitted when empty	max_len 256	
`timestamp`	`float`	required	unit `epoch_seconds`	

16.14.10 `log.status.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

Server and subscription diagnostics, on request.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`server`	`string`	required	max_len 64	
`proto`	`string`	required	max_len 16	
`subscribers`	`int`	required	unit `dimensionless`	
`active_backfills`	`int`	required	unit `dimensionless`	
`last_seq`	`int`	required	unit `dimensionless`	
`inbox_dropped`	`int`	required	unit `dimensionless`	
`subscription`	`SubscriptionStatus`	`null` when unset	—	
`timestamp`	`float`	required	unit `epoch_seconds`	

Note

subscription is null when the requester has no live subscription - asking for status is legal without one, and null says “you have none” where an absent key would say “the server declined to tell you”.

16.14.11 `log.backfill.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

One chunk of replayed history for a subscription.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`request_id`	`string`	required	max_len 64	
`chunk`	`int`	required	unit `dimensionless`	
`rows`	list of `LogRow`	required	—	
`row_count`	`int`	required	unit `dimensionless`	
`done`	`bool`	required	—	
`total_sent`	`int`	required	unit `dimensionless`	
`truncated`	`bool`	required	—	
`last_seq`	`int`	required	unit `dimensionless`	
`timestamp`	`float`	required	unit `epoch_seconds`	

Note

Chunked because a backfill can be far larger than one message: done marks the last chunk, and truncated says the server stopped at max_rows rather than at the end of history.
The two are different answers to "why did it stop".
See also: `log.backfill_request`, `log.event.{SUB_ID}`

16.14.12 `log.event.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

A batch of live rows for a STREAM subscription.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`rows`	list of `LogRow`	required	min_items 1	Never empty; the server skips a flush with nothing to send.
`row_count`	`int`	required	unit `dimensionless`	
`seq_from`	`int`	required	unit `dimensionless`	
`seq_to`	`int`	required	unit `dimensionless`	

Field	Type	Presence	Rules	Description
`server_last_seq`	`int`	required	unit `dimensionless`	
`dropped`	`int`	required	unit `dimensionless`	Lifetime rows dropped for this subscription, not this batch.
`timestamp`	`float`	required	unit `epoch_seconds`	

See also: `log.notify.{SUB_ID}`

16.14.13 `log.notify.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

Periodic counts for a NOTIFY subscription: how much happened, without the rows.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`count`	`int`	required	unit `dimensionless`	
`levels`	list of `LevelCount`	required	—	
`last_seq`	`int`	required	unit `dimensionless`	
`server_last_seq`	`int`	required	unit `dimensionless`	
`timestamp`	`float`	required	unit `epoch_seconds`	

Note

levels was a map keyed by level name.
It is a list of records now - the key was a value, and design section 15.4 says a spec that appears to need a map is describing a message that should have been this.

16.14.14 `log.lease_expired.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

The subscription was reaped for want of a renew.

Field	Type	Presence	Rules	Description
`sub_id`	`string`	required	max_len 64	
`reason`	`string`	required	max_len 256	

Field	Type	Presence	Rules	Description
`lease_sec`	`float`	required	unit `dimensionless`	
`dropped_rows`	`int`	required	unit `dimensionless`	
`timestamp`	`float`	required	unit `epoch_seconds`	

Note

Published on the off-chance the subscriber is alive but wedged: nothing about a crashed process is visible on a PUB socket, so this tells a client that is in fact listening that it must re-subscribe rather than wait for rows that will never come.

16.14.15 `log.error.{sub_id}`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

A control request was rejected, with a machine-readable code.

Field	Type	Presence	Rules	Description
`accepted`	`bool`	required	—	Always false; present so every reply has the same first key.
`sub_id`	`string`	required	max_len 64	
`code`	`string`	required	max_len 32	
`reason`	`string`	required	max_len 512	
`timestamp`	`float`	required	unit `epoch_seconds`	

16.14.16 `log.server_state`

Published by: `log_server`

Transport: `engine_pub`

Since: 1.0

Periodic server heartbeat and configuration, broadcast to everyone rather than addressed - it is how a viewer finds the server at all.

Field	Type	Presence	Rules	Description
`server`	`string`	required	max_len 64	
`state`	enum: `UP`, `DOWN`	required	—	
`proto`	`string`	required	max_len 16	
`pub_addr`	`string`	required	max_len 128	

Field	Type	Presence	Rules	Description
`pull_addr`	`string`	required	max_len 128	
`subscribers`	`int`	required	unit `dimensionless`	
`active_backfills`	`int`	required	unit `dimensionless`	
`last_seq`	`int`	required	unit `dimensionless`	
`inbox_dropped`	`int`	required	unit `dimensionless`	
`default_lease_sec`	`float`	required	unit `dimensionless`	
`timestamp`	`float`	required	unit `epoch_seconds`	

16.15 Family `news`

16.15.1 `news.event`

Published by: `market_sim`

Transport: `sim_pub`

Since: 1.0

pm-market-sim to everyone: one headline, or a change in what an earlier one is known to be (a rumour confirmed or retracted).

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 32	
`ts_ns`	`int`	required	unit `epoch_nanos`	
`scope`	enum: `SYMBOL`, `SECTOR`, `MARKET`	required	—	
`targets`	list of `string`	required	—	Symbols (SYMBOL) or sector names (SECTOR); empty for MARKET.
`kind`	enum: `EARNINGS`, `GUIDANCE`, `MNA`, `REGULATORY`, `PRODUCT`, `LEGAL`, `MANAGEMENT`, `MACRO`, `OPERATIONS`, `RESEARCH`, `CONTRACT`, `ESG`	required	—	
`status`	enum: `RUMOUR`, `CONFIRMED`, `RETRACTED`	required	—	

Field	Type	Presence	Rules	Description
`headline`	`string`	required	max_len 200	
`sentiment`	`float`	required	ge -1, le 1, unit `dimensionless`	Tone of the news, -1 (bad) to 1 (good); not its size.
`credibility`	`float`	omitted when unset	ge 0, le 1, unit `dimensionless`	How believable a rumour is; absent on confirmed news.
`related_id`	`string`	defaults to `''`	max_len 32	The rumour a CONFIRMED or RETRACTED event resolves.

Note

A rumour comes first with status RUMOUR and a credibility; when it resolves, a second event with a new id carries status CONFIRMED or RETRACTED and points back with `related_id`. A confirmation is when the value moves; a retraction moves nothing.
Published on pm-market-sim's PUB socket; the API gateway relays it to every key and pm-ticker shows it.
See also: `sim.value`

16.16 Family `order`

16.16.1 Record types

`OcoLeg`

One side of an OCO pair. It has no symbol or quantity of its own: both legs trade the same instrument in the same size, and the OCO carries those. That is what makes it a different record from a combo leg, which does own a symbol and a quantity.

Field	Type	Presence	Rules	Description
`side`	enum: `BUY`, `SELL`	required	—	
`order_type`	enum: `MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	required	—	

Field	Type	Presence	Rules	Description
`price_ticks`	`ticks`	omitted when unset	unit `ticks`	Limit price in engine ticks. Absent for a leg with none. The scale is `order.oco.tick_decimals`: both legs trade one instrument, so one scale covers the pair.
`stop_price_ticks`	`ticks`	omitted when unset	unit `ticks`	
`trail_offset_ticks`	`ticks`	omitted when unset	unit `ticks`	

`ComboLeg`

One leg of a combo. Unlike an OcoLeg it owns a symbol and a quantity: the legs of a combo trade different instruments, in sizes that need not match. That is why the two are separate types rather than one shared `leg` - an early draft of design section 15 assumed they could be merged and was wrong.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`side`	enum: `BUY`, `SELL`	required	—	
`order_type`	enum: `MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	required	—	
`quantity`	`int`	required	gt 0, unit `shares`	

Field	Type	Presence	Rules	Description
`tick_decimals`	`int`	required	ge 0, le 8, unit `dimensionless`	Decimal scale for this leg's tick prices; 1 tick = 10^-tick_decimals. Per leg rather than on the combo, because a combo's legs trade different instruments and two instruments need not share a scale - which is exactly the distinction that makes ComboLeg a separate record from OcoLeg.
`price_ticks`	`ticks`	`null` when unset	unit `ticks`	Limit price in engine ticks; null for a leg with none.
`stop_price_ticks`	`ticks`	`null` when unset	unit `ticks`	
`smp_action`	enum: `NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	`null` when unset	—	Null means the client did not specify SMP, which is distinct from an explicit NONE. Combo-level in the ALF protocols, so every leg carries the same value.

`OrderDisplay`

One resting order as the engine reports it in an `order.orders` snapshot, in display units. It is `Order.to_dict()` with price, stop_price and trail_offset converted from ticks to display money - the projection `order_to_display_dict` builds so an operator reads prices in the same money the book shows, not raw ticks. `timestamp` is carried through as `ts_ns` unconverted (AR-0.3b): unlike price, a raw nanosecond integer is exactly what a post-mortem needs to compare against `trade.executed.ts_ns` and the rest of the wire's ordering convention, and converting it to seconds here would only have to be undone by anything that wanted to compare it. `spec/messages/index.yaml`'s family made the same call for the same reason in AR-0.3; this type was out of scope for that pass (it is not part of the index family) and CP-0's own checklist item 3 is what caught it being left behind. Gateway_id is

not included here; it is topic-only (part of the message topic as order.orders.{gateway_id}, not part of the record). The record contains the order state (id, symbol, side, etc.) exactly as Order.to_dict() produces, minus gateway_id; the eleven nullable ones ride as null when unset.

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 64	Engine order id; a UUID string.
`symbol`	`string`	required	max_len 16	
`side`	enum: `BUY`, `SELL`	required	—	
`order_type`	enum: `MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	required	—	
`tif`	enum: `DAY`, `GTC`, `ATO`, `ATC`	required	—	
`quantity`	`int`	required	gt 0, unit `shares`	Total original quantity.
`remaining_qty`	`int`	required	ge 0, unit `shares`	Quantity yet to be filled.
`trail_offset`	`float`	`null` when unset	unit `display_price`	TRAILING_STOP: trail distance, in display money.
`oco_group_id`	`string`	`null` when unset	max_len 64	

Field	Type	Presence	Rules	Description
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	Client-supplied submission time (Order.timestamp, unconverted). NOT the book's time priority key - see arrival_seq. AR-0.3b: previously named "timestamp" and expressed as a seconds-based float; see OrderDisplay's type doc for why that was out of step with the rest of the wire and got fixed here rather than left as the one field of that kind outside log.yaml.
`status`	enum: `NEW`, `PARTIAL`, `FILLED`, `CANCELLED`, `REJECTED`, `EXPIRED`	required	—	
`price`	`float`	`null` when unset	unit `display_price`	Limit price in display money. Null for MARKET, which has none.
`stop_price`	`float`	`null` when unset	unit `display_price`	STOP / STOP_LIMIT / TRAILING_STOP trigger.
`visible_qty`	`int`	`null` when unset	unit `shares`	ICEBERG: fixed peak size.
`displayed_qty`	`int`	`null` when unset	unit `shares`	ICEBERG: current visible slice on the book.

Field	Type	Presence	Rules	Description
`smp_action`	enum: `NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	`null` when unset	—	Self-match prevention. Null means the client did not specify SMP at all, distinct from an explicit NONE. See SmpAction's docstring.
`combo_parent_id`	`string`	`null` when unset	max_len 64	
`leg_index`	`int`	`null` when unset	unit `dimensionless`	Position in the parent combo's legs, 0-based.
`origin`	enum: `ORDER`, `QUOTE`, `IMPLIED`	defaults to `'ORDER'`	—	Defaulted rather than nullable: to_dict always supplies ORDER.

Field	Type	Presence	Rules	Description
`is_seed`	`bool`	defaults to `False`	—	True when this order was created by config-driven startup seeding (market_maker_quotes / market_maker_combos in the engine config) rather than a live gateway submission. Purely observational — nothing in the engine branches on it; origin still governs routing. Lets a post-mortem tell a config bootstrap apart from real participant activity without depending on id-naming conventions. Absent on records written before this field existed.
`quote_id`	`string`	`null` when unset	max_len 64	
`client_tag`	`string`	`null` when unset	max_len 64	Client correlation tag, echoed on every lifecycle event.
`arrival_seq`	`int`	defaults to `0`	unit `dimensionless`	Engine-assigned monotonic arrival sequence; 0 = unassigned.

`PriceLevelOrder`

One resting order as reported by order.price_level_orders — the same projection as OrderDisplay (order_to_display_dict, including its ts_ns field — see OrderDisplay's doc for why that one is not converted to display units the way price/stop_price/trail_offset are), with one field added:

gateway_id. OrderDisplay can leave gateway_id topic-only because an order.orders reply is always about a single, already-known gateway; a price_level_orders reply spans every gateway resting at a symbol/price, so each record must say whose order it is. The generator has no type-extension mechanism, so this duplicates OrderDisplay's field list rather than referencing it — keep the two in sync by hand if OrderDisplay's fields change.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Owning gateway — the one piece of information OrderDisplay omits because its own topic already says it.
`id`	`string`	required	max_len 64	Engine order id; a UUID string.
`symbol`	`string`	required	max_len 16	
`side`	enum: `BUY`, `SELL`	required	—	
`order_type`	enum: `MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	required	—	
`tif`	enum: `DAY`, `GTC`, `ATO`, `ATC`	required	—	
`quantity`	`int`	required	gt 0, unit `shares`	Total original quantity.
`remaining_qty`	`int`	required	ge 0, unit `shares`	Quantity yet to be filled.
`trail_offset`	`float`	`null` when unset	unit `display_price`	TRAILING_STOP: trail distance, in display money.
`oco_group_id`	`string`	`null` when unset	max_len 64	

Field	Type	Presence	Rules	Description
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	Client-supplied submission time (Order.timestamp, unconverted). NOT the book's time priority key - see arrival_seq. AR-0.3b: previously named "timestamp" and expressed as a seconds-based float; see OrderDisplay's type doc for why that was out of step with the rest of the wire and got fixed here rather than left as the one field of that kind outside log.yaml.
`status`	enum: `NEW`, `PARTIAL`, `FILLED`, `CANCELLED`, `REJECTED`, `EXPIRED`	required	—	
`price`	`float`	`null` when unset	unit `display_price`	Limit price in display money. Null for MARKET, which has none.
`stop_price`	`float`	`null` when unset	unit `display_price`	STOP / STOP_LIMIT / TRAILING_STOP trigger.
`visible_qty`	`int`	`null` when unset	unit `shares`	ICEBERG: fixed peak size.
`displayed_qty`	`int`	`null` when unset	unit `shares`	ICEBERG: current visible slice on the book.

Field	Type	Presence	Rules	Description
`smp_action`	enum: `NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	`null` when unset	—	Self-match prevention. Null means the client did not specify SMP at all, distinct from an explicit NONE. See SmpAction's docstring.
`combo_parent_id`	`string`	`null` when unset	max_len 64	
`leg_index`	`int`	`null` when unset	unit `dimensionless`	Position in the parent combo's legs, 0-based.
`origin`	enum: `ORDER`, `QUOTE`, `IMPLIED`	defaults to `'ORDER'`	—	Defaulted rather than nullable: to_dict always supplies ORDER.

Field	Type	Presence	Rules	Description
`is_seed`	`bool`	defaults to `False`	—	True when this order was created by config-driven startup seeding (market_maker_quotes / market_maker_combos in the engine config) rather than a live gateway submission. Purely observational — nothing in the engine branches on it; origin still governs routing. Lets a post-mortem tell a config bootstrap apart from real participant activity without depending on id-naming conventions. Absent on records written before this field existed.
`quote_id`	`string`	`null` when unset	max_len 64	
`client_tag`	`string`	`null` when unset	max_len 64	Client correlation tag, echoed on every lifecycle event.
`arrival_seq`	`int`	defaults to `0`	unit `dimensionless`	Engine-assigned monotonic arrival sequence; 0 = unassigned.

16.16.2 `execution_report` (no bus topic)

Published by: `gateway`

Transport: `balf`

Since: 1.0

Private per-order fill notification, sent to the gateway session that owns the order. Both sides of a match receive their own report.

Field	Type	Presence	Rules	Description
`client_order_id`	`int`	required	unit `dimensionless`	Echoed from the original NEW_ORDER so a client can correlate.
`order_id`	`int`	required	unit `dimensionless`	Session-scoped BALF order id assigned by the gateway. A u64 on the wire, not a string - this is the field the reference example got wrong.
`fill_price`	`float`	required	gt 0, unit `display_price`	Execution price in display money. On the wire it is an i64 scaled by the fixed BALF PRICE_SCALE of 10^8, never by the instrument's tick_decimals.
`fill_qty`	`int`	required	gt 0, unit `shares`	Quantity matched in this event, not cumulatively.
`remaining_qty`	`int`	required	ge 0, unit `shares`	Unfilled quantity after this fill; zero means the order is done.
`timestamp_ns`	`int`	required	ge 0, unit `epoch_nanos`	Trade time in nanoseconds since the Unix epoch.
`symbol`	`string`	required	max_len 8, pattern `^[A-Z0-9._]+$`	Instrument, echoed for convenience; matches the original order.
`side`	enum: `BUY`, `SELL`	required	—	Side of the filled order.

Field	Type	Presence	Rules	Description
`status`	enum: `PARTIAL`, `FILLED`	required	—	Whether this fill completed the order. Only these two values exist on BALF - there is no NEW or CANCELLED execution report.

Note

Sent for every partial or full fill, so a single order may produce several.
remaining_qty reaching zero is what marks the order done; status FILLED says the same thing and the two must agree.
See also: `trade.executed`

16.16.3 `stop_triggered` (no bus topic)

Published by: `gateway`

Transport: `balf`

Since: 1.3

A stop, stop-limit or trailing stop of this session has fired. Sent before the order's own execution reports, so a client knows why its stop traded – the FIX analogue is ExecType=L.

Field	Type	Presence	Rules	Description
`client_order_id`	`int`	required	unit `dimensionless`	Echoed from the original NEW_ORDER.
`order_id`	`int`	required	unit `dimensionless`	Session-scoped BALF order id assigned by the gateway.
`stop_price`	`float`	required	gt 0, unit `display_price`	The trigger price it fired at (a trailing stop's ratcheted one), in display money; i64 x 10^8 on the wire.
`trigger_price`	`float`	required	ge 0, unit `display_price`	The price of the trade that reached the trigger; 0 when the last price came from a restart.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 8, pattern `^[A-Z0-9._]+$`	
`order_type`	enum: `STOP`, `STOP_LIMIT`, `TRAILING_STOP`	required	—	The order as it was armed (NEW_ORDER's order_type codes).
`becomes`	enum: `MARKET`, `LIMIT`	required	—	STOP and TRAILING_STOP become MARKET; STOP_LIMIT becomes LIMIT.

See also: `order.triggered.{GW_ID}`, `execution_report`

16.16.4 `order.ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Acknowledge acceptance or rejection of a new order, addressed to the gateway that submitted it.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`order_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 256	Rejection detail; empty when accepted.

Field	Type	Presence	Rules	Description
`reject_code`	enum: `MALFORMED_MESSAGE`, `MISSING_FIELD`, `INVALID_VALUE`, `UNSUPPORTED_FIELD`, `AUTH_REQUIRED`, `AUTH_FAILED`, `ROLE_DENIED`, `NOT_OWNER`, `RATE_LIMITED`, `GATEWAY_NOT_CONFIGURED`, `UNKNOWN_SYMBOL`, `SYMBOL_NOT_READY`, `TICK_VIOLATION`, `LOT_VIOLATION`, `PRICE_OUT_OF_RANGE`, `QTY_OUT_OF_RANGE`, `COLLAR_BREACH`, `MAX_ORDER_QTY`, `MAX_ORDER_VALUE`, `POSITION_LIMIT`, `KILL_SWITCH_ACTIVE`, `MARKET_CLOSED`, `SESSION_NOT_PERMITTED`, `INSTRUMENT_HALTED`, `CIRCUIT_BREAKER_ACTIVE`, `ORDER_NOT_FOUND`, `ORDER_ALREADY_TERMINAL`, `AMEND_NOT_PERMITTED`, `DUPLICATE_ORDER`, `INSUFFICIENT_LIQUIDITY`, `SELF_MATCH_PREVENTED`, `INTERNAL_ERROR`, `UNKNOWN`	omitted when unset	—	Machine-readable rejection classification, stable across every order-entry transport (ALF, BALF, REST) and every layer (gateway-local validation and engine-side business rules). Absent when accepted; present on every rejection. The human-readable detail rides in reason; this field never carries data values. New members may be added; existing members are never removed or renamed. A client must treat an unrecognised code as UNKNOWN.

Field	Type	Presence	Rules	Description
`ts_ns`	`int`	omitted when unset	ge 0, unit `epoch_nanos`	The engine's clock when it decided this order's fate (accepted or refused), in Unix epoch nanoseconds from the same strictly increasing source as trade.executed.ts_ns, so an ack sorts against the prints around it. Absent on a record written before the field existed.
`symbol`	`string`	omitted when unset	max_len 16	
`side`	`string`	omitted when unset	max_len 8	
`order_type`	`string`	omitted when unset	max_len 16	
`tif`	`string`	omitted when unset	max_len 8	
`qty`	`int`	omitted when unset	unit `shares`	
`price`	`float`	omitted when unset	unit `display_price`	Absent for a MARKET order, which has no limit price.
`stop_price`	`float`	omitted when unset	unit `display_price`	STOP / STOP_LIMIT / TRAIL-ING_STOP trigger. Absent for a type that carries none.
`visible_qty`	`int`	omitted when unset	unit `shares`	ICEBERG: fixed peak size. Absent for a type that carries none.

Field	Type	Presence	Rules	Description
`trail_offset`	`float`	omitted when unset	unit `display_price`	TRAILING_STOP: trail distance, in display money. Absent for a type that carries none.
`smp_action`	enum: `NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	omitted when unset	—	Self-match prevention. Null means the client did not specify SMP at all, distinct from an explicit NONE. See SmpAction's docstring.
`client_tag`	`string`	omitted when unset	max_len 64	
`request_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for this request, echoed on the resulting event or rejection. Distinct from the target order's client_tag, which identifies the order rather than the request acting on it. A client may have several requests outstanding against one order.
`oco_group_id`	`string`	omitted when unset	max_len 64	
`combo_parent_id`	`string`	omitted when unset	max_len 64	
`quote_id`	`string`	omitted when unset	max_len 64	
`leg_index`	`int`	omitted when unset	unit `dimensionless`	

Field	Type	Presence	Rules	Description
`is_seed`	`bool`	defaults to `False`	—	True when this order was created by config-driven startup seeding rather than a live gateway submission. See OrderDisplay.is_seed (above) for the full rationale; this is the ack-time echo of the same fact. A seeded order has no order.new (the engine made it, no gateway sent it), so its order.ack is the first audit-visible event for it; a live order's order.new is echoed into the trail first.
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line, or of a window of the trail that begins long after the book snapshot that declared it, can turn it into money without guessing. Absent when the message carries no tick price.

Field	Type	Presence	Rules	Description
`price_ticks`	`ticks`	omitted when unset	unit `ticks`	`price` in engine ticks. Canonical; the float beside it is for display. Scale: the symbol's tick_decimals.
`stop_price_ticks`	`ticks`	omitted when unset	unit `ticks`	`stop_price` in engine ticks. Canonical; the float beside it is for display. Scale: the symbol's tick_decimals.
`trail_offset_ticks`	`ticks`	omitted when unset	unit `ticks`	`trail_offset` in engine ticks. Canonical; the float beside it is for display. Scale: the symbol's tick_decimals.

Note

The order-detail fields are present only when the engine had the order to hand; on a rejection before lookup they are absent.
reason is empty on an acceptance.
See also: `order.fill.{GW_ID}`, `order.new`

16.16.5 `order.fill.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Private fill notification for one order, addressed to the gateway that owns it. The public counterpart is trade.executed.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`order_id`	`string`	required	max_len 64	
`fill_qty`	`int`	required	unit `shares`	
`fill_price`	`float`	required	unit `display_price`	
`remaining_qty`	`int`	required	unit `shares`	

Field	Type	Presence	Rules	Description
`status`	enum: `PARTIAL`, `FILLED`	required	—	Whether this fill completed the order. The same two values BALF's execution_report uses, and the same vocabulary as order.new.status - there is one name for an order state across the system. This was the only status field the spec left as an unconstrained string, and it drifted: six publish sites derived the value from OrderStatus and sent PARTIAL while the continuous-matching hot path and _publish_amend_rematch sent PARTIAL_FILL, so a fill on a quote leg reported a different status from a fill on an ordinary order. Nothing in the system ever branched on PARTIAL_FILL - every consumer passed it through - so the two outliers were corrected rather than the six. BREAKING for anything matching the string: the ALF FILL line, the

Field	Type	Presence	Rules	Description
`symbol`	`string`	omitted when unset	max_len 16	
`side`	`string`	omitted when unset	max_len 8	
`order_type`	`string`	omitted when unset	max_len 16	
`tif`	`string`	omitted when unset	max_len 8	
`qty`	`int`	omitted when unset	unit `shares`	
`price`	`float`	omitted when unset	unit `display_price`	
`client_tag`	`string`	omitted when unset	max_len 64	
`oco_group_id`	`string`	omitted when unset	max_len 64	
`combo_parent_id`	`string`	omitted when unset	max_len 64	
`quote_id`	`string`	omitted when unset	max_len 64	
`leg_index`	`int`	omitted when unset	unit `dimensionless`	
`is_seed`	`bool`	defaults to `False`	—	True when this order was created by config-driven startup seeding rather than a live gateway submission. See OrderDisplay.is_seed for the full rationale; carried here too so a fill reached after the originating order.ack has rotated out of the audit log still says whether the filled order was a config seed.

Field	Type	Presence	Rules	Description
`ts_ns`	`int`	omitted when unset	ge 0, unit `epoch_nanos`	The match time of the trade behind this fill, in Unix epoch nanoseconds: identical to trade.executed.ts_ns (the last trade, when a sweep was coalesced into one fill). A fill and the print it reports carry one clock read. Absent on a record written before the field existed.

Field	Type	Presence	Rules	Description
`trade_ids`	list of `string`	defaults to `[]`	—	The public trade.executed id(s) that composed this fill event. Usually one; more than one when an aggressor swept several resting orders and the engine coalesced them into a single VWAP fill (H5/H6). Every current call site only publishes order_fill from a real trade, so in practice this is never empty; it is typed as a list (not required) defensively, for a fill notification that might one day exist without one. Lets a reader link a private fill to the public trade tape without re-deriving the join.

Field	Type	Presence	Rules	Description
`liquidity_flag`	enum: `MAKER`, `TAKER`	omitted when unset	—	Derived from the trade's aggressor side: the aggressor is the TAKER and the resting side the MAKER (same derivation as drop_copy.yaml::liquidity_flag). Nullable/omit_when_none for the same reason trade_ids is typed as an empty-able list: every current call site only publishes order_fill from a real trade, so this is never actually absent today, but nothing enforces that a future fill notification always has a trade behind it.
`fill_notional_ticks`	sint	omitted when unset	ge 0, unit `notional_ticks`	Σ price_ticks × quantity of the trades in `trade_ids`: the fill's money exactly (`fill_price` is its VWAP, rounded for display).
`cum_qty`	int	omitted when unset	ge 0, unit `shares`	Quantity the order has filled in all, this fill included.
`cum_notional_ticks`	int	omitted when unset	ge 0, unit `notional_ticks`	Σ price_ticks × quantity of every fill of the order so far; with `cum_qty`, the exact average price.

See also: `trade.executed`

16.16.6 `order.cancelled.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Confirm that a resting order has been cancelled.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`order_id`	`string`	required	max_len 64	
`symbol`	`string`	omitted when unset	max_len 16	Instrument this order belonged to. The engine has always had this at the publish site (the cancelled Order's own .symbol); it was simply never carried on the wire, which made order.cancelled invisible to a –symbol filter (AR-0.2). Nullable/omit_when_none, like old_price/old_qty on order.amended below, so archived records written before this field existed still parse.
`client_tag`	`string`	omitted when unset	max_len 64	
`request_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for this cancel request. Engine-initiated cancels publish with request_tag=null.

Field	Type	Presence	Rules	Description
`cancel_reason`	enum: `SELF_MATCH_PREVENTED`, `INSUFFICIENT_LIQUIDITY`, `KILL_SWITCH`, `CIRCUIT_BREAKER_HALT`, `GATEWAY_DISCONNECT`, `ADMIN_CANCEL_SYMBOL`, `QUOTE_REPLACED`, `QUOTE_LEG_FILLED`	omitted when unset	—	Why the exchange cancelled this order, when the exchange decided it rather than the client. Null for a client-requested cancel, and for engine-initiated cancels whose cause is not yet classified - so request_tag=null together with cancel_reason=null still means “the exchange did this, cause unstated”. Deliberately not the same vocabulary as order_ack.reject_code: a cancel is not a rejection, and most reject codes can never apply to one. New members may be added; existing members are never removed or renamed. A client must ignore a value it does not recognise.
`oco_group_id`	`string`	omitted when unset	max_len 64	
`combo_parent_id`	`string`	omitted when unset	max_len 64	
`quote_id`	`string`	omitted when unset	max_len 64	
`leg_index`	`int`	omitted when unset	unit `dimensionless`	

Field	Type	Presence	Rules	Description
`command_id`	`string`	omitted when unset	max_len 64	The admin/kill-switch command_id (see system.admin_action) that caused this cancel, when one exists — lets a post-mortem join this event back to the exact command that triggered it. Null for a client-requested cancel and for engine-initiated cancels with no originating command (e.g. self-match prevention, insufficient liquidity, a quote leg cancelled because its sibling filled).
`remaining_qty`	`int`	omitted when unset	ge 0, unit `shares`	Quantity left unfilled when the order was cancelled.
`cum_qty`	`int`	omitted when unset	ge 0, unit `shares`	Quantity the order had filled before it was cancelled.

16.16.7 `order.expired.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

An order has expired because its time-in-force window ended. The optional reason distinguishes auction-only expiry from end-of-day expiry while remaining compatible with existing consumers.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`order_id`	`string`	required	max_len 64	
`symbol`	`string`	omitted when unset	max_len 16	Instrument this order belonged to. Same AR-0.2 rationale as order.cancelled.symbol above - the engine already has it at the publish site, it was simply never carried.
`client_tag`	`string`	omitted when unset	max_len 64	
`oco_group_id`	`string`	omitted when unset	max_len 64	
`combo_parent_id`	`string`	omitted when unset	max_len 64	
`quote_id`	`string`	omitted when unset	max_len 64	
`leg_index`	`int`	omitted when unset	unit `dimensionless`	
`remaining_qty`	`int`	omitted when unset	ge 0, unit `shares`	Quantity left unfilled when the order expired.
`cum_qty`	`int`	omitted when unset	ge 0, unit `shares`	Quantity the order had filled before it expired.
`reason`	`string`	omitted when unset	—	Why the order expired: “Opening auction ended”, “Closing auction ended”, or “Trading day ended”.

16.16.8 `order.triggered.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.3

A stop, stop-limit or trailing stop has fired: the trade that reached its trigger, and what the order became. Published before the order’s own fills, so a client knows why its stop traded – the FIX

analogue is ExecType=L.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`order_id`	`string`	required	max_len 64	
`symbol`	`string`	required	max_len 16	
`order_type`	enum: `STOP`, `STOP_LIMIT`, `TRAILING_STOP`	required	—	The order as it was armed.
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line, or of a window of the trail that begins long after the book snapshot that declared it, can turn it into money without guessing. Absent when the message carries no tick price.
`stop_price_ticks`	`ticks`	required	unit `ticks`	The trigger price it fired at (a trailing stop's ratcheted one).
`trigger_trade_id`	`string`	omitted when unset	max_len 64	The trade that reached the trigger; absent when the last price came from a restart.
`trigger_price_ticks`	`ticks`	omitted when unset	unit `ticks`	That trade's price.

Field	Type	Presence	Rules	Description
`becomes`	enum: `MARKET`, `LIMIT`	required	—	STOP and TRAILING_STOP become MARKET; STOP_LIMIT becomes LIMIT at its limit.
`client_tag`	`string`	omitted when unset	max_len 64	

See also: `order.fill.{GW_ID}`, `engine.trace.{SYMBOL}`

16.16.9 `engine.trace.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.3

What the book did that no trade or order message shows, for the audit trail (no gateway forwards it): an order resting or arming and the place in the queue it got, an iceberg refilling its slice at the back of its level, a trailing stop ratcheting its trigger. A replay can then compare its model of priority and of trailing triggers with the engine's own, instead of inferring them. Three kinds say why the engine decided (S36): UNCROSS_PRICE (which rule chose an uncross's price, among which tied prices), AON_PREFLIGHT (per leg of an all-or-none combo, what could fill, and whether the combo matches, rests or is rejected) and SMP_DECISION (the same-gateway pair self-match prevention met, and the policy applied).

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`kind`	enum: `RESTED`, `ICEBERG_REFILL`, `TRAILING_RATCHET`, `UNCROSS_PRICE`, `AON_PREFLIGHT`, `SMP_DECISION`	required	—	
`order_id`	`string`	omitted when unset	max_len 64	The order the record is about; SMP_DECISION: the order whose policy applied (the aggressor; in an uncross the bid, unless only the ask has a policy).

Field	Type	Presence	Rules	Description
`priority_seq`	`int`	omitted when unset	ge 0, unit `dimensionless`	
`displayed_qty`	`int`	omitted when unset	ge 0, unit `shares`	ICEBERG_REFILL: the new visible slice.
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line, or of a window of the trail that begins long after the book snapshot that declared it, can turn it into money without guessing. Absent when the message carries no tick price.
`stop_price_ticks`	`ticks`	omitted when unset	unit `ticks`	TRAILING_RATCHET: the new trigger.
`last_price_ticks`	`ticks`	omitted when unset	unit `ticks`	TRAILING_RATCHET: the trade price that moved it.
`auction_id`	`string`	omitted when unset	max_len 32	UNCROSS_PRICE: the uncross, as on its `auction.result`.
`eq_price_ticks`	`ticks`	omitted when unset	unit `ticks`	UNCROSS_PRICE: the price the uncross runs at (it prints nothing there when `eq_qty` is 0: SMP or an OCO pair took the whole overlap).

Field	Type	Presence	Rules	Description
`eq_qty`	`int`	omitted when unset	ge 0, unit `shares`	UNCROSS_PRICE: the executable quantity at that price.
`surplus`	`int`	omitted when unset	ge 0, unit `shares`	UNCROSS_PRICE: the unmatched quantity at that price.
`rule`	enum: `VOLUME`, `SURPLUS`, `LOWEST_PRICE`, `OVERRIDE`	omitted when unset	—	UNCROSS_PRICE: what decided – the most volume (one price had it), then the least surplus among those, then the lowest price; OVERRIDE when the price was imposed (closing backstop, ADMIN_MANUAL).
`tied_prices_ticks`	list of `int`	omitted when empty	—	UNCROSS_PRICE: the prices still tied when `rule` decided, lowest first (one price for VOLUME; the most-volume prices for SURPLUS; those with the least surplus too for LOWEST_PRICE). In ticks.
`combo_id`	`string`	omitted when unset	max_len 64	AON_PREFLIGHT: the client's combo id.
`leg_index`	`int`	omitted when unset	ge 0, unit `dimensionless`	AON_PREFLIGHT: the leg, in the combo's order.
`quantity`	`int`	omitted when unset	ge 0, unit `shares`	AON_PREFLIGHT: the leg's quantity.

Field	Type	Presence	Rules	Description
`fillable_qty`	`int`	omitted when unset	ge 0, unit `shares`	AON_PREFLIGHT: what the leg could fill now, the gateway's own orders left out under its SMP policy.
`fillable_no_smp_qty`	`int`	omitted when unset	ge 0, unit `shares`	AON_PREFLIGHT: what the leg would trade now counting the gateway's own orders (more than 0 on a blocked combo is what rejects it).
`outcome`	enum: `MATCH`, `REST`, `REJECT`	omitted when unset	—	AON_PREFLIGHT: the combo's outcome, the same on every leg – every leg fills in full and the legs match; a leg is short and the legs rest unmatched; or a leg is short and one would trade now, so the combo fails at its first leg (INSUFFICIENT_LIQUIDITY). Halted symbols and phases without matching rest.
`other_order_id`	`string`	omitted when unset	max_len 64	SMP_DECISION: the same gateway's order on the other side.

Field	Type	Presence	Rules	Description
`smp_action`	enum: `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	omitted when unset	—	SMP_DECISION: the policy applied, of `order_id`'s order: which of the two was cancelled (both for CANCEL_BOTH). An unfillable FOK meets each same-gateway order in its range: one record each.

Note

`priority_seq` is the book's arrival sequence; it orders orders at one price, nothing more.
A trailing stop rests in a list checked in arming order, so its RESTED has no `priority_seq`.
UNCROSS_PRICE names no order; AON_PREFLIGHT names the combo and the leg, not an order (the leg's order does not exist yet).
See also: `order.triggered.{GW_ID}`, `order.amended.{GW_ID}`

16.16.10 `order.amended.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Confirm an accepted amendment and report the resulting order.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`order_id`	`string`	required	max_len 64	
`symbol`	`string`	omitted when unset	max_len 16	Instrument this order belonged to. Same AR-0.2 rationale as order.cancelled.symbol above - the engine already has it at the publish site, it was simply never carried.

Field	Type	Presence	Rules	Description
`price`	`float`	`null` when unset	unit `display_price`	New limit price, or null for an order that has none.
`qty`	`int`	required	unit `shares`	
`remaining_qty`	`int`	required	unit `shares`	
`priority_reset`	`bool`	required	—	True when the amendment lost the order its time priority.
`old_price`	`float`	omitted when unset	unit `display_price`	The order's limit price immediately before this amendment, or null for an order that had none. Lets a post-mortem replay the amend without re-deriving the prior state from an earlier message. Absent on records written before this field existed.
`old_qty`	`int`	omitted when unset	unit `shares`	The order's qty immediately before this amendment. Same replay rationale as old_price; absent on records written before this field existed.
`client_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for the amended order.

Field	Type	Presence	Rules	Description
`request_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for this amend request. Engine-initiated cancels publish with request_tag=null.
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line, or of a window of the trail that begins long after the book snapshot that declared it, can turn it into money without guessing. Absent when the message carries no tick price.
`price_ticks`	`ticks`	omitted when unset	unit `ticks`	`price` in engine ticks. Canonical; the float beside it is for display. Scale: the symbol's tick_decimals.
`priority_seq`	`int`	omitted when unset	ge 0, unit `dimensionless`	The order's place in its book after the amendment (the book's arrival sequence): changed when `priority_reset`.

Note

price is nullable but always present: a MARKET order has no limit price, and the field says so with null rather than by being absent.
This is the one message in the group that emits null rather than omitting - it is what the hand-written builder did.

16.16.11 `order.new`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Submit a new order to the matching engine. Sent over PUSH/PULL rather than the pub bus, but it carries a topic so the audit log can classify it alongside everything else.

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 64	Engine order id; a UUID string.
`symbol`	`string`	required	max_len 16	
`side`	enum: `BUY`, `SELL`	required	—	
`order_type`	enum: `MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	required	—	
`tif`	enum: `DAY`, `GTC`, `ATO`, `ATC`	required	—	
`quantity`	`int`	required	gt 0, unit `shares`	Total original quantity.
`remaining_qty`	`int`	required	ge 0, unit `shares`	Quantity yet to be filled; equals quantity on submission.
`gateway_id`	`string`	required	max_len 32	

Field	Type	Presence	Rules	Description
`tick_decimals`	`int`	required	ge 0, le 8, unit `dimensionless`	Decimal scale for every `_ticks` field below; 1 tick = 10^-tick_decimals. Carried so a reader can turn a tick price into money from this message alone. Without it the scale has to be recovered from a `book` or `trade.executed` for the same symbol, which a reader of a *window* of the audit trail may not have - the snapshot that declared it is usually at session start, hours outside the window being read.
`trail_offset_ticks`	`ticks`	`null` when unset	unit `ticks`	TRAILING_STOP: fixed distance to trail the market price.
`oco_group_id`	`string`	`null` when unset	max_len 64	
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	Client-supplied submission time. NOT what the book uses for time priority - see arrival_seq. BALF has no timestamp field on NEW_ORDER, so balf_gwy stamps one at ingress.

Field	Type	Presence	Rules	Description
`status`	enum: `NEW`, `PARTIAL`, `FILLED`, `CANCELLED`, `REJECTED`, `EXPIRED`	required	—	Always NEW on submission; the enum is the full lifecycle.
`price_ticks`	`ticks`	`null` when unset	unit `ticks`	Limit price in ticks. Null for MARKET, which has none.
`stop_price_ticks`	`ticks`	`null` when unset	unit `ticks`	STOP / STOP_LIMIT / TRAILING_STOP trigger.
`visible_qty`	`int`	`null` when unset	unit `shares`	ICEBERG: fixed peak size.
`displayed_qty`	`int`	`null` when unset	unit `shares`	ICEBERG: ignored on input; the engine sets the first slice from visible_qty.
`smp_action`	enum: `NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	`null` when unset	—	Self-match prevention. Null means the client did not specify SMP at all, which is distinct from an explicit NONE: the engine resolves null to the gateway's configured default. See SmpAction's docstring.
`combo_parent_id`	`string`	`null` when unset	max_len 64	
`leg_index`	`int`	`null` when unset	unit `dimensionless`	Position in the parent combo's legs, 0-based.
`origin`	enum: `ORDER`, `QUOTE`, `IMPLIED`	defaults to `'ORDER'`	—	Defaulted rather than nullable: from_dict supplies ORDER.

Field	Type	Presence	Rules	Description
`is_seed`	`bool`	defaults to `False`	—	True when this order was created by config-driven startup seeding rather than a live gateway submission. See OrderDisplay.is_seed for the full rationale.
`quote_id`	`string`	`null` when unset	max_len 64	
`client_tag`	`string`	`null` when unset	max_len 64	Client correlation tag, echoed on every lifecycle event.
`arrival_seq`	`int`	defaults to `0`	unit `dimensionless`	Engine-assigned monotonic arrival sequence; time priority is keyed on this, not on timestamp, so a back-dated payload cannot jump the queue. Zero means unassigned, which is what a submission carries.

Note

The payload is exactly Order.to_dict().
Eleven fields are nullable and are emitted as null when unset rather than omitted - a MARKET order carries "price_ticks": null.
The engine's Order.from_dict reads absent and null alike, so a producer that omits them is still accepted.
See also: `order.ack.{GW_ID}`, `order.cancel`, `order.amend`

16.16.12 `order.cancel`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Request cancellation of one resting order by id.

Field	Type	Presence	Rules	Description
`order_id`	`string`	required	max_len 64	
`gateway_id`	`string`	required	max_len 32	
`request_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for this cancel request, echoed on the resulting cancellation event or rejection.

See also: `order.cancelled.{GW_ID}`

16.16.13 `order.amend`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Request a price and/or quantity change to a resting order.

Field	Type	Presence	Rules	Description
`order_id`	`string`	required	max_len 64	
`gateway_id`	`string`	required	max_len 32	
`price`	`float`	omitted when unset	unit `display_price`	New limit price; absent means the price is unchanged.
`qty`	`int`	omitted when unset	unit `shares`	New quantity; absent means the quantity is unchanged.
`request_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for this amend request, echoed on the resulting amendment event or rejection.

Note

price and qty are omitted when not being changed, and the engine reads that absence as “leave this alone” - so unlike order.new these two DO take omit_when_none.
tests/test_messages.py pins the omission directly.
price here is display money, not the ticks that order.new carries.
See also: `order.amended.{GW_ID}`

16.16.14 `order.combo_cancel`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Cancel a combo order and all of its resting child legs.

Field	Type	Presence	Rules	Description
`combo_id`	`string`	defaults to `''`	max_len 64	Client-supplied combo label, not the internal UUID.
`gateway_id`	`string`	defaults to `''`	max_len 32	
`request_tag`	`string`	omitted when empty	max_len 64	Echoed on combo.cancel_ack, so the caller can attribute the outcome.

See also: `order.combo`, `order.cancel`

16.16.15 `order.combo`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Submit a combo: two or more orders on different instruments that the engine posts together and tracks as one aggregate.

Field	Type	Presence	Rules	Description
`combo_id`	`string`	required	max_len 64	Client-supplied tracking label, not the engine's internal id.
`gateway_id`	`string`	required	max_len 32	
`combo_type`	enum: `AON`	required	—	All-or-none: the combo completes only when every leg fills.
`tif`	enum: `DAY`, `GTC`, `ATO`, `ATC`	required	—	

Field	Type	Presence	Rules	Description
`legs`	list of `ComboLeg`	required	min_items 2, max_items 10	The child orders. The bounds below were previously enforced only by api_gateway's pydantic schema, which left the ALF console and gateway free to submit a one-legged combo.
`client_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for this combo submission.

Note

This is the message the whole of design section 15 is about.
It was unspecifiable for three separate reasons, and each turned out to be the wire being wrong rather than the IDL being short: leg prices whose unit depended on their runtime type (15.2), engine lifecycle state riding on a client submission (15.4), and finally the ordinary need for `nested` and `list[T]` (15.5).
The payload below is 'ComboOrder.
to_submission_dict()' and carries no engine state at all.
See also: `combo.ack.{GW_ID}`, `order.combo_cancel`

16.16.16 `order.oco`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Submit a One-Cancels-Other pair: two orders on the same instrument, of which a fill on either cancels the other.

Field	Type	Presence	Rules	Description
`oco_id`	`string`	defaults to `''`	max_len 64	Client-supplied label for the pair, unique among the gateway's live pairs: a submit reusing one is refused. Free again once the pair is done.
`gateway_id`	`string`	defaults to `''`	max_len 32	
`symbol`	`string`	defaults to `''`	max_len 16	Both legs trade this instrument.
`tick_decimals`	`int`	defaults to `0`	ge 0, le 8, unit `dimensionless`	Decimal scale for both legs' `_ticks` prices; 1 tick = 10^-tick_decimals. One value for the pair rather than one per leg, because both legs trade the instrument named above.
`quantity`	`int`	defaults to `0`	unit `shares`	Size of each leg; they are equal by construction.
`tif`	enum: `DAY`, `GTC`, `ATO`, `ATC`	defaults to `'DAY'`	—	
`leg1`	`OcoLeg`	required	—	
`leg2`	`OcoLeg`	required	—	
`client_tag`	`string`	omitted when unset	max_len 64	Client correlation tag for this OCO submission.

Note

The first message in any spec to use a nested record.
Both legs are `OcoLeg`, and their prices are engine ticks at `tick_decimals` - the gateway converts.
A leg omits a price it does not have rather than sending null, which is what the three producing gateways already do.
See also: `oco.ack.{GW_ID}`, `order.oco_cancel`

16.16.17 `order.oco_cancel`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Cancel an OCO pair and both of its legs.

Field	Type	Presence	Rules	Description
`oco_id`	`string`	defaults to `''`	max_len 64	Client-supplied OCO label. Names a pair of the sending gateway only.
`gateway_id`	`string`	defaults to `''`	max_len 32	
`request_tag`	`string`	omitted when empty	max_len 64	Echoed on oco.cancel_ack, so the caller can attribute the outcome.

See also: `order.oco`, `order.cancel`

16.16.18 `order.orders_request`

Published by: `admin`, `api_gateway`, `gateway`

Transport: `engine_push`

Since: 1.0

Caller to engine: the resting (unfilled, non-cancelled) orders a gateway currently has on the books, across all symbols.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Whose resting orders to return, and the reply's correlation key.

See also: `order.orders.{GW_ID}`, `system.symbols_request`

16.16.19 `order.orders.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to caller: the gateway's resting orders in display units, one OrderDisplay record each. Empty when the gateway is unknown or flat.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped from the body by the default projection.
`orders`	list of `OrderDisplay`	required	—	Resting orders, as the engine iterates its books.

Note

gateway_id names the caller in the topic and is dropped from the body by the default projection, so the body is a single `orders` list - the same shape system.symbols uses.
See also: `order.orders_request`, `order.new`

16.16.20 `order.price_level_orders_request`

Published by: `admin`

Transport: `engine_push`

Since: 1.0

ADMIN to engine: every resting order for one symbol, across every gateway, optionally narrowed to a single price level. Rejected for any non-ADMIN participant (see order.price_level_orders' rejection note) since it exposes other participants' resting order detail that order.orders_request deliberately withholds.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	The ADMIN participant asking, and the reply's correlation key — not a filter on whose orders come back.
`symbol`	`string`	required	max_len 16	Instrument to inspect.
`price`	`float`	omitted when unset	unit `display_price`	Narrow to orders resting at exactly this price. Omitted or null returns every resting order for the symbol, across all price levels.

See also: `order.price_level_orders.{GW_ID}`, `order.orders_request`, `book.{symbol}`

16.16.21 `order.price_level_orders.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN caller: per-order detail (not just the aggregate {price, qty, count} book.* already carries) for every resting order matching the request — every gateway, ordered by price then by arrival_seq within a price level so time priority is visible. Empty, with rejected=true and a reason, when the requester is not an ADMIN participant or the symbol is unknown.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped from the body by the default projection.
`symbol`	`string`	required	max_len 16	
`price`	`float`	omitted when unset	unit `display_price`	Echoed from the request when it filtered to one level; omitted when the request asked for the whole symbol.
`rejected`	`bool`	required	—	True when the requester was not ADMIN or the symbol is unknown; orders is then always empty.
`reason`	`string`	omitted when unset	max_len 256	Set only when rejected is true.
`orders`	list of `PriceLevelOrder`	required	—	Matching resting orders, ordered by price then arrival_seq — empty when rejected, or when nothing rests at the requested level.

See also: `order.price_level_orders_request`

16.17 Family `quote`

16.17.1 `quote.new`

Published by: `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Market maker to engine: submit or replace a two-sided quote on one instrument. A quote is a bid and an ask posted as a pair; replacing one cancels the previous pair for that gateway and symbol.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`tick_decimals`	`int`	required	ge 0, le 8, unit `dimensionless`	Decimal scale for both `_ticks` prices; 1 tick = 10^-tick_decimals. Carried so a reader can turn a tick price into money from this message alone, rather than recovering the scale from a `book` for the same symbol that may be outside the window being read.
`bid_price_ticks`	`ticks`	required	gt 0, unit `ticks`	
`bid_qty`	`int`	required	gt 0, unit `shares`	
`ask_price_ticks`	`ticks`	required	gt 0, unit `ticks`	
`ask_qty`	`int`	required	gt 0, unit `shares`	

Field	Type	Presence	Rules	Description
`tif`	enum: `DAY`, `GTC`, `ATO`, `ATC`	defaults to `'DAY'`	—	Applies to both legs; the engine reads it once. Same four values as mod-els/order.py::TIF and order.combo's own tif - a quote's legs are ordinary orders once they rest.
`quote_id`	`string`	omitted when empty	max_len 64	

Note

Prices are integer ticks.
The engine rejects a float outright rather than converting it, because a display price of 150.0 accepted as 150 ticks would post the quote at 1/100th of the intended level on a two-decimal instrument - silent, and in the wrong direction for the side that gets hit.
`quote_id` is the client's own handle, echoed on every ack and status event.
It is optional: a gateway that submits one quote per symbol can identify it by symbol alone, and the hand-written builders omitted the key entirely rather than sending " ".
See also: `quote.ack.{GW_ID}`, `quote.cancel`, `quote.status.{GW_ID}`

16.17.2 `quote.cancel`

Published by: `admin`, `api_gateway`, `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Market maker to engine: pull the active quote on one symbol.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`request_tag`	`string`	omitted when empty	max_len 64	Echoed on quote.cancel_ack, so the caller can attribute the outcome.

Note

Addressed by symbol rather than by quote_id: a gateway has at most one active quote per instrument, so the pair identifies it.
That is also why quote_id is optional on submission.

See also: `quote.new`

16.17.3 `quote.ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to market maker: the quote was accepted or rejected.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`quote_id`	`string`	defaults to `''`	max_len 64	
`accepted`	`bool`	required	—	
`symbol`	`string`	defaults to `''`	max_len 16	Instrument the quote was for. `order.ack` has always carried this; `quote.ack` did not, which left every market-making client to work out for itself which symbol an ack belonged to – the topic is per-gateway, not per-symbol. Empty only on a rejection raised before the symbol was known (a malformed payload, or a missing SYM).
`reason`	`string`	defaults to `''`	max_len 512	
`bid_order_id`	`string`	defaults to `''`	max_len 64	
`ask_order_id`	`string`	defaults to `''`	max_len 64	

Note

The two order ids are the engine's handles for the resting legs, and are what ties a subsequent order.fill to the quote that produced it.
Both are always emitted, as " " on rejection - the hand-written builder put them in the base payload, and a rejected quote rests nothing.

16.17.4 `quote.status.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to market maker: the quote left the book, and why.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`quote_id`	`string`	defaults to `''`	max_len 64	
`status`	enum: `ACTIVE`, `INACTIVE_BID_FILLED`, `INACTIVE_ASK_FILLED`, `CANCELLED`	required	—	Mirrors models/quote.py::QuoteState.
`reason`	`string`	defaults to `''`	max_len 512	
`command_id`	`string`	omitted when unset	max_len 64	The admin/kill-switch command_id (see system.admin_action) that caused this quote to leave the book, when one exists — same correlation purpose as order.cancelled.command_id. Null when the quote left for a reason with no originating command (participant cancel, replacement by a new quote, a leg fill).

Note

The two INACTIVE_* states say which side was hit, which a market maker needs in order to re-quote the other one.
They are distinct states rather than one INACTIVE plus a side field because models/quote.py::QuoteState is what the engine actually holds, and a wire that renames its own state machine is a translation nobody asked for.
See also: `quote.ack.{GW_ID}`

16.17.5 `quote.cancel_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to market maker: the outcome of a quote.cancel.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`quote_id`	`string`	defaults to `''`	max_len 64	The withdrawn quote's id; " " when there was none.
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`reject_code`	`string`	omitted when empty	max_len 32	On a refusal, a member of order.ack's `reject_code` vocabulary (ORDER_NOT_FOUND, ORDER_ALREADY_TERMINAL, GATEWAY_NOT_CONFIGURED, AUTH_REQUIRED); absent when accepted. A string rather than that enum only because the IDL has no shared enum type: the engine is held to the vocabulary by tests/test_engine_structure_canc
`request_tag`	`string`	omitted when empty	max_len 64	Echoed from the cancel request, so the caller can attribute the outcome.
`cancelled_order_ids`	list of `string`	defaults to `[]`	—	The engine order ids the cancel removed; empty on a refusal.

Note

Every cancel gets exactly one of these, accepted or refused, so a caller (REST's `?wait=ack`) can wait for its own outcome.
Before WP-API17 a refusal rode the submit ack and a success had no group-level reply at all, so neither could be attributed to the request that asked.

16.18 Family `risk`

16.18.1 `risk.kill_switch`

Published by: `admin`, `api_gateway`, `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Gateway or admin to engine: cancel a gateway's open risk-bearing exposure. The gateway is NOT halted - it may submit again as soon as the ack arrives.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	defaults to `''`	max_len 16	Scope to one instrument; " " cancels across all of them.
`note`	`string`	omitted when empty	max_len 256	Free-text reason, recorded on the admin monitor.
`command_id`	`string`	omitted when empty	max_len 64	Echoed on the ack. A kill-switch ack carries no natural identifier - unlike the symbol acks, which carry `symbol` - so without this two concurrent mass cancels for one gateway are indistinguishable once both acks are in flight.

Note

`symbol` scopes the cancel to one instrument.
It is always emitted, as " " for the whole-gateway case, because the handler reads it as `if symbol_filter:` - empty and absent mean the same thing to it, and the hand-written builder always sent the key.

`gateway_id` names whose exposure is cancelled, not who asked.
This message only ever acts on the caller's own gateway; use risk.kill_switch_gateway for one participant acting on another.
See also: `risk.kill_switch_ack.{GW_ID}`, `risk.kill_switch_gateway`

16.18.2 `risk.kill_switch_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to caller: what the kill switch cancelled.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`cancelled_orders`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`cancelled_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_orders, so a caller can tell which orders were hit rather than only how many. A list whose length disagrees with cancelled_orders is itself a detectable defect.
`cancelled_quotes`	`int`	defaults to `0`	ge 0, unit `dimensionless`	

Field	Type	Presence	Rules	Description
`cancelled_quote_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_quotes. Named "*_order_ids" rather than "*_quote_ids" on purpose: cancelled_quotes has always counted cancelled quote LEGS, each an independent Order with its own order_id, not quote objects — a quote's own quote_id is client-supplied and optional (see quote.yaml), so it cannot serve as this list's key even when present.
`command_id`	`string`	omitted when empty	max_len 64	

Note

The counters are always emitted, as 0 on rejection: the hand-written builder put them in the base payload beside `accepted`, and a rejected kill switch cancelled nothing.
`reason` is likewise always present, as " " on success.

16.18.3 `risk.kill_switch_gateway`

Published by: `api_gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: cancel every order and quote belonging to one named participant.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	The ADMIN caller; the ack is addressed to this id.

Field	Type	Presence	Rules	Description
`target_gateway_id`	`string`	required	max_len 32	Whose orders and quotes are cancelled.
`symbol`	`string`	omitted when empty	max_len 16	Scope the cancel to one instrument; omitted cancels across all.
`quotes_only`	`bool`	defaults to `False`	—	Cancel only the target's quotes, leaving its other resting orders. With `symbol`, this is an admin pulling one market maker's quote on one instrument – through this ADMIN message rather than by sending the participant's own quote.cancel, which would put the target's name on the audit trail and fail while it is disconnected.
`note`	`string`	omitted when empty	max_len 256	
`command_id`	`string`	omitted when empty	max_len 64	

Note

The one message in this group where the caller and the affected gateway are allowed to differ. `gateway_id` is the ADMIN making the request - it is what the role and connection checks run against, and what addresses the ack - while `target_gateway_id` is whose exposure is cancelled. Two fields rather than one because they are two different participants, which is exactly what risk.kill_switch cannot express.

See also: `risk.kill_switch`, `risk.kill_switch_global`

16.18.4 `risk.kill_switch_gateway_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: what the gateway-targeted kill switch cancelled.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`target_gateway_id`	`string`	defaults to `''`	max_len 32	
`symbol`	`string`	omitted when empty	max_len 16	Echoed when the cancel was scoped.
`quotes_only`	`bool`	defaults to `False`	—	Echoed: whether only quotes were cancelled.
`reason`	`string`	defaults to `''`	max_len 512	
`cancelled_orders`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`cancelled_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_orders, so a caller can tell which orders were hit rather than only how many. A list whose length disagrees with cancelled_orders is itself a detectable defect.
`cancelled_quotes`	`int`	defaults to `0`	ge 0, unit `dimensionless`	

Field	Type	Presence	Rules	Description
`cancelled_quote_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_quotes. Named "*_order_ids" rather than "*_quote_ids" on purpose: cancelled_quotes has always counted cancelled quote LEGS, each an independent Order with its own order_id, not quote objects — a quote's own quote_id is client-supplied and optional (see quote.yaml), so it cannot serve as this list's key even when present.
`command_id`	`string`	omitted when empty	max_len 64	

Note

`target_gateway_id` is echoed in the body, unlike `gateway_id`, which the topic already carries. The two are different participants here, so an ack naming only the topic's id would not say who was actually acted on.

16.18.5 `risk.kill_switch_global`

Published by: `api_gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: cancel every resting order and quote, for every gateway. The full-market emergency stop.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	The ADMIN caller; the ack is addressed to this id.
`note`	`string`	omitted when empty	max_len 256	
`command_id`	`string`	omitted when empty	max_len 64	

Note

Distinct from risk.circuit_breaker_halt_all, which halts trading but leaves resting orders in place. This one cancels them outright, and does not halt anything - a gateway may submit again immediately.
See also: `risk.circuit_breaker_halt_all`

16.18.6 `risk.kill_switch_global_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: what the market-wide kill switch cancelled.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`cancelled_orders`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`cancelled_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_orders, so a caller can tell which orders were hit rather than only how many. A list whose length disagrees with cancelled_orders is itself a detectable defect.
`cancelled_quotes`	`int`	defaults to `0`	ge 0, unit `dimensionless`	

Field	Type	Presence	Rules	Description
`cancelled_quote_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_quotes. Named "*_order_ids" rather than "*_quote_ids" on purpose: cancelled_quotes has always counted cancelled quote LEGS, each an independent Order with its own order_id, not quote objects — a quote's own quote_id is client-supplied and optional (see quote.yaml), so it cannot serve as this list's key even when present.
`affected_gateways`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`affected_gateway_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the gateway ids counted in affected_gateways — every gateway that had at least one order or quote leg cancelled by this sweep.
`command_id`	`string`	omitted when empty	max_len 64	

Note

`affected_gateways` is what distinguishes this ack from the other two: the same two counters, plus how many participants they were spread across.

16.18.7 `risk.band.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.3

A symbol's collar, as the engine will apply it: published once per collared symbol at startup (a reference reload leaves collars as they are). The static band is fixed from these; the dynamic band is `dynamic_band_pct` around each last trade price, so a reader derives it from the trades rather than from a message per trade. Circuit-breaker corridors travel on circuit_breaker.halt/extend.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`kind`	enum: `COLLAR`	required	—	
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line can turn it into money without guessing.
`reference_price_ticks`	`ticks`	required	unit `ticks`	The static band's reference (last close or the config seed).
`static_low_ticks`	`ticks`	required	unit `ticks`	
`static_high_ticks`	`ticks`	required	unit `ticks`	
`dynamic_band_pct`	`float`	required	ge 0, unit `dimensionless`	The dynamic band's half-width as a fraction of the last trade price (0.02 = 2 %).

See also: `circuit_breaker.halt.{SYMBOL}`, `system.startup`

16.18.8 `risk.symbol_halt`

Published by: `admin`, `api_gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: halt trading on one instrument.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`level`	`string`	omitted when empty	max_len 32	A configured circuit-breaker level; omitted halts indefinitely.
`note`	`string`	omitted when empty	max_len 256	
`command_id`	`string`	omitted when empty	max_len 64	

Note

With `level` naming one of the symbol's configured circuit_breaker.levels, the halt runs through the same CircuitBreakerState.activate() state machine a price-triggered halt uses - so it gets a real resume_at_ns and ACE corridor, and the normal circuit-breaker tick resumes it.
Without one the halt is indefinite, cleared only by an explicit risk.symbol_resume.
The engine rejects a level for a symbol that has no circuit breaker configured.
See also: `risk.symbol_resume`, `risk.circuit_breaker_halt_all`

16.18.9 `risk.symbol_halt_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: the per-symbol halt's outcome.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`symbol`	`string`	defaults to `''`	max_len 16	
`reason`	`string`	defaults to `''`	max_len 512	
`cancelled_quotes`	`int`	defaults to `0`	ge 0, unit `dimensionless`	Quotes pulled because the instrument stopped trading.

Field	Type	Presence	Rules	Description
`cancelled_quote_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_quotes. Named "*_order_ids" rather than "*_quote_ids" on purpose: cancelled_quotes has always counted cancelled quote LEGS, each an independent Order with its own order_id, not quote objects — a quote's own quote_id is client-supplied and optional (see quote.yaml), so it cannot serve as this list's key even when present.
`command_id`	`string`	omitted when empty	max_len 64	

Note

`symbol` is echoed in the body.
It is what tells two concurrent halt acks for one gateway apart - the reason risk.kill_switch needed a command_id and this did not.

16.18.10 `risk.symbol_resume`

Published by: `admin`, `api_gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: resume trading on one halted instrument.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`note`	`string`	omitted when empty	max_len 256	
`command_id`	`string`	omitted when empty	max_len 64	

Field	Type	Presence	Rules	Description

See also: `risk.symbol_halt`

16.18.11 `risk.symbol_resume_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: the per-symbol resume's outcome.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`symbol`	`string`	defaults to `''`	max_len 16	
`reason`	`string`	defaults to `''`	max_len 512	
`command_id`	`string`	omitted when empty	max_len 64	

Note

No counters: a resume cancels nothing.
The one ack in this family that carries only an outcome and the symbol it applies to.

16.18.12 `risk.cancel_symbol`

Published by: `admin`, `api_gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: cancel every resting order for one instrument, across every gateway.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`note`	`string`	omitted when empty	max_len 256	
`command_id`	`string`	omitted when empty	max_len 64	

Note

The instrument-scoped counterpart to risk.kill_switch's gateway-scoped sweep: that one cancels one gateway across all symbols, this one cancels one symbol across all gateways.
Neither halts anything - trading may resume immediately.
See also: `risk.symbol_halt`, `risk.kill_switch`

16.18.13 `risk.cancel_symbol_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: what the symbol-wide mass cancel removed.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`symbol`	`string`	defaults to `''`	max_len 16	
`reason`	`string`	defaults to `''`	max_len 512	
`cancelled_orders`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`cancelled_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_orders, so a caller can tell which orders were hit rather than only how many. A list whose length disagrees with cancelled_orders is itself a detectable defect.
`cancelled_quotes`	`int`	defaults to `0`	ge 0, unit `dimensionless`	

Field	Type	Presence	Rules	Description
`cancelled_quote_order_ids`	`list of string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_quotes. Named "*_order_ids" rather than "*_quote_ids" on purpose: cancelled_quotes has always counted cancelled quote LEGS, each an independent Order with its own order_id, not quote objects — a quote's own quote_id is client-supplied and optional (see quote.yaml), so it cannot serve as this list's key even when present.
`command_id`	`string`	omitted when empty	max_len 64	

16.18.14 `risk.force_uncross`

Published by: `admin`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: force one symbol to uncross now, optionally asserting the print price, and clear any halt in the same step. The operational recovery for an auction that failed to discover a price - see risk.cancel_symbol for the "abandon and re-collect" alternative.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`price`	`float`	omitted when unset	gt 0, unit `display_price`	Operator print price; null opens at the computed equilibrium.

Field	Type	Presence	Rules	Description
`dry_run`	`bool`	defaults to `False`	—	Peek the indicative print without changing any state.
`note`	`string`	omitted when empty	max_len 256	Free-text reason, recorded on the admin monitor.
`command_id`	`string`	omitted when empty	max_len 64	

Note

`price` is null to open at the naturally computed equilibrium, or a value to assert an operator opening price when no natural equilibrium exists (a failed-auction recovery).
It is nullable rather than a sentinel because "no price given" and "a price of zero" are different requests, exactly as auction.result's `eq_price` is null-not-zero.
`dry_run` peeks the indicative print without mutating any state: no halt is cleared, no uncross runs. It is the on-demand replacement for the auction.indicative feed, which is push-only and skips halted symbols, so an operator would otherwise set `price` blind.
See also: `risk.symbol_resume`, `risk.cancel_symbol`, `auction.result.{SYMBOL}`

16.18.15 `risk.force_uncross_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: the outcome of a force-uncross. On a dry run it carries the indicative figures only; on a live run it also reports what actually printed.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`symbol`	`string`	defaults to `''`	max_len 16	
`reason`	`string`	defaults to `''`	max_len 512	
`dry_run`	`bool`	defaults to `False`	—	True when this ack answers a peek rather than a live uncross.
`indicative_price`	`float`	`null` when unset	gt 0, unit `display_price`	Indicative equilibrium, or null if the book would not cross.

Field	Type	Presence	Rules	Description
`indicative_qty`	`int`	defaults to `0`	ge 0, unit `shares`	Quantity that would execute at the indicative price.
`surplus`	`int`	defaults to `0`	ge 0, unit `shares`	Surplus on `imbalance_side`; zero when balanced.
`imbalance_side`	enum: `BUY`, `SELL`	omitted when unset	—	Which side would be left unfilled; absent when balanced.
`printed_price`	`float`	`null` when unset	gt 0, unit `display_price`	Live-run print price, or null on a dry run or rejection.
`traded_qty`	`int`	defaults to `0`	ge 0, unit `shares`	Quantity that actually executed on a live run.
`command_id`	`string`	omitted when empty	max_len 64	

Note

`indicative_*` mirror auction.indicative: `indicative_price` is null when the book would not cross. `printed_price` and `traded_qty` are the live-run outcome and stay at null/zero on a dry run or a rejection.

16.18.16 `risk.circuit_breaker_halt_all`

Published by: `admin`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: halt trading on every known symbol.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

Note

Halts, and leaves resting orders in place.
risk.kill_switch_global is the other market-wide stop and does the opposite - it cancels everything and halts nothing.
The pair is the reason both exist.
Carries no note and no command_id, on either side.
See the block comment above this group.

See also: `risk.circuit_breaker_resume_all`, `risk.kill_switch_global`

16.18.17 `risk.circuit_breaker_halt_all_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: how wide the market-wide halt reached.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`halted_symbols`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`halted_symbol_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the symbols counted in halted_symbols, by name.
`cancelled_quotes`	`int`	defaults to `0`	ge 0, unit `dimensionless`	

Field	Type	Presence	Rules	Description
`cancelled_quote_order_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the order ids counted in cancelled_quotes. Named "*_order_ids" rather than "*_quote_ids" on purpose: cancelled_quotes has always counted cancelled quote LEGS, each an independent Order with its own order_id, not quote objects — a quote's own quote_id is client-supplied and optional (see quote.yaml), so it cannot serve as this list's key even when present.

16.18.18 `risk.circuit_breaker_resume_all`

Published by: `admin`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: resume every symbol halted by the market-wide halt.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

See also: `risk.circuit_breaker_halt_all`

16.18.19 `risk.circuit_breaker_resume_all_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: how many symbols came back.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`resumed_symbols`	`int`	defaults to `0`	ge 0, unit `dimensionless`	
`resumed_symbol_ids`	list of `string`	defaults to `[]`	—	AR-0.5: the symbols counted in resumed_symbols, by name.

Note

No cancelled_quotes counterpart: resuming pulls nothing.
The halt ack and the resume ack are deliberately different shapes because they report different things, not one shape with a field left at zero.

16.19 Family `session`

16.19.1 Record types

`NextTransition`

The transition after this one: what the session moves to, and when. Present only when the scheduler drove the transition, since it alone knows the day's timetable. A manual or admin-driven transition carries no next - deliberately, because the schedule says what *should* happen while the engine decides what *does*, and a countdown derived from the timetable alone would tick toward a transition nobody will perform. The two fields are one record rather than two optional keys because neither is meaningful alone: a phase without a time cannot be counted down to, and a time without a phase does not say what happens.

Field	Type	Presence	Rules	Description
`state`	`string`	required	max_len 32	The session state being moved to.
`at`	`string`	required	max_len 32	UTC ISO-8601 instant of the transition.

`ReplyTo`

Where to send the outcome of a command, and under what correlation id. Supplied by an *interactive* requester that wants to know what happened. pm-scheduler omits it: it drives the timetable, has nobody to report back to, and the public session.state broadcast already says what occurred. A record rather than two optional keys for the same reason as NextTransition - a command_id with no gateway to answer on is undeliverable, and a gateway with no command_id cannot be correlated.

Field	Type	Presence	Rules	Description
`command_id`	`string`	required	max_len 64	
`gateway_id`	`string`	required	max_len 32	

16.19.2 `session.state`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Broadcast the engine's current session state to every subscriber. The most widely consumed topic in the system.

Field	Type	Presence	Rules	Description
`state`	`string`	required	max_len 32	The session state now in effect.
`prev_state`	`string`	omitted when empty	max_len 32	The state departed from; absent on the first broadcast.
`next`	`NextTransition`	omitted when unset	—	

Field	Type	Presence	Rules	Description
`command_id`	`string`	omitted when empty	max_len 64	AR-0.6: echoes the session.transition.reply_to.co that caused this broadcast, when the requester supplied one. Absent for a schedule-driven transition (pm-scheduler sends no reply_to at all) and for a rejected request (no broadcast happens at all in that case). Converts the engine's own STRONG session.transition -> session.state link (adjacency in the stream) into a CERTAIN one (a shared id), the same way the risk family's acks already do for kill-switch/circuit-breaker commands. Deliberately just the command_id, not the full reply_to: gateway_id is not repeated here – session_transition_ack (addressed to that one gateway) is still the only place accepted, reason and to_state travel, since those are meaningless broadcast to every subscriber. A bare

Note

prev_state is omitted when empty rather than emitted as " ", which is what the hand-written payload did.
next is present only on a scheduler-driven transition.
See also: `session.transition`

16.19.3 `session.transition`

Published by: `scheduler`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Scheduler or operator to engine: request a state change.

Field	Type	Presence	Rules	Description
`to_state`	`string`	required	max_len 32	
`next`	`NextTransition`	omitted when unset	—	Describes the transition *after* this one, so the engine can publish a countdown target. A manual transition omits it, which clears any stale target the engine was holding.
`reply_to`	`ReplyTo`	omitted when unset	—	

See also: `session.transition_ack.{GW_ID}`, `session.state`

16.19.4 `session.next`

Published by: `scheduler`

Transport: `engine_push`

Since: 1.2

Scheduler to engine: announce the next transition without performing one. A `session.transition` can only describe the transition after itself, so a scheduler that starts while the exchange is closed, or before the day's first transition, would otherwise leave the engine with no countdown target until that transition fires – possibly many hours away.

Field	Type	Presence	Rules	Description
`next`	`NextTransition`	required	—	

Note

Not broadcast: the engine only stores the target, and a caller reads it back from `system.session_status`.
Re-publishing `session.state` for a session that has not changed would look like a transition to every subscriber.
The engine ignores an announcement whose state is not a legal transition from its current one.
See also: `session.transition`, `system.session_status.{GW_ID}`

16.19.5 `session.transition_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to the requesting gateway: the outcome of a transition request.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`command_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`to_state`	`string`	defaults to `''`	max_len 32	
`reason`	`string`	defaults to `''`	max_len 256	Rejection detail; empty when accepted.

Note

Addressed rather than broadcast: accepted, reason and to_state are meaningful only to the one gateway that asked, and broadcasting a rejection's reason to every subscriber would be noise at best.
It also closes a silent failure: a request the engine discarded previously produced no reply at all, so a caller could not tell a rejection from a timeout.
This used to be the whole story for command_id too, on the theory that broadcasting it would hand every subscriber another operator's correlation id.
AR-0.6 revisited that: session.state now echoes just the command_id (never gateway_id, accepted, or reason) when one caused the transition, because pm-audit-replay needs a CERTAIN link from session.transition to session.state and adjacency in the stream (a STRONG link) is not enough to build one.
A bare id is a small enough disclosure to be worth that; the rest of the reply still is not, so it stays here.

16.20 Family `sim`

16.20.1 Record types

`NewsRequest`

A headline the instructor wants published (NEWS_INJECT).

Field	Type	Presence	Rules	Description
`scope`	enum: `SYMBOL`, `SECTOR`, `MARKET`	required	—	
`targets`	list of `string`	required	—	Symbols (SYMBOL) or sector names (SECTOR); empty for MARKET.
`kind`	enum: `EARNINGS`, `GUIDANCE`, `MNA`, `REGULATORY`, `PRODUCT`, `LEGAL`, `MANAGEMENT`, `MACRO`, `OPERATIONS`, `RESEARCH`, `CONTRACT`, `ESG`	required	—	
`sentiment`	`float`	required	ge -1, le 1, unit `dimensionless`	
`impact`	`float`	required	ge -1, le 1, unit `dimensionless`	Move of the log true value when the news is (or becomes) confirmed: 0.10 is about +10.5%. Sector and market news move each symbol by its beta to that factor. Never published.
`headline`	`string`	defaults to `''`	max_len 200	Empty: pm-market-sim writes one from the kind and sentiment.
`rumour`	`bool`	defaults to `False`	—	Publish as a RUMOUR; the value moves only on NEWS_CONFIRM.
`credibility`	`float`	omitted when unset	ge 0, le 1, unit `dimensionless`	

SymbolValue

One symbol's true value at the step's simulated time.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`value`	`float`	required	gt 0, unit `display_price`	Display money, like every price on the engine bus.

16.20.2 `sim.value`

Published by: `market_sim`

Transport: `sim_pub`

Since: 1.0

pm-market-sim to the bots, the market makers that anchor to it and the instructor's tools: every symbol's true value after one model step.

Field	Type	Presence	Rules	Description
`seq`	`int`	required	gt 0, unit `dimensionless`	Step counter; continues across a pm-market-sim restart.
`ts_ns`	`int`	required	unit `epoch_nanos`	Publication time.
`values`	list of `SymbolValue`	required	min_items 1	

Note

Batched: one message per step carries every symbol (300 symbols are about 11 KB), so a subscriber sees one consistent cross-section rather than 300 messages from different instants.
Published during continuous trading only: that is when the model steps.
A trading day's variance accrues over the continuous phase as the engine reports it, so a day compressed by `pm-scheduler --speed` moves as much as a real one.
See also: `sim.state`

16.20.3 `sim.state`

Published by: `market_sim`

Transport: `sim_pub`

Since: 1.0

pm-market-sim's heartbeat: whether the model is stepping and how far it has got.

Field	Type	Presence	Rules	Description
`state`	enum: `RUNNING`, `PAUSED`	required	—	RUNNING during continuous trading, PAUSED otherwise.
`session`	`string`	required	max_len 32	The engine session state the model follows.
`seq`	`int`	required	unit `dimensionless`	The last sim.value published (0 = none yet).
`ts_ns`	`int`	required	unit `epoch_nanos`	
`step_ns`	`int`	required	gt 0, unit `duration_nanos`	Time between steps while running.
`symbols`	`int`	required	unit `dimensionless`	
`seed`	`int`	required	unit `dimensionless`	

Note

Published once a second whatever the session, so a subscriber can tell "the exchange is closed" (PAUSED) from "pm-market-sim is gone" (no heartbeat).
See also: `sim.value`

16.20.4 `sim.command`

Published by: `admin`

Transport: `sim_push`

Since: 1.0

Instructor tooling to pm-market-sim: one command.

Field	Type	Presence	Rules	Description
`command_id`	`string`	required	max_len 64	
`gateway_id`	`string`	required	max_len 32	The sender; the reply goes to sim.command_ack..
`action`	enum: `STATUS`, `NEWS_INJECT`, `NEWS_CONFIRM`, `NEWS_RETRACT`	required	—	

Field	Type	Presence	Rules	Description
`news`	`NewsRequest`	omitted when unset	—	The headline to publish (NEWS_INJECT).
`news_id`	`string`	defaults to `''`	max_len 32	The rumour to confirm or retract.

Note

pm-market-sim binds its PULL socket to loopback and accepts a command only when `gateway_id` is an ADMIN participant of the deployed configuration.
See also: `sim.command_ack.{GW_ID}`

16.20.5 `sim.command_ack.{gateway_id}`

Published by: `market_sim`

Transport: `sim_pub`

Since: 1.0

pm-market-sim to the commanding tool: the outcome of one command.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`command_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 256	Refusal detail, or what was done.
`news_id`	`string`	defaults to `''`	max_len 32	The news.event published, if any.

See also: `sim.command`

16.21 Family `structure`

16.21.1 `combo.ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to gateway: a combo submission was accepted or rejected.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

Field	Type	Presence	Rules	Description
`combo_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	

Note

Three scalars.
It carried a full ComboOrder.to_dict() state dump until the submission, event and persistence shapes were separated in 5.1c - and no consumer had ever read it: alf_console, alf_gwy, pm-stats and the api_gateway event stream all take only these three.
Design section 15.4 records that removal.
See also: `order.combo`, `combo.status.{GW_ID}`

16.21.2 `combo.status.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to gateway: a combo moved to a new lifecycle state.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`combo_id`	`string`	required	max_len 64	
`status`	enum: `PENDING`, `PARTIALLY_MATCHED`, `MATCHED`, `FAILED`, `CANCELLED`, `REJECTED`	required	—	Mirrors models/combo.py::ComboStatus.
`reason`	`string`	omitted when empty	max_len 512	Why a terminal status was reached; absent on the happy path.

Note

PENDING is never published - it is the state a combo is created in, so the first event a client sees is always a transition out of it.
`reason` replaced a `details` map carrying exactly one key, always "reason", which both consumers unwrapped on arrival.
It is omitted when empty, which is what the map's `if reason else None` guard did.
See also: `combo.ack.{GW_ID}`

16.21.3 `oco.ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to gateway: an OCO pair was accepted or rejected.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`oco_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`order_id_1`	`string`	defaults to `''`	max_len 64	The first leg's engine order id.
`order_id_2`	`string`	defaults to `''`	max_len 64	The second leg's engine order id.

Note

The two order ids are always emitted, as " " on rejection: the hand-written builder put them in the base payload rather than under a guard, and a rejected pair has no orders to name.
They are what lets a client tie the pair to the two single-order acks that follow.
See also: `order.oco`, `oco.cancelled.{GW_ID}`

16.21.4 `oco.cancelled.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to gateway: one leg of an OCO pair was cancelled because the other was actioned.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`oco_id`	`string`	required	max_len 64	
`cancelled_order_id`	`string`	required	max_len 64	
`reason`	`string`	defaults to `''`	max_len 512	

Note

Distinct from order.cancelled, which says an order is gone.
This says *why* - the sibling filled or was cancelled - and names the pair, which an order-level event cannot.
See also: `oco.ack.{GW_ID}`, `order.cancelled.{GW_ID}`

16.21.5 `oco.cancel_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to gateway: the outcome of an order.oco_cancel.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`oco_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`reject_code`	`string`	omitted when empty	max_len 32	On a refusal, a member of order.ack's `reject_code` vocabulary (ORDER_NOT_FOUND, ORDER_ALREADY_TERMINAL, GATEWAY_NOT_CONFIGURED, AUTH_REQUIRED); absent when accepted. A string rather than that enum only because the IDL has no shared enum type: the engine is held to the vocabulary by tests/test_engine_structure_canc
`request_tag`	`string`	omitted when empty	max_len 64	Echoed from the cancel request, so the caller can attribute the outcome.
`cancelled_order_ids`	list of `string`	defaults to `[]`	—	The engine order ids the cancel removed; empty on a refusal.

Note

Every cancel gets exactly one of these, accepted or refused, so a caller (REST's `?wait=ack`) can wait for its own outcome.
Before WP-API17 a refusal rode the submit ack and a success had no group-level reply at all, so neither could be attributed to the request that asked.

16.21.6 `combo.cancel_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to gateway: the outcome of an order.combo_cancel.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`combo_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	
`reject_code`	`string`	omitted when empty	max_len 32	On a refusal, a member of order.ack's `reject_code` vocabulary (ORDER_NOT_FOUND, ORDER_ALREADY_TERM GATEWAY_NOT_CONFIGUR AUTH_REQUIRED); absent when accepted. A string rather than that enum only because the IDL has no shared enum type: the engine is held to the vocabulary by tests/test_engine_structur
`request_tag`	`string`	omitted when empty	max_len 64	Echoed from the cancel request, so the caller can attribute the outcome.
`cancelled_order_ids`	list of `string`	defaults to `[]`	—	The engine order ids the cancel removed; empty on a refusal.

Note

Every cancel gets exactly one of these, accepted or refused, so a caller (REST's `?wait=ack`) can wait for its own outcome.

Before WP-API17 a refusal rode the submit ack and a success had no group-level reply at all, so neither could be attributed to the request that asked.

16.22 Family `system`

16.22.1 Record types

`SymbolInfo`

One tradable instrument, as the engine sees it for a given caller. What used to be one entry of `symbols` and one entry of `symbol_meta`, which were always the same instrument written twice. The market-maker fields are resolved per caller, not per symbol: the engine layers the gateway's own policy, the global per-symbol policy and the gateway's per-symbol override before publishing. Two gateways asking at the same moment get different values for the same instrument, which is why they travel on this reply rather than in the reference bundle.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`tick_decimals`	`int`	required	ge 0, le 9, unit `dimensionless`	The tick scale, as the integer exponent the engine actually holds. Same field, same name and same meaning as `book.tick_decimals` and `system.eod`'s. It replaces `tick_size`, which was this value passed through `10 ** -n` on the way out and `round(-log10(x))` on the way back in by two consumers, and read under a third spelling by a third.

Field	Type	Presence	Rules	Description
`enforce_mm_obligation`	`bool`	omitted when unset	—	Whether this caller carries a market-making obligation on this instrument. Absent when the caller is not a configured gateway – regime 3 rather than `false`, because "no obligation configured" and "obligation explicitly disabled" are different answers and the console renders them differently.
`mm_max_spread_ticks`	`int`	omitted when unset	ge 0, unit `ticks`	Widest quote that satisfies the obligation. Absent with it.
`mm_min_qty`	`int`	omitted when unset	ge 0, unit `shares`	Smallest quote that satisfies the obligation. Absent with it.
`prev_close`	`float`	omitted when unset	unit `display_price`	Previous session's closing reference price, from persisted book stats. Absent on an instrument that has never closed, which is a real state on the first day of a deployment.

`Collar`

Price-band configuration. The two bands `CollarConfig` reads; a deployment writing other keys under `collar:` in its YAML had them carried onto the wire and read by nothing, and they no longer travel.

Field	Type	Presence	Rules	Description
`static_band_pct`	`float`	required	ge 0, unit `percent`	
`dynamic_band_pct`	`float`	required	ge 0, unit `percent`	

`OrderLimits`

Pre-trade order-size and notional caps, as configured on a symbol. Each cap is independently optional: an absent cap is not enforced, the same way an absent `collar` leaves a symbol uncollared.

Field	Type	Presence	Rules	Description
`max_order_qty`	`int`	omitted when unset	gt 0, unit `shares`	Largest quantity a single order may carry. Absent when no cap is configured.
`max_order_value`	`float`	omitted when unset	gt 0, unit `money`	Largest notional (`quantity * price`) a single order may carry. Absent when no cap is configured, and never evaluated for an order that carries no price on the wire – MARKET and IOC – which the collar's price bands already skip for the same reason.

`CircuitBreakerLevel`

One rung of a symbol's circuit-breaker ladder, as configured. `name` is the string `circuit_breaker.halt.level` and `admin.action.scope.level` carry onward, and is bounded here to match them.

Field	Type	Presence	Rules	Description
`name`	`string`	required	max_len 32	
`price_shift_pct`	`float`	required	gt 0, unit `percent`	

Field	Type	Presence	Rules	Description
`halt_duration_ns`	`int`	required	ge 0, unit `duration_nanos`	How long a halt at this rung lasts; null for a rung that halts for the rest of the day. A duration, not an instant – the distinction `epoch_nanos` could not make, and the reason the unit registry gained `duration_nanos` in this phase. See design section 28.7.

`SymbolCircuitBreaker`

A symbol's configured circuit-breaker ladder and its lookback.

Field	Type	Presence	Rules	Description
`reference_window_ns`	`int`	required	ge 0, unit `duration_nanos`	Lookback window for the reference price. A duration.
`levels`	list of `CircuitBreakerLevel`	required	—	

`CheckpointOrder`

One order as a book checkpoint holds it: enough to rebuild the book exactly – its place in the queue, hidden iceberg quantity, its stop and trailing triggers, and the links to its quote, OCO pair or combo.

Field	Type	Presence	Rules	Description
`order_id`	`string`	required	max_len 64	
`gateway_id`	`string`	required	max_len 32	
`side`	enum: `BUY`, `SELL`	required	—	
`order_type`	enum: `MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	required	—	

Field	Type	Presence	Rules	Description
`tif`	enum: `DAY`, `GTC`, `ATO`, `ATC`	required	—	
`quantity`	`int`	required	ge 0, unit `shares`	
`remaining_qty`	`int`	required	ge 0, unit `shares`	Hidden iceberg quantity included.
`price_ticks`	`ticks`	omitted when unset	unit `ticks`	
`stop_price_ticks`	`ticks`	omitted when unset	unit `ticks`	The current trigger (a trailing stop's ratcheted one).
`trail_offset_ticks`	`ticks`	omitted when unset	unit `ticks`	
`visible_qty`	`int`	omitted when unset	unit `shares`	An iceberg's slice size.
`displayed_qty`	`int`	omitted when unset	unit `shares`	What an iceberg shows now.
`priority_seq`	`int`	omitted when unset	unit `dimensionless`	The book's arrival sequence; absent for a trailing stop (checked in list order).
`smp_action`	`string`	omitted when unset	—	
`oco_group_id`	`string`	omitted when unset	—	
`quote_id`	`string`	omitted when unset	—	
`combo_parent_id`	`string`	omitted when unset	—	

`LoadedFile`

One persistence file the engine read at startup, identified by its bytes: a replay can tell which state the run started from.

Field	Type	Presence	Rules	Description
`name`	`string`	required	max_len 64	File name in the data dir, e.g. gtc_orders.json.

Field	Type	Presence	Rules	Description
`sha256`	`string`	`null` when unset	max_len 64	SHA-256 of the file's bytes; null when the file did not exist.
`entries`	`int`	required	ge 0, unit `dimensionless`	Orders, combos or symbols the file holds (0 when absent).

`ReferenceSymbol`

One instrument's static configuration. Distinct from `SymbolInfo`, which is the same instrument as a *caller* sees it: this record is identical for every caller and changes only on reload, which is why the bundle can be cached and hashed and the symbols reply cannot.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	Was the map key. Section 19.2.
`tick_decimals`	`int`	required	ge 0, le 9, unit `dimensionless`	As on `SymbolInfo`; `tick_size` is gone from here too.
`level`	`string`	omitted when unset	max_len 32	Which risk-control level this symbol resolves to, naming an entry of `risk.levels`. Absent on a symbol with no level configured.
`collar`	`Collar`	omitted when unset	—	Both bands or neither – section 16.2's combination.
`order_limits`	`OrderLimits`	omitted when unset	—	The symbol's order-size and notional caps. Configured per symbol and nowhere else – there is no level or global default – so an absent cap is one that is not enforced.

Field	Type	Presence	Rules	Description
`circuit_breaker`	`SymbolCircuitBreaker`	omitted when unset	—	

`RiskLevel`

One named risk-control level. Was a map entry keyed by `name`; section 19.2's shape, with the key as a field.

Field	Type	Presence	Rules	Description
`name`	`string`	required	max_len 32	
`collar`	`Collar`	omitted when unset	—	The level's default bands, which a symbol may override. Absent when the level configures none – previously an empty `collar` object, which said the same thing in more bytes.

`ReferenceRisk`

The risk-control ladder as configured, and which rung is the default.

Field	Type	Presence	Rules	Description
`default_level`	`string`	omitted when unset	max_len 32	Names an entry of `levels`. Absent when none is configured.
`levels`	list of `RiskLevel`	required	—	

`SessionTimes`

The trading day's clock, as five wall-clock times. Carried by `system.session_schedule` and, nested inside `ReferenceSchedule`, by `system.reference` – one shape declared once rather than two declarations that can drift apart. The values are strings because that is what the config file holds and what every consumer renders. Individually nullable at the wire level for structural uniformity, though in practice all five are always populated together: `config_loader` rejects a `schedule:` day block that defines some of the five keys but not all.

Field	Type	Presence	Rules	Description
`pre_open`	`string`	`null` when unset	max_len 32	
`opening_auction_start`	`string`	`null` when unset	max_len 32	

Field	Type	Presence	Rules	Description
`continuous_start`	`string`	`null` when unset	max_len 32	
`closing_auction_start`	`string`	`null` when unset	max_len 32	
`closing_auction_end`	`string`	`null` when unset	max_len 32	

`NextTransition`

The next session transition: what the session moves to, and when. The same record as `session`'s NextTransition, declared again because a type cannot be shared across families; the wire shape is identical.

Field	Type	Presence	Rules	Description
`state`	`string`	required	max_len 32	
`at`	`string`	required	max_len 32	UTC ISO-8601 instant of the transition.

`Holiday`

One bank holiday in the exchange's holiday calendar.

Field	Type	Presence	Rules	Description
`date`	`string`	required	max_len 10	ISO-8601 calendar date, in the session timezone.
`name`	`string`	required	max_len 64	The holiday's name as python-holidays gives it.

`WeeklySchedule`

A fully-resolved weekly session schedule: one `SessionTimes` per weekday (mon..sun, any of which may be null for CLOSED) plus a separate holidays entry, mirroring `edumatcher.engine.config_loader.ScheduleConf` The `weekdays`/ `weekend` shortcuts and per-day overrides in the YAML are already resolved by the time this is built – every field here is one concrete day, never a shortcut. `today`/`today_is_holiday` are a server-computed convenience: the engine already runs the same `python-holidays` lookup for pm-scheduler, so it resolves "which entry applies right now" once and publishes the answer, rather than making every client (in particular a browser) carry its own holiday calendar just to answer that one question. `today` is still `SessionTimes \| null` – null means CLOSED today, exactly like any other day slot. `upcoming_holidays` answers the next question, "when is the exchange next on its holiday schedule", the same way. It is computed when the record is built and covers a year, so a client picks the first entry on or after its own today rather than trusting the list to start there.

Field	Type	Presence	Rules	Description
`mon`	`SessionTimes`	`null` when unset	—	
`tue`	`SessionTimes`	`null` when unset	—	
`wed`	`SessionTimes`	`null` when unset	—	
`thu`	`SessionTimes`	`null` when unset	—	
`fri`	`SessionTimes`	`null` when unset	—	
`sat`	`SessionTimes`	`null` when unset	—	
`sun`	`SessionTimes`	`null` when unset	—	
`holidays`	`SessionTimes`	`null` when unset	—	Applied instead of the weekday's own entry on a bank holiday. Null means CLOSED on holidays.
`today`	`SessionTimes`	`null` when unset	—	The entry actually in effect today (holidays entry if today is a bank holiday, else today's weekday entry). Null means CLOSED today.
`today_is_holiday`	`bool`	required	—	True only when `today` came from the holidays entry rather than the weekday's own entry.
`upcoming_holidays`	list of `Holiday`	required	max_items 64	Bank holidays for the configured country from the session date the record was built on, through the following 365 days, ascending. Each one runs on the `holidays` entry.

`ReferenceSchedule`

The venue's calendar configuration: whether sessions run at all, which country's holidays they observe, and the resolved weekly clock. `schedule` is nested rather than flattened beside its two siblings,

which is a change to `GET /reference/schedule`. The alternative was declaring `WeeklySchedule`'s fields a second time inline, and a shape described twice is the drift section 1 is about.

Field	Type	Presence	Rules	Description
`sessions_enabled`	`bool`	required	—	
`country`	`string`	omitted when unset	max_len 2	ISO 3166-1 alpha-2, for the holiday calendar. Absent when unset.
`schedule`	`WeeklySchedule`	`null` when unset	—	Null when the deployment configures no `schedule:` block. Regime 2 rather than 3: both readers pass the bundle through verbatim to a caller, and a key that is sometimes absent is harder to consume than one that is sometimes null.

`IndexDefinition`

One index as configured – its membership and its starting level, not its current one. `GET /history/index-daily` serves the live value; see `index.rebalance`.

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 32	
`description`	`string`	defaults to `''`	max_len 128	
`base_value`	`float`	required	unit `dimensionless`	Index points. `dimensionless` to match `index.yaml`.
`constituents`	list of `string`	required	—	

`EodBookLevel`

One aggregated price level of a closing book. The same three fields as `book.BookLevel`, redeclared because records are family-scoped and the IDL has no cross-family reference – see design section 28.5 on why that stays a known duplication rather than becoming a shared-types construct on the strength of one instance. `price` is required here. `BookLevelPayload`, the hand-written dataclass this replaces, made it optional; the only producer is `OrderBook.snapshot()`, which has never emitted a level without one.

Field	Type	Presence	Rules	Description
`price`	`float`	required	unit `display_price`	
`qty`	`int`	required	unit `shares`	
`count`	`int`	required	unit `dimensionless`	

`EodBook`

One symbol's closing book. A deliberately trimmed `book.book_snapshot: SystemEodPayload.from_dict` has always dropped `last_qty`, `last_buy_price`, `last_sell_price` and `recent_trades` from the snapshot it is handed, and this record says so rather than leaving it to a `from_dict` a reader has to go and find.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`tick_decimals`	`int`	required	ge 0, le 9, unit `dimensionless`	The scale these display prices were produced at.
`bids`	list of `EodBookLevel`	required	—	Descending by price.
`asks`	list of `EodBookLevel`	required	—	Ascending by price.
`last_price`	`float`	omitted when unset	unit `display_price`	Absent on an instrument that never traded.

`HaltedSymbol`

One currently-halted instrument. The circuit-breaker detail is present only when a circuit breaker is what halted it: an ADMIN halt sets the flag without a breaker behind it, so the three fields travel together or not at all – section 16.2's combination, expressed as three regime-3 fields rather than a record because they are three independent CALF-side values and no reader holds them as a unit (section 26.2's reasoning, second application).

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`resume_at_ns`	`int`	omitted when unset	unit `epoch_nanos`	When the halt lapses. Null on an indefinite halt.
`level`	`string`	omitted when unset	max_len 32	The rung that triggered it. Bounded to match the ladder.
`halt_source`	`string`	omitted when unset	max_len 32	

`Position`

One instrument the gateway is not flat in. Only non-zero net positions are reported, so an empty list means flat everywhere.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`net_qty`	`int`	required	unit `shares`	Positive is long, negative is short. Never zero.
`avg_cost`	`float`	required	ge 0, unit `display_price`	Volume-weighted entry price; 0.0 when unknown.

`ActiveQuote`

One active two-sided quote, with both legs' live order state. What a market maker reads on reconnect to find out what it already has resting. A leg whose order is gone reports `MISSING` with zero quantities rather than being omitted: the quote still exists as far as the engine's index is concerned, and a bootstrap that silently dropped one side would let a bot re-quote into its own resting order.

Field	Type	Presence	Rules	Description
`quote_id`	`string`	required	max_len 64	
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	required	max_len 16	
`state`	`string`	required	max_len 32	
`bid_order_id`	`string`	required	max_len 64	
`ask_order_id`	`string`	required	max_len 64	
`bid_price`	`float`	`null` when unset	unit `display_price`	
`ask_price`	`float`	`null` when unset	unit `display_price`	
`bid_qty`	`int`	required	ge 0, unit `shares`	
`ask_qty`	`int`	required	ge 0, unit `shares`	
`bid_remaining_qty`	`int`	required	ge 0, unit `shares`	
`ask_remaining_qty`	`int`	required	ge 0, unit `shares`	
`bid_status`	`string`	required	max_len 32	
`ask_status`	`string`	required	max_len 32	

`QuoteLeg`

One live leg of an active quote, with its order's current state.

Field	Type	Presence	Rules	Description
`quote_id`	`string`	required	max_len 64	

Field	Type	Presence	Rules	Description
`order_id`	`string`	required	max_len 64	
`symbol`	`string`	required	max_len 16	
`leg_side`	enum: `BUY`, `SELL`	required	—	Which side of the quote this leg is. Exactly two producers.
`price`	`float`	omitted when unset	unit `display_price`	Leg limit price in display money. A quote leg is always a priced limit order; null only if the resting order's price is somehow unavailable.
`qty`	`int`	required	ge 0, unit `shares`	
`remaining`	`int`	required	ge 0, unit `shares`	
`filled`	`int`	required	ge 0, unit `shares`	
`status`	`string`	required	max_len 32	
`quote_status`	`string`	required	max_len 32	

`QuoteLegSnapshot`

A leg as it stood when its quote left the book. No live qty/remaining here in the sense the name suggests – these are the final values, recorded at removal, because once an order leaves the book its state is not available anywhere in the engine.

Field	Type	Presence	Rules	Description
`order_id`	`string`	required	max_len 64	
`price`	`float`	omitted when unset	unit `display_price`	Leg limit price in display money, as it stood at removal. Mirrors `QuoteLeg.price` so an ACTIVE row and a RECENT row for the same leg read the same; null when the order's price was unavailable at snapshot time.
`qty`	`int`	required	ge 0, unit `shares`	

Field	Type	Presence	Rules	Description
`remaining`	`int`	required	ge 0, unit `shares`	
`filled`	`int`	required	ge 0, unit `shares`	
`status`	`string`	required	max_len 32	

`RecentQuote`

One recently-removed quote, from the engine's bounded per-gateway inactivation history. A quote-level summary rather than a per-leg one. Does not survive an engine restart.

Field	Type	Presence	Rules	Description
`quote_id`	`string`	required	max_len 64	
`symbol`	`string`	required	max_len 16	
`bid_order_id`	`string`	required	max_len 64	
`ask_order_id`	`string`	required	max_len 64	
`quote_status`	`string`	required	max_len 32	
`reason`	`string`	required	max_len 64	Why it was removed. `quote_status` is derived from this.
`removed_at_ns`	`int`	required	unit `epoch_nanos`	
`bid_leg`	`QuoteLegSnapshot`	`null` when unset	—	Null when the quote had no bid leg to record.
`ask_leg`	`QuoteLegSnapshot`	`null` when unset	—	

`LiveCircuitBreaker`

A symbol's circuit breaker as it stands right now. The live counterpart to `ReferenceSymbol.circuit_breaker`, which is the configuration.

Field	Type	Presence	Rules	Description
`halted`	`bool`	required	—	
`reference_price`	`float`	`null` when unset	unit `display_price`	
`trigger_price`	`float`	`null` when unset	unit `display_price`	
`triggered_level`	`string`	`null` when unset	max_len 32	
`expansion_index`	`int`	`null` when unset	ge 0, unit `dimensionless`	

Field	Type	Presence	Rules	Description
`corridor_low`	`float`	`null` when unset	unit `display_price`	Flat, and named without the `corridor.` box it used to sit in. `_corridor_payload` returns `{corridor_low, corridor_high, expansion}` and `risk_state` nested that under a key called `corridor`, so the wire read `corridor.corridor_low` – the prefix and the box saying the same thing twice. The same helper is splatted *flat* into `circuit_breaker.halt`, so one producer was emitting two shapes of one value. Flat here matches the event. See design 29.2.
`corridor_high`	`float`	`null` when unset	unit `display_price`	
`corridor_expansion`	`int`	`null` when unset	ge 0, unit `dimensionless`	The ACE expansion count that goes with the bounds above. Distinct from `expansion_index`, which is always a real integer; this one is null exactly when the corridor is.
`resume_at_ns`	`int`	`null` when unset	unit `epoch_nanos`	

SymbolRiskState

One symbol's live risk state. Was a map entry keyed by symbol – section 19.2's shape for the twelfth time.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`collar_reference_price`	`float`	omitted when unset	unit `display_price`	Absent on a symbol with no collar configured.
`circuit_breaker`	`LiveCircuitBreaker`	omitted when unset	—	Absent on a symbol with no circuit breaker configured.

GatewayInfo

One configured participant, and whether it is connected right now.

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 32	
`role`	`string`	required	max_len 32	
`description`	`string`	defaults to `''`	max_len 128	
`connected`	`bool`	required	—	

SymbolVolume

One instrument's traded volume so far today. Was a map entry keyed by symbol; the key is a field now, as everywhere else in this family.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`qty`	`int`	required	ge 0, unit `shares`	
`value`	`float`	required	ge 0, unit `money`	Traded notional, rounded to two decimals by the producer.
`trades`	`int`	required	ge 0, unit `dimensionless`	

16.22.2 `system.gateway_connect`

Published by: `admin`, `api_gateway`, `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Gateway to engine: authenticate this participant. Sent over PUSH/PULL rather than the pub bus; it carries a topic so the audit log can classify it alongside everything else.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Who is connecting. The engine's PULL socket is a boundary of its own – section 22.3 – so this is clamped on arrival.

See also: `system.gateway_auth.{GW_ID}`, `system.gateway_disconnect`

16.22.3 `system.gateway_auth.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all subscribers: a participant's connection was accepted or rejected. The PUB-side answer to `system.gateway_connect`, and the widest-read message in this half – five consumers structurally, and it is in `PRIVATE_PREFIXES`.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`accepted`	`bool`	required	—	
`reason`	`string`	defaults to `''`	max_len 512	Why it was rejected; " " on acceptance. Regime 1, because `GatewayAuthPayload.to_dict` has always emitted the key.
`description`	`string`	defaults to `''`	max_len 128	The gateway's configured display name. Regime 1, as above.

Note

`gateway_id` is in the topic AND the body, and every consumer reads it from the body – `balf_gwy` dispatches on the topic suffix and then reads the payload's copy.
So the field is listed in `include:` explicitly.
Section 26.4: `include: all` means "every field except the topic parameters", and taking the default

here would have dropped the key from the wire with `pm-msgen check` still passing.
See also: `system.gateway_connect`, `system.gateway_bye.{GW_ID}`

16.22.4 `system.gateway_reject`

Published by: `api_gateway`, `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Gateway to engine: I refused a participant's request myself, before the engine saw it. Authentication, role and rate-limit refusals are decided in the ALF, BALF and REST gateways and never reach the engine, so without this the exchange could neither show nor disprove "I sent it and you ignored it". PUSH/PULL like `gateway_connect`; the engine echoes it onto the bus like any command, which is how the audit trail comes to hold it. Malformed frames and field-validation refusals are deliberately not reported: they are counted by the gateway, and any one of them is unfalsifiable noise rather than a disputable decision.

Field	Type	Presence	Rules	Description
`source`	enum: `ALF`, `BALF`, `REST`	required	—	Which gateway refused the request.
`gateway_id`	`string`	defaults to `''`	max_len 32	The participant the gateway believed it was talking to; " " before authentication. Claimed, not proven. Never an API key.
`peer`	`string`	defaults to `''`	max_len 64	Remote address of the connection (host:port).
`reject_code`	enum: `AUTH_REQUIRED`, `AUTH_FAILED`, `ROLE_DENIED`, `RATE_LIMITED`	required	—	The same codes `order.ack` uses for these refusals.
`reason`	`string`	defaults to `''`	max_len 256	
`count`	`int`	defaults to `1`	ge 1, unit `dimensionless`	Refusals this record stands for: itself plus those folded in since the last record for the same peer and code.

Field	Type	Presence	Rules	Description
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	The gateway's clock when it refused.

Note

A gateway reports at most one record per peer host and reject code per second and carries the number it folded in as `count`, so a client flooding refused commands cannot flood the engine through this message, even with a new connection (source port) per request.
See also: `system.gateway_connect`, `order.ack.{GW_ID}`

16.22.5 `system.gateway_disconnect`

Published by: `admin`, `api_gateway`, `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Gateway to engine: I am leaving cleanly. PUSH/PULL, like `gateway_connect`; the engine republishes it as `gateway_bye` so PUB subscribers hear about it at all.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`reason`	`string`	defaults to `''`	max_len 512	

See also: `system.gateway_bye.{GW_ID}`, `system.gateway_connect`

16.22.6 `system.gateway_heartbeat`

Published by: `api_gateway`, `gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

Gateway to engine: this participant's process is still alive. A process that dies without sending `gateway_disconnect` (kill -9, a crashed host, a dropped network) would otherwise keep its ID marked connected for ever, and the engine refuses a second connect for an ID that is already connected. The engine counts the beats it expects – one per `interval_sec` – and disconnects the session, applying its `disconnect_behaviour`, after three consecutive misses. Liveness is opt-in: a session that has never sent one is never timed out.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

Field	Type	Presence	Rules	Description
`interval_sec`	`int`	required	ge 1, le 3600, unit `dimensionless`	Seconds between this sender's beats. Carried on every beat so the engine needs no per-client setting; the bound stops a client declaring an interval so long that its ID could never be reclaimed.

See also: `system.gateway_connect`, `system.gateway_disconnect`

16.22.7 `system.gateway_bye.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all subscribers: a participant has disconnected. The PUB-side counterpart to `gateway_auth`. The inbound `system.gateway_disconnect` is a PULL message and never reaches subscribers, so without this broadcast clearing could not close the matching session.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`reason`	`string`	defaults to `''`	max_len 512	

Note

`gateway_id` is in the topic and the body, and `clearing` – the only structural reader – takes it from the body.
Enumerated for the same reason as `gateway_auth`.
See also: `system.gateway_disconnect`, `system.gateway_auth.{GW_ID}`

16.22.8 `system.eod`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to all subscribers, once, before shutdown: the closing book of every instrument. Four consumers – clearing, index, stats and the RALF gateway – which is the widest fan-out in this half, and the reason the record is declared rather than left to a `from_dict`.

Field	Type	Presence	Rules	Description
`books`	list of `EodBook`	required	—	One per instrument the engine had a book for, unordered.

Note

A broadcast with no request: nothing asks for end of day, the engine announces it.
That is why it has no `_request` sibling and why it is in this half rather than 6.1f's request/reply pairs.
See also: `book.{SYMBOL}`, `session.state`

16.22.9 `system.book_checkpoint.{symbol}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.3

One symbol's whole book, for the audit: every resting order in priority order with its hidden quantity, every armed stop with its current trigger, the last prices. Published for every symbol after startup (what the restore and the seeds made), after each session transition, and at shutdown. A replay can start from any of them – an audit that begins mid-day, or continues after an engine restart – and, where it already holds the book, compare order by order.

Field	Type	Presence	Rules	Description
`symbol`	`string`	required	max_len 16	
`reason`	enum: `STARTUP`, `PHASE`, `EOD`	required	—	
`phase`	`string`	required	max_len 32	The session state the book is in.
`bids`	list of `CheckpointOrder`	required	—	
`asks`	list of `CheckpointOrder`	required	—	
`buy_stops`	list of `CheckpointOrder`	required	—	
`sell_stops`	list of `CheckpointOrder`	required	—	
`trailing_stops`	list of `CheckpointOrder`	required	—	
`last_trade_price_ticks`	`ticks`	omitted when unset	unit `ticks`	
`last_buy_price_ticks`	`ticks`	omitted when unset	unit `ticks`	

Field	Type	Presence	Rules	Description
`last_sell_price_ticks`	`ticks`	omitted when unset	unit `ticks`	
`last_trade_id`	`string`	omitted when unset	—	The trade that set the last trade price.
`tick_decimals`	`int`	omitted when unset	ge 0, le 9, unit `dimensionless`	Decimal scale of the `*_ticks` prices in this message (1 tick = 10^-tick_decimals) – the symbol's. Every message that carries a tick price carries its scale, so a reader of one line, or of a window of the trail that begins long after the book snapshot that declared it, can turn it into money without guessing. Absent when the message carries no tick price.

Note

`bids` and `asks` are in matching priority (best price first, then arrival); `buy_stops` and `sell_stops` in the order they would trigger; `trailing_stops` in the order they are checked.
See also: `system.startup`, `engine.trace.{SYMBOL}`, `book.{SYMBOL}`

16.22.10 `system.startup`

Published by: `engine`

Transport: `engine_pub`

Since: 1.3

Engine to all subscribers, the first message of every start, before the restore: which engine, which configuration and which persisted state this run began from. Makes an audit trail self-describing – a replay rebuilds the engine from this message alone, years later, without the data dir's `ref_data/` (which the next deploy overwrites) or a guess at the engine version. Also the marker of an engine start: every per-topic `seq` begins again after it.

Field	Type	Presence	Rules	Description
`format_version`	`int`	required	ge 1, unit `dimensionless`	Version of this message's meaning; 1 at EduMatcher 1.0.
`engine_version`	`string`	required	max_len 64	The installed edumatcher package version.
`git_commit`	`string`	required	max_len 64	`git describe --always --dirty` of the source the engine runs from; "unknown" for an installed wheel. The package version stays put between releases, the commit does not.
`python_version`	`string`	required	max_len 32	
`run_seq`	`int`	required	ge 0, unit `dimensionless`	The engine-run sequence this start took (trade id prefix).
`config_sha256`	`string`	`null` when unset	max_len 64	SHA-256 of the compiled config artifact; null when none is deployed.
`config`	`string`	`null` when unset	—	The compiled config artifact, verbatim JSON text; null when the engine runs on a YAML config or none.
`loaded`	list of `LoadedFile`	required	max_items 8	The persistence files the restore reads: orders, combos, book statistics.

Note

`config` is the compiled artifact as `pm-config-deploy` wrote it, verbatim (about 250 KB for 300 symbols), so `config_sha256` is the artifact file's hash and `config_artifact.decode(config)` gives back exactly what the engine loaded.

See also: `system.startup_recovery`, `system.recovery_item`

16.22.11 `system.startup_recovery`

Published by: `engine`

Transport: `engine_pub`

Since: 1.1

Engine to all subscribers, once, right after `_restore_gtc()` and before the config-seed step runs: how much of the previous session's resting state came back. Every one of these counts previously only reached `log.info/log.error` — invisible to `pm-audit`, which is a bare PUB subscriber and sees nothing that is not published (see ../../docs/books/operator-guide/part-6-observe-and-recover/020-audit-trail.md). A post-mortem investigating "why does the book look different after a restart" needs this the same way it needs `order.cancelled` for a live cancel.

Field	Type	Presence	Rules	Description
`restored_orders`	`int`	required	ge 0, unit `dimensionless`	GTC/same-day-DAY orders successfully restored to their book.
`discarded_stale_day_orders`	`int`	required	ge 0, unit `dimensionless`	TIF=DAY orders discarded because the business day rolled over.
`failed_orders`	`int`	required	ge 0, unit `dimensionless`	Persisted orders that raised during restore and were skipped.
`quote_remnants_restored`	`int`	required	ge 0, unit `dimensionless`	Restored quote-origin legs whose sibling did not survive, so they rest as plain orders rather than being rejoined into QuoteIndex.
`rebuilt_quotes`	`int`	required	ge 0, unit `dimensionless`	Active QuoteIndex entries rebuilt from both-legs-restored quotes.
`restored_combos`	`int`	required	ge 0, unit `dimensionless`	GTC combos restored from the previous session.

Note

A broadcast with no request, like `system.eod` — nothing asks for a recovery summary, the engine announces it once at startup.
See also: `book.{SYMBOL}`, `auction.result`

16.22.12 `system.recovery_item`

Published by: `engine`

Transport: `engine_pub`

Since: 1.2

Engine to all subscribers: one restored, discarded or failed entity from `_restore_gtc()` (AR-0.5). `startup_recovery`'s six counts say *how many*; this says *which ones* — "which order failed to restore?" was unanswerable from either the counts alone or the process log, since a post-mortem investigating a startup only sees what was published (see ../../docs/books/operator-guide/part-6-observe-and-recover/020-audit-trail.md). Published once per entity, before the single summary `startup_recovery` broadcast, so the counts there are the cross-check: `restored_orders + discarded_stale_day_orders + failed_orders` must equal the number of ORDER-kind recovery_item lines whose outcome is RESTORED, DISCARDED_STALE_DAY or FAILED respectively, and `restored_combos` must equal the number of COMBO-kind lines.

Field	Type	Presence	Rules	Description
`entity_id`	`string`	required	max_len 64	The order id (ORDER) or combo id (COMBO) this line is about.
`kind`	enum: `ORDER`, `COMBO`	required	—	What restore_gtc() was processing.

Field	Type	Presence	Rules	Description
`outcome`	enum: `RESTORED`, `DISCARDED_STALE_DAY`, `FAILED`, `QUOTE_REMNANT`	required	—	RESTORED/DISCARDED come from the main GTC-order (or GTC-combo) restore pass and are mutually exclusive per entity. QUOTE_REMNANT is a second, additional line for a RESTORED quote-origin order whose sibling leg did not come back — see the example_note above. COMBO-kind lines are always RESTORED today: `load_gtc_combos()` has no per-combo guard, so there is no combo failure path to report yet.
`symbol`	`string`	omitted when unset	max_len 16	The instrument, for an ORDER-kind line. Null for COMBO, which may span more than one symbol across its legs and has none of its own.
`detail`	`string`	omitted when empty	max_len 512	Free text, when there is something worth saying beyond kind and outcome — the exception message for a FAILED order, in particular.

Note

`QUOTE_REMNANT` is deliberately not exclusive with `RESTORED`: a quote-origin order whose sibling leg did not survive is *both* one of `restored_orders` (it rests on the book, ordinarily) *and* one of `quote_remnants_restored` (it is not quote-managed going forward) — `_restore_gtc()` has always counted it in both totals, so it gets two recovery_item lines with the same entity_id, not one that overwrites the other.
`rebuilt_quotes` — an active QuoteIndex entry rebuilt once *both* legs of a quote survive — is deliberately not an entity here.
It is a pairing of two already-reported order ids, not a persisted thing with an id of its own to fail or be discarded; the summary count on `startup_recovery` is the whole story for it, same as it always was.
See also: `system.startup_recovery`

16.22.13 `system.quote_seed_skipped`

Published by: `engine`

Transport: `engine_pub`

Since: 1.3

Engine to all subscribers: a configured seed quote the engine did not inject at startup because `seed_once` is set and the gateway already has an active quote in that symbol, restored from the persisted orders. A seed that fires leaves its own legs' acks; one that does not left no trace but a line in the process log, so a post-mortem asking "why is MM01's quote in this book not the configured seed?" could only infer it from the seed's absence.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	The market maker the seed is configured for.
`symbol`	`string`	required	max_len 16	
`quote_id`	`string`	required	max_len 64	The seed's quote id: the one that was not injected.
`reason`	enum: `ACTIVE_QUOTE_RESTORED`	required	—	`seed_once` is set and an active quote was restored.

See also: `system.recovery_item`, `system.quote_bootstrap.{GW_ID}`

16.22.14 `system.diagnostic`

Published by: `engine`

Transport: `engine_pub`

Since: 1.1

Engine to all subscribers: an internal failure or anomaly that the engine absorbed and kept running past, so trading was not interrupted but the event still happened and a post-mortem needs to see

it. Before this existed, a maintenance-flush exception, a handler crash, an undecodable inbound message, or a message on an unhandled topic reached only the process log — `pm-audit`'s bare PUB subscription on the engine's :5556 socket (see ../../docs/books/operator-guide/part-6-observe-and-recover/020-audit-trail.md) never saw any of them, so a bug that only ever showed up as "the book snapshot stopped updating" or "an order silently never got a reply" had no trail to replay. This does not replace logging (the process log keeps the full traceback); it is the wire-visible marker that something in `component` happened, with enough detail to go find the log line.

Field	Type	Presence	Rules	Description
`component`	enum: `MAINTENANCE_FLUSH`, `DISPATCH_ERROR`, `UNDECODABLE_MESSAGE`, `UNROUTED_TOPIC`	required	—	Which internal area this diagnostic came from.
`detail`	`string`	required	max_len 256	What within component — the flush function name, the message topic, or empty when there is no finer-grained name (an undecodable message has no topic to report).
`error`	`string`	defaults to `''`	max_len 512	The exception text, when this diagnostic came from one.
`count`	`int`	required	ge 0, unit `dimensionless`	This component's running occurrence count for the process lifetime — the same counter the log line already reports, carried onto the wire so a subscriber can tell a first occurrence from a recurring one without correlating timestamps against the log.

Note

Deliberately generic — a `component/detail/error` triple rather than one message type per failure mode, because the failure modes (a maintenance flush, a dispatch-handler crash, an undecodable pull message, an unrouted topic) share nothing structural beyond "engine kept running, something still needs recording.".
See also: `order.ack.{GW_ID}`

16.22.15 `system.symbols_request`

Published by: `admin`, `api_gateway`, `gateway`, `stats`

Transport: `engine_push`

Since: 1.0

Caller to engine: which instruments are tradable, and on what terms for me.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Both the correlation key for the reply topic and the identity the market-maker fields are resolved against.

See also: `system.symbols.{GW_ID}`

16.22.16 `system.symbols.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to caller: the tradable instruments, with the tick scale and this caller's market-making terms for each.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped from the body by the default projection.
`symbols`	list of `SymbolInfo`	required	—	Sorted by symbol, as the engine iterates its books.

Note

The default projection is right here: `gateway_id` names the caller in the topic and has never been in the body.
Verified against the producer rather than assumed – section 26.4 is the case where assuming cost five wires a field.
One collection, not two.
The old payload carried `symbols` as a list of strings beside `symbol_meta` as a map from those same strings to their metadata, built in the same loop; nine readers joined them back together.
The join is gone.
See also: `system.symbols_request`, `system.reference.{GW_ID}`

16.22.17 `system.reference_request`

Published by: `api_gateway`

Transport: `engine_push`

Since: 1.0

Any caller to engine: the compiled reference-data bundle. Static configuration only – nothing that changes during a session.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Correlation key only. The API gateway passes an API key here for read-only callers, since the bundle does not vary by caller and this only has to be unique.

See also: `system.reference.{GW_ID}`, `system.reference_reload`

16.22.18 `system.reference.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to caller: every piece of static venue configuration in one round trip – tick scales, risk levels, circuit-breaker ladders, index definitions and the calendar – with a hash that changes when any of it does.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 64	Topic-only; dropped from the body by the default projection. Bounded at 64 rather than 32 because the API gateway passes an API key here for read-only callers.
`symbols`	list of `ReferenceSymbol`	required	—	Sorted by symbol. Was a map keyed by it.
`risk`	`ReferenceRisk`	required	—	
`indexes`	list of `IndexDefinition`	required	—	Already a list of records before this phase.
`schedule`	`ReferenceSchedule`	required	—	
`config_version`	`string`	`null` when unset	max_len 64	Truncated SHA-256 of the bundle. Null before a config is loaded, which is the only remaining difference between the two states the old two-shape reply used to distinguish. Regime 2: every consumer compares it, so the key has to be there to compare.

Note

The whole payload was a `dict[str, Any]` passed to `encode` unread.
One producer builds it – `_rebuild_reference_cache` – with five fixed top-level keys, so it was a record nobody had written down.
ONE SHAPE, ALWAYS.
Before an engine config is loaded the reply used to be `{"config_version": null}` and nothing else: a second payload shape for the same topic, which every slicing endpoint compensated for with a `.get(key, {})` default.

The bundle is now always complete, with empty collections and a null version, and the compensating defaults go with it.
REST-VISIBLE.
`api_gateway/routers/reference.py` returns slices of this bundle verbatim, so `reference.symbols` becoming a list of records and `reference.schedule` gaining a level of nesting change `GET /reference/symbols` and `GET /reference/schedule`.
Sanctioned, and the better JSON in both cases – a list of objects each carrying its own `symbol` is what a client can iterate without knowing the keys.
`../../docs/books/operator-guide/part-5-gateways/050-api-gateway.md` moves with it.
See also: `system.reference_request`, `system.reference_reload_ack.{GW_ID}`, `system.symbols.{GW_ID}`

16.22.19 `system.reference_reload`

Published by: `api_gateway`

Transport: `engine_pub`, `engine_push`

Since: 1.0

ADMIN to engine: re-read static reference data from disk. Deliberately narrower than a startup load – it never re-seeds quotes, creates or removes books, or touches session and halt state, so a reload that changed the symbol set is rejected rather than partially applied. The engine also rejects reloads when the active config cannot be reconstructed from the audit trail: compiled artifacts have no reloadable source file, and YAML fallback config is not recorded in system.startup. Deploy a compiled artifact and restart to apply config changes without making the audit trail ambiguous.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`command_id`	`string`	required	max_len 64	Correlated by the ack. Read off the wire unclamped before this phase, and quoted straight into a bounded ack field – section 22.3's silent non-answer, four reply paths.

See also: `system.reference_reload_ack.{GW_ID}`, `system.reference.{GW_ID}`

16.22.20 `system.reference_reload_ack.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: the reload verdict, and the new configuration hash when it took.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped from the body by the default projection.
`command_id`	`string`	required	max_len 64	
`accepted`	`bool`	required	—	
`config_version`	`string`	omitted when unset	max_len 64	Present on acceptance.
`reason`	`string`	omitted when unset	max_len 512	Present on rejection.

Note

The default projection is right: `gateway_id` is topic-only and has never been in the body. `config_version` and `reason` are the two halves of the verdict and never travel together – an accepted reload carries the version, a rejected one carries the reason.
See also: `system.reference_reload`

16.22.21 `system.session_state_request`

Published by: `admin`, `api_gateway`, `gateway`, `scheduler`

Transport: `engine_push`

Since: 1.0

Caller to engine: what session are we in, without advancing it. The scheduler is the dominant producer – it asks rather than assuming, because it is not the only thing that can move the session.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

See also: `system.session_status.{GW_ID}`, `session.state`

16.22.22 `system.session_status.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to caller: the current session state, on request. The polled answer to the `session.state` broadcast, for a caller that has just started and missed the last transition.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped from the body by the default projection.
`state`	`string`	required	max_len 32	Same values as `session.state.state`.
`sessions_enabled`	`bool`	required	—	False when the deployment runs continuously with no session schedule at all, which makes `state` advisory.
`next`	`NextTransition`	omitted when unset	—	The countdown target the engine holds, as `session.state.next` last carried it or `session.next` announced it. Lets a caller that missed the last transition still count down to the next one.

Note

Default projection: `gateway_id` is topic-only.
`state` is a plain bounded string rather than an enum, matching `session.state.state` – the value set lives in the session machine, and enumerating it in two specs would be two places to update.
See also: `session.state`, `system.session_state_request`

16.22.23 `system.session_schedule_request`

Published by: `admin`, `api_gateway`

Transport: `engine_push`

Since: 1.0

Operator to engine: the configured session schedule.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

See also: `system.session_schedule.{GW_ID}`, `system.reference.{GW_ID}`

16.22.24 `system.session_schedule.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to operator: the trading day's clock as configured. The same `SessionTimes` record `system.reference` carries, which is what forced these two topics into one phase.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped from the body by the default projection.
`sessions_enabled`	`bool`	required	—	
`schedule`	`WeeklySchedule`	`null` when unset	—	Null when no `schedule:` block is configured.

Note

Default projection: `gateway_id` is topic-only.
Both readers – `commands/client.py` and `GET /admin/schedule` – pass the payload through to a caller without touching a key, so nothing structural constrains the presence regime here; `schedule` is regime 2 to match the copy inside `ReferenceSchedule` rather than because a reader needs it.
It was `schedule or {}` before, so an unconfigured venue sent an empty object where it now sends null.
See also: `system.session_schedule_request`, `system.reference.{GW_ID}`

16.22.25 `system.halt_status_request`

Published by: `api_gateway`

Transport: `engine_push`

Since: 1.0

Any process to engine: which instruments are halted now.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

See also: `system.halt_status.{GW_ID}`, `circuit_breaker.halt.{SYMBOL}`

16.22.26 `system.halt_status.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to caller: every currently-halted instrument, with the breaker state behind it where a breaker is what halted it. The polled answer to the `circuit_breaker.halt` broadcast, for a caller that has just started and missed it.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped by the default projection.
`halted`	list of `HaltedSymbol`	required	—	

Note

An empty list means nothing is halted, which is the normal reply.
The three optional fields are present together or not at all: an ADMIN halt sets the flag with no breaker behind it.
See also: `system.halt_status_request`, `system.risk_state.{GW_ID}`

16.22.27 `system.position_request`

Published by: `gateway`

Transport: `engine_push`

Since: 1.0

Gateway to engine: what am I holding.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Both the correlation key and the account being asked about. A gateway can only ask about itself: the handler answers from `_gateway_positions[gatewa` and nothing else.

See also: `system.position_snapshot.{GW_ID}`

16.22.28 `system.position_snapshot.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to gateway: per-symbol net position and average cost, for the asking gateway only.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped by the default projection.
`positions`	list of `Position`	required	—	

Note

An unauthenticated or unknown gateway gets an empty list rather than a rejection – flat and not-a-gateway are the same answer here, which is deliberate: the alternative tells an unauthenticated caller whether an id exists.
This pair has no consumer in `src/` at all.
It is exercised only by `tests/test_position_snapshot.py`, which is section 27.4's shape – a capability exercised only by its own tests.
It is specified rather than removed because unlike `drop_copy.replay_request` it is fully implemented on both sides and reachable by any gateway; what it lacks is a caller, not an implementation.
Recorded here so the next phase to touch it knows the difference.
See also: `system.position_request`, `trade.executed`

16.22.29 `system.quote_bootstrap_request`

Published by: `admin`, `api_gateway`, `gateway`

Transport: `engine_push`

Since: 1.0

Market maker to engine, on reconnect: what quotes do I already have resting. Without it a bot cannot tell a fresh start from a reconnect and will quote into its own orders.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	defaults to `''`	max_len 16	Narrows the reply to one instrument. `“ ”` means all.

See also: `system.quote_bootstrap.{GW_ID}`, `system.quote_legs_request`

16.22.30 `system.quote_bootstrap.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to market maker: the active quotes it already holds.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped by the default projection.
`quotes`	list of `ActiveQuote`	required	—	

See also: `system.quote_bootstrap_request`, `quote.status.{GW_ID}`

16.22.31 `system.quote_legs_request`

Published by: `api_gateway`, `gateway`

Transport: `engine_push`

Since: 1.0

Operator or market maker to engine: the per-leg detail behind this gateway's quotes, live and recently removed. What `QLEGS` asks for.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	
`symbol`	`string`	defaults to `''`	max_len 16	Narrows the reply to one instrument. `""` means all.
`show`	enum: `ACTIVE`, `RECENT`, `ALL`	defaults to `'ALL'`	—	Which half to return. `ACTIVE` is live legs, `RECENT` is the removal history, `ALL` is both. Enumerated rather than a free string because the handler branches on exactly these three and treats anything else as `ACTIVE` – a typo currently answers, quietly, with the wrong half.

See also: `system.quote_legs.{GW_ID}`, `system.quote_bootstrap_request`

16.22.32 `system.quote_legs.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to caller: live quote legs, recently-removed quotes, or both.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped by the default projection.
`legs`	list of `QuoteLeg`	required	—	Live legs. Empty unless `show` was `ACTIVE` or `ALL`.
`show_requested`	enum: `ACTIVE`, `RECENT`, `ALL`	required	—	Echo of the request's `show`, so replies can be told apart.
`complete`	`bool`	required	—	False when the reply could not fully answer what was asked. Always true today; kept because the recent-history buffer is bounded and a truncated answer needs a way to say so.
`recent`	list of `RecentQuote`	required	—	Removal history. Empty unless `show` was `RECENT` or `ALL`.

Note

`legs` and `recent` are both always present, empty when the requested half does not include them. Regime 4 would be the IDL's default instinct – absent and `[]` are the same value to `alf_gwy`, the only structural reader – but `GET /quotes/legs` returns this payload verbatim, and a REST client should not have to guess whether a key exists.

`[]` is a true statement here rather than an invented one: both halves always mean something on this message.

`show_requested` echoes what was asked, so a caller that pipelined two requests can tell the replies apart.

See also: `system.quote_legs_request`

16.22.33 `system.risk_state_request`

Published by: `api_gateway`

Transport: `engine_push`

Since: 1.0

ADMIN to engine: the live collar and circuit-breaker state of every symbol that has either configured, halted or not.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

See also: `system.risk_state.{GW_ID}`, `system.reference.{GW_ID}`

16.22.34 `system.risk_state.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to ADMIN: live risk state per symbol. The counterpart to `reference.risk`, which is the static definitions, and to `halt_status`, which is only the symbols currently halted.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped by the default projection.
`symbols`	list of `SymbolRiskState`	required	—	Sorted by symbol. Was a map keyed by it.

Note

The fourth `symbols` field in this family and the fourth different thing: a list of `SymbolInfo` on `system.symbols`, of `ReferenceSymbol` on `reference`, of `SymbolRiskState` here and of `SymbolVolume` on `volume`.
Same name, four types, no relationship – which is why a find-and-replace across them would be the worst available mistake.
See also: `system.risk_state_request`, `system.halt_status.{GW_ID}`

16.22.35 `system.gateways_request`

Published by: `admin`, `api_gateway`

Transport: `engine_push`

Since: 1.0

Operator to engine: which participants exist, and who is on.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

See also: `system.gateways.{GW_ID}`, `system.gateway_auth.{GW_ID}`

16.22.36 `system.gateways.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to operator: every configured participant with its role and current connection status. The polled counterpart to the `gateway_auth` / `gateway_bye` broadcasts.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped by the default projection.
`gateways`	list of `GatewayInfo`	required	—	Sorted by id. Empty when no config is loaded.

See also: `system.gateways_request`, `system.gateway_bye.{GW_ID}`

16.22.37 `system.volume_request`

Published by: `admin`

Transport: `engine_push`

Since: 1.0

Operator to engine: how much has traded today.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	

See also: `system.volume.{GW_ID}`, `trade.executed`

16.22.38 `system.volume.{gateway_id}`

Published by: `engine`

Transport: `engine_pub`

Since: 1.0

Engine to operator: traded quantity, notional and trade count, per instrument and exchange-wide.

Field	Type	Presence	Rules	Description
`gateway_id`	`string`	required	max_len 32	Topic-only; dropped by the default projection.
`symbols`	list of `SymbolVolume`	required	—	Sorted by symbol. Was a map keyed by it.
`total_qty`	`int`	required	ge 0, unit `shares`	
`total_value`	`float`	required	ge 0, unit `money`	
`total_trades`	`int`	required	ge 0, unit `dimensionless`	

Note

The totals are carried rather than left to the caller to sum.
That is redundant on the wire and load-bearing off it: they are the engine's own running counters, not a sum of the rows, so a caller adding up `symbols` would silently disagree with the engine about any instrument whose book was removed mid-session.
See also: `system.volume_request`

16.23 Family `trade`

16.23.1 `trade.executed`

Published by: `engine`

Transport: `engine_pub`, `calf`

Since: 1.0

Public print of a completed match. The authoritative record of what traded, consumed by statistics, clearing, index and market data.

Field	Type	Presence	Rules	Description
`id`	`string`	required	max_len 64, pattern `^\d{6}-\d{9}$`	Durable, sortable trade id. The prefix is the persisted engine-run sequence and the suffix is the per-run trade counter.

Field	Type	Presence	Rules	Description
`run_seq`	`int`	required	ge 0, unit `dimensionless`	Durable engine-run sequence used as the trade id prefix. A change in run_seq marks an engine restart explicitly for consumers.
`symbol`	`string`	required	max_len 16, pattern `^[A-Z0-9._]+$`	Instrument the match occurred in.
`buy_order_id`	`string`	required	max_len 64	Resting or aggressing order id on the buy side.
`sell_order_id`	`string`	required	max_len 64	Resting or aggressing order id on the sell side.
`buy_gateway_id`	`string`	required	max_len 32	Gateway that submitted the buy order.
`sell_gateway_id`	`string`	required	max_len 32	Gateway that submitted the sell order.
`price`	`float`	required	gt 0, unit `display_price`	Execution price in display money, already converted from ticks by the publisher. Contrast trade_log.price, which is ticks - the mismatch this `unit` declaration exists to make reviewable.
`quantity`	`int`	required	gt 0, unit `shares`	Matched quantity.

Field	Type	Presence	Rules	Description
`aggressor_side`	enum: `BUY`, `SELL`, `AUCTION`	required	—	Side that removed liquidity. AUCTION when both sides rested, which happens on an uncross print where there is no true aggressor.
`ts_ns`	`int`	required	ge 0, unit `epoch_nanos`	Match time in Unix epoch nanoseconds, exactly as the engine's clock read it – no scaling at publish time. The whole match batch shares one clock read, so trades printed in the same batch carry an identical value; `id` (run_seq + dense counter) is what orders them. Named `ts_ns` rather than `timestamp` so the unit travels with the field, as it does for `resume_at_ns` and `halt_duration_ns`: a bare float `timestamp` was indistinguishable by type from millis or nanos, which is what forced clearing's magnitude-guessing guard (finding CL-M6) and cost ~238ns of the engine's own precision.

Field	Type	Presence	Rules	Description
`tick_decimals`	`int`	defaults to `2`	ge 0, le 8, unit `dimensionless`	Decimal scale for `price`; 1 tick = 10^-tick_decimals.
`price_ticks`	`ticks`	omitted when unset	unit `ticks`	`price` in engine ticks. Canonical; the float beside it is for display. Scale: the symbol's tick_decimals.
`match_type`	enum: `CONTINUOUS`, `OPENING_AUCTION`, `CLOSING_AUCTION`, `REOPEN_AUCTION`, `RECOVERY_AUCTION`, `BACKSTOP`, `FORCE_UNCROSS`	omitted when unset	—	How the trade came about; the engine sets it on every trade (absent only on prints from before it existed). CONTINUOUS for every match with an aggressor (a stop an uncross triggers included); an uncross print (aggressor_side AUCTION) names its uncross: the scheduled opening or closing auction, a halted symbol's reopening, the restore of a crossed book on an engine without sessions, the closing backstop, or an operator's forced uncross.
`auction_id`	`string`	omitted when unset	max_len 32	The uncross this print belongs to: `auction.result.auction_id`.

Field	Type	Presence	Rules	Description
`trigger_order_id`	`string`	omitted when unset	max_len 64	The aggressor's order id when the aggressor is a stop that has triggered; absent otherwise.

Note

aggressor_side is AUCTION for uncross prints, where both sides rested and there is no true aggressor.

See also: `book.{SYMBOL}`, `order.fill.{GW_ID}`