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

OPERATOR'S GUIDE

A Complete Guide to Operating and Managing the EduMatcher Trading System

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 Operator's Guide
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.

Operator's Guide

`edumatcher`, Version: v0.77.0

Install, configure, run, and observe an EduMatcher exchange

Johan Persson <johan162@gmail.com>

Generated on November 8, 2026

Chapters

1 How to Use This Book 1

2 Installation 3

3 VM Runtime Image with Pinned PyPI Release 23

4 Inside the Container Deployment 29

5 The Configuration File, Block by Block 61

6 The Configuration Workflow 93

7 Configuration Verifier (`pm-cverifier`) 127

8 Configuration GUI (`config-gui`) 141

9 Example Engine Configs 183

10 Running the Exchange 185

11 The Admin Console and Exchange Commands 209

12 Running the AI-Trader Swarm 235

13 Listing a New Symbol (`pm-new-symbol`) 239

14 IPO Valuation (`pm-valuation`) 245

15 Session Scheduling and Auctions 301

16 Risk Controls 315

17 Statistics and Reporting 345

18 Market Index (`pm-index`) 377

19 Index Administration (`pm-index-admin-cli`) 391

20 The Market Model (`pm-market-sim`) 399

21 Closing the Day: Settlement, Statements and Housekeeping 403

22 ALF Gateway — Order Entry 425

23 BALF Gateway — Binary Order Entry 443

24 CALF Gateway — Market Data 459

25 RALF Gateway — Post-Trade Dissemination 483

26 API Gateway (REST/WebSocket) 495

27 Drop-Copy Gateway (`pm-dc-gwy`) 509

28 Persistence — Data Across Trading Sessions 517

29 Audit Trail 535

30 Audit Replay (pm-audit-replay) 559

31 Answering a Dispute 579

32 Centralized Log Server 583

33 Log Operator Console (`pm-log-ui`) 601

34 Health Monitoring (`pm-monitor`) 621

35 Operator Dashboard (`pm-dash`) 631

36 FAQ: Setup and Operation 641

Contents

1 How to Use This Book **1**
- 1.1 How the book is organized . . . 1
- 1.2 A first class session, chapter by chapter . . . 2

2 Installation **3**
- 2.1 Choosing an installation mode . . . 3
- 2.2 Containers: the whole system in one command . . . 4
  - 2.2.1 What the installer does . . . 5
  - 2.2.2 Installer options . . . 5
  - 2.2.3 Configuring it: the `.env` file . . . 5
  - 2.2.4 Everyday commands . . . 7
  - 2.2.5 Choosing what the exchange trades . . . 7
- 2.3 Containers from the source checkout . . . 8
  - 2.3.1 Flags that control the build . . . 8
  - 2.3.2 Flags that control a start . . . 8
  - 2.3.3 Settings in `.env` . . . 9
  - 2.3.4 Image build arguments . . . 10
- 2.4 How the containers are wired . . . 11
  - 2.4.1 One compose project, one private network . . . 11
  - 2.4.2 The two planes, and where each address is decided . . . 12
  - 2.4.3 What `0.0.0.0` does and does not expose . . . 13
  - 2.4.4 Why there is no longer a loopback rewrite . . . 14
  - 2.4.5 Two things that look like evidence and are not . . . 14
  - 2.4.6 The read-only API key . . . 15
- 2.5 Directories and paths . . . 15
  - 2.5.1 The data directory . . . 15
  - 2.5.2 Where that lands in each mode . . . 16
  - 2.5.3 Supporting directories . . . 16
- 2.6 VM bootstrap — a ready-to-run Multipass VM . . . 16
- 2.7 pipx — commands on your own machine . . . 17
- 2.8 Web applications with a pipx exchange . . . 18
  - 2.8.1 Step 1: the exchange . . . 18
  - 2.8.2 Step 2: the GUI containers . . . 18
  - 2.8.3 What the GUIs need from the exchange . . . 19
- 2.9 Poetry checkout — developing EduMatcher . . . 20
- 2.10 Shell completion . . . 20
- 2.11 Where to go next . . . 21

3 VM Runtime Image with Pinned PyPI Release **23**
- 3.1 Summary . . . 23
- 3.2 Why pip in a venv instead of pipx . . . 23
- 3.3 Build the VM . . . 24
  - 3.3.1 `mknode.sh` options . . . 24

3.4 What provisioning does . . . 26
3.4.1 Verify the install . . . 27
3.5 Operate the VM . . . 27
3.6 Snapshots, upgrades and reset . . . 27
3.7 Development nodes are a different thing . . . 28
3.8 Reference . . . 28

4 Inside the Container Deployment 29
4.1 Five container terms, as they apply here . . . 29
4.2 The two directories . . . 30
4.2.1 What is in each directory . . . 30
4.3 The images . . . 31
4.3.1 Eight images, one backend Dockerfile . . . 31
4.3.2 Why the whole exchange is one container . . . 32
4.3.3 The backend Dockerfile, stage by stage . . . 32
4.3.4 Build arguments of the backend image . . . 34
4.3.5 The web GUI images . . . 34
4.3.6 How released images are built . . . 35
4.4 What happens when the exchange container starts . . . 35
4.4.1 Step 1: deploying the configuration . . . 36
4.4.2 Step 4: stopping cleanly . . . 37
4.5 The compose files, one by one . . . 37
4.5.1 How compose files combine . . . 37
4.5.2 `${VARIABLE:-default}`: where the values come from . . . 38
4.5.3 `compose.yaml` — the exchange . . . 38
4.5.4 `compose.zmq.yaml` — the raw message bus . . . 39
4.5.5 `compose.ssh.yaml` — SSH into the exchange . . . 40
4.5.6 `compose.guis.yaml` — the four connected web GUIs . . . 40
4.5.7 `compose.config-gui.yaml` — the configuration builder . . . 42
4.5.8 `compose.ai-sim.yaml` — the self-trading market . . . 42
4.6 Projects, networks and pods . . . 43
4.6.1 The compose project and its names . . . 44
4.6.2 The network . . . 44
4.6.3 Pods (Podman only) . . . 45
4.7 Volumes and mounts . . . 46
4.8 Ports and why each one exists . . . 49
4.8.1 The exchange container . . . 49
4.8.2 The GUI containers . . . 51
4.9 Every variable, in one place . . . 51
4.9.1 `deployment/docker/.env` . . . 51
4.9.2 `~/.edumatcher/.env` (curl install) . . . 52
4.9.3 `~/.edumatcher-sim/.env` (curl `--sim` install) . . . 52
4.9.4 Variables only `entrypoint.sh` reads . . . 52
4.10 Using the helper scripts . . . 52
4.10.1 The `Makefile` in `deployment/docker` . . . 52
4.10.2 `install.sh` (curl) . . . 55
4.10.3 `edumatcher.sh` (curl) . . . 56
4.11 Doing it by hand . . . 56
4.11.1 Rebuilding the exchange image . . . 57
4.11.2 Rebuilding a GUI image . . . 57
4.11.3 Starting with compose directly . . . 58
4.11.4 The released stack by hand . . . 58
4.11.5 Make target to raw command . . . 58
4.12 Common situations . . . 59

5 The Configuration File, Block by Block 61
5.1 The shape of the file . . . 61

5.1.1 Conventions used by every block . . . 63
5.2 Splitting the file with `include` . . . 64
5.3 Step 1: the smallest working file . . . 65
5.3.1 `symbols` . . . 65
5.3.2 `participants` . . . 66
5.3.3 `sessions_enabled` . . . 67
5.4 Step 2: reference data . . . 67
5.5 Step 3: the trading day . . . 68
5.5.1 `schedule` . . . 69
5.5.2 `locale` . . . 69
5.6 Step 4: price protection . . . 70
5.6.1 The global switches . . . 72
5.6.2 `risk_controls` . . . 72
5.6.3 `circuit_breaker_defaults` . . . 72
5.6.4 Per-symbol exceptions . . . 74
5.7 Step 5: market makers and seeded liquidity . . . 74
5.7.1 The rule you will trip over first . . . 76
5.7.2 `participant_defaults` . . . 76
5.7.3 More `participants` fields . . . 76
5.7.4 `mm_obligation_defaults` . . . 77
5.7.5 `symbols.<SYMBOL>.market_maker_quotes` . . . 77
5.7.6 `market_maker_combos` . . . 78
5.8 Step 6: engine tuning . . . 78
5.9 Step 7: a market index . . . 79
5.10 Step 8: the access points . . . 80
5.10.1 `alf_gateway` — text order entry (`pm-alf-gwy`) . . . 82
5.10.2 `balf_gateway` — binary order entry (`pm-balf-gwy`) . . . 82
5.10.3 `market_data_gateway` — market data (`pm-md-gwy`, CALF) . . . 83
5.10.4 `post_trade_gateway` — post-trade feed (`pm-ralf-gwy`, RALF) . . . 83
5.10.5 `dc_gateway` — drop copy (`pm-dc-gwy`) . . . 84
5.10.6 `api_gateways` — REST and WebSocket (`pm-api-gwy`) . . . 84
5.11 Step 9: operations — logging and health . . . 85
5.11.1 `log_server` — central log collector (`pm-log-srv`) . . . 86
5.11.2 `monitor` — health beacons (`pm-monitor`) . . . 87
5.12 The complete file . . . 88
5.13 Ports at a glance . . . 90
5.14 Where to go next . . . 91

6 The Configuration Workflow 93
6.1 Configuration Workflow . . . 93
6.2 Configuring the Exchange . . . 93
6.3 File Location . . . 94
6.4 Generate Configs with `pm-config-gen` . . . 95
6.5 Generate Configs with `config-gui` . . . 95
6.6 Verify Configs with `pm-cverifier` . . . 95
6.6.1 Quick start . . . 95
6.6.2 Important behavior . . . 96
6.6.3 Option reference . . . 96
6.6.4 Keys for many participants . . . 107
6.6.5 `--participants` format . . . 107
6.6.6 `--symbol-opts` format . . . 108
6.6.7 `--combo` format . . . 109
6.6.8 Practical recipes . . . 110
6.7 Compile Configs with `pm-config-deploy` . . . 115
6.7.1 Add a symbol with `pm-new-symbol` . . . 115
6.7.2 Deploying example configurations . . . 115
6.7.3 What the artifact records about itself . . . 117

6.7.4 Missing-file Behavior . . . 117
6.8 Inspect Configs with `pm-config-show` . . . 118
6.8.1 Adapting to the terminal . . . 119
6.8.2 Options . . . 119
6.8.3 PDF output . . . 120
6.9 Choosing a Starting Point . . . 120
6.10 Configuration Checklist . . . 121
6.11 Startup and Persistence Order . . . 121
6.12 Adding or Removing Symbols . . . 122
6.13 Validation Commands . . . 122
6.14 Verifying the Deployed Artifact . . . 122
6.14.1 The content digest (an error) . . . 123
6.14.2 The source staleness check (a warning) . . . 124
6.14.3 Checking both by hand . . . 124
6.14.4 Fixing a digest mismatch . . . 124
6.15 See Also . . . 124

7 Configuration Verifier (`pm-cverifier`) 127
7.1 Why a Config Verifier? . . . 127
7.2 Quick Start . . . 127
7.3 CLI Reference . . . 128
7.4 Verification Layers . . . 128
7.5 Understanding the Output . . . 130
7.5.1 Text output (default) . . . 130
7.5.2 JSON output (`--format json`) . . . 131
7.6 Check Catalogue . . . 131
7.6.1 Layer 1 — YAML Syntax (`Y001`–`Y005`) . . . 131
7.6.2 Layer 2 — Schema (`S001`–`S131`) . . . 131
7.6.3 Layer 3 — Semantic (`M001`–`M033`) . . . 137
7.6.4 Layer 4 — Completeness (`C001`–`C013`) . . . 138
7.7 Using in CI . . . 139
7.8 The Risk Summary . . . 139
7.9 Relationship to Other Tools . . . 140

8 Configuration GUI (`config-gui`) 141
8.1 Overview . . . 141
8.2 Prerequisites . . . 143
8.2.1 Running the container (recommended for most users) . . . 143
8.2.2 Local development . . . 143
8.2.3 Optional: server-side verification . . . 143
8.2.4 Browser . . . 143
8.3 Running the application . . . 143
8.3.1 Option 1 — One command with the container stack . . . 144
8.3.2 Option 2 — A pre-built image artifact . . . 144
8.3.3 Option 3 — Local development . . . 144
8.4 A tour of the interface . . . 145
8.5 Personas . . . 146
8.6 The tabs at a glance . . . 148
8.7 Basics: symbols and gateways . . . 148
8.8 Core concepts and workflows . . . 149
8.8.1 Symbols and the IPO dialog . . . 149
8.8.2 Importing and exporting symbols . . . 151
8.8.3 Importing and exporting participants . . . 151
8.8.4 The Symbols workspace . . . 152
8.8.5 Symbol overview . . . 156
8.8.6 Risk and collars . . . 156
8.8.7 Circuit breakers . . . 159

8.8.8 Sessions and schedule . . . 159
8.8.9 Market makers . . . 159
8.8.10 Indices . . . 164
8.8.11 Combos . . . 164
8.8.12 Auxiliary gateways . . . 166
8.8.13 Health monitoring . . . 167
8.8.14 Engine tuning . . . 171
8.9 Diagnostics and live validation . . . 171
8.10 Importing an existing configuration . . . 174
8.11 Reviewing and exporting . . . 174
8.12 Deployment and operations . . . 177
8.12.1 Single container (recommended) . . . 177
8.12.2 Manual production build . . . 177
8.12.3 Backend environment variables . . . 177
8.12.4 Building behind a corporate proxy or firewall . . . 178
8.12.5 Optional: server-side verification . . . 178
8.12.6 Security notes . . . 178
8.13 How the GUI stays valid . . . 180
8.14 Architecture at a glance . . . 180
8.15 Troubleshooting . . . 180
8.16 See Also . . . 181

9 Example Engine Configs **183**
9.1 Scope . . . 183
9.2 How they are generated . . . 184
9.3 Training usage . . . 184
9.4 Related material . . . 184

10 Running the Exchange **185**
10.1 Operator model . . . 185
10.2 Three ways to run it . . . 185
10.3 Runtime source of truth . . . 186
10.4 Running modes . . . 186
10.5 Data directory . . . 187
10.6 Preflight checklist . . . 188
10.6.1 Preflight for a container install . . . 188
10.7 Minimum viable run . . . 189
10.7.1 Step 1 - start the engine . . . 189
10.7.2 Step 2 - connect two traders . . . 189
10.7.3 Step 3 - submit a test order . . . 189
10.8 Starting the stack with `pm-opctl-cli` . . . 189
10.8.1 The six built-in profiles . . . 190
10.8.2 What it gives you that a row of terminals does not . . . 190
10.9 Recommended startup order . . . 191
10.10 Starting each process . . . 196
10.10.1 Core engine . . . 196
10.10.2 Loss-sensitive recorders . . . 196
10.10.3 Session scheduler . . . 197
10.10.4 Interactive traders and operators . . . 197
10.10.5 Terminal observers . . . 197
10.10.6 Automation . . . 197
10.10.7 External order entry . . . 197
10.10.8 External market data, post-trade and API services . . . 197
10.10.9 Centralized operational logs . . . 197
10.10.10 TapeDeck trader information terminal . . . 198
10.10.11 Order book viewer (`pm-book`) . . . 198
10.10.12 Operator dashboard (`pm-dash`) . . . 198

10.11 Process groups by scenario . . . 198
10.12 The `tools/launch_all.sh` convenience launcher . . . 199
10.13 Verifying the system is running correctly . . . 199
10.13.1 Immediate checks after startup . . . 199
10.14 Monitoring a running exchange . . . 200
10.14.1 Operator surfaces . . . 200
10.14.2 Common operator queries . . . 201
10.14.3 Lightweight health check . . . 201
10.15 Logging levels . . . 202
10.16 Troubleshooting startup problems . . . 202
10.16.1 Engine exits immediately . . . 202
10.16.2 Gateway authentication timeout . . . 202
10.16.3 Viewer or board shows an empty book . . . 202
10.16.4 Stats, ticker or board show no activity . . . 203
10.16.5 Clearing and stats do not reconcile . . . 203
10.16.6 Scheduler transitions do not happen . . . 203
10.16.7 Admin command rejected . . . 203
10.16.8 External feed is reachable but silent . . . 203
10.16.9 TapeDeck loads but shows offline or missing history . . . 204
10.16.10 Container-specific problems . . . 204
10.17 Restart and shutdown . . . 204
10.17.1 Restarting non-engine processes . . . 204
10.17.2 Restarting the engine . . . 205
10.17.3 Full clean shutdown . . . 205
10.17.4 Shutting down a managed stack . . . 205
10.18 Appendix: operator playbooks . . . 205
10.18.1 Playbook 1 - five-minute local smoke test . . . 205
10.18.2 Playbook 2 - recorded classroom session . . . 206
10.18.3 Playbook 3 - external client integration test . . . 206
10.18.4 Playbook 4 - TapeDeck wallboard . . . 207
10.18.5 Playbook 5 - incident response: bad or unexpected fills . . . 207
10.18.6 Playbook 6 - config change between sessions . . . 207
10.19 See also . . . 208

11 The Admin Console and Exchange Commands 209
11.1 Concept . . . 209
11.1.1 How to send ADMIN commands . . . 209
11.2 ADMIN console (`pm-admin`) . . . 210
11.2.1 What you see at startup . . . 210
11.2.2 Console command syntax . . . 210
11.2.3 Example session . . . 212
11.2.4 Relationship to the other operator tools . . . 213
11.3 CLI tool (`pm-admin-cli`) . . . 213
11.3.1 Global flags . . . 213
11.3.2 Subcommands . . . 214
11.3.3 Shell scripting example . . . 219
11.4 `pm-index-cli` — Index Structural/Audit History Query Tool . . . 220
11.4.1 Global options . . . 220
11.4.2 Subcommands . . . 220
11.4.3 Output columns . . . 221
11.4.4 Examples . . . 221
11.4.5 `health` — Process and index file health . . . 222
11.4.6 No-row behaviour . . . 223
11.4.7 Plotting index level history (Python) . . . 223
11.5 `pm-index-admin-cli` — Index Corporate Action / Constituent Change CLI . . . 224
11.5.1 Subcommands . . . 224
11.5.2 Global options . . . 224

11.5.3 Exit codes . . . 224
11.6 `ExchangeCommandClient` . . . 225
11.6.1 Constructor parameters . . . 225
11.6.2 Usage example . . . 225
11.7 Command reference . . . 225
11.8 Command details . . . 227
11.8.1 `halt_all` — Exchange-wide circuit-breaker halt . . . 227
11.8.2 `resume_all` — Lift the exchange-wide halt . . . 228
11.8.3 `kill_switch` — Cancel all exposure for a gateway . . . 228
11.8.4 `mass_cancel` — Cancel exposure for one symbol . . . 228
11.8.5 `quote_cancel` — Cancel a market-maker's quote for one symbol . . . 228
11.8.6 `gateway_kick` — Forcefully disconnect a gateway . . . 229
11.8.7 `book_depth` — L1 / L2 order-book snapshot . . . 229
11.8.8 `order_list` — Inspect a gateway's resting orders . . . 229
11.8.9 `price_level_orders` — Per-order price-level composition . . . 230
11.8.10 `symbol_list` — Discover configured instruments . . . 230
11.8.11 `session_advance` — Manually drive the trading day . . . 230
11.9 Full ADMIN operator workflow . . . 231
11.10 Extending the framework . . . 232
11.11 See also . . . 232

12 Running the AI-Trader Swarm **235**
12.1 What runs . . . 235
12.2 Before you start . . . 235
12.3 Starting it . . . 235
12.3.1 Command-line options . . . 236
12.3.2 `swarm.yaml` . . . 236
12.3.3 Sizing . . . 237
12.4 Logs and daily summaries . . . 237
12.5 When something goes down . . . 238
12.6 Stopping . . . 238
12.7 Compressed soak . . . 238

13 Listing a New Symbol (`pm-new-symbol`) **239**
13.1 What an IPO is, and what the exchange does . . . 239
13.2 Quick start . . . 239
13.3 What the command does . . . 240
13.4 Which file is edited . . . 241
13.5 The IPO price . . . 241
13.6 The seed quote . . . 241
13.6.1 When one is written . . . 241
13.6.2 How it is priced . . . 242
13.7 Optional sections: `--field` . . . 242
13.8 Symbol names . . . 242
13.9 Why it refused . . . 243
13.10 The IPO-day checklist . . . 243
13.11 Options . . . 244
13.12 Where to go next . . . 244

14 IPO Valuation (`pm-valuation`) **245**
14.1 Going public: what an IPO is . . . 245
14.1.1 Who takes part . . . 246
14.1.2 Primary and secondary markets . . . 246
14.2 The pricing dilemma: science and black magic . . . 246
14.3 Money has a time value . . . 247
14.3.1 Interest and compounding . . . 247
14.3.2 Present value and discounting . . . 247

14.3.3 The risk-free rate . . . 248
14.3.4 Risk and the required return . . . 248
14.3.5 Streams of cash: NPV, perpetuities and growth . . . 248
14.4 What a company is worth . . . 249
14.4.1 Three ways to value a company . . . 249
14.4.2 From profit to cash . . . 249
14.4.3 Enterprise value and equity value . . . 250
14.5 Forecasting a young company . . . 251
14.5.1 How big can it get? TAM, SAM and the reachable share . . . 251
14.5.2 Customers, churn and price . . . 251
14.5.3 People and costs . . . 252
14.6 The discounted cash flow, step by step . . . 252
14.6.1 What `pm-valuation` does differently . . . 253
14.7 Comparables: what the market pays . . . 253
14.8 Uncertainty: scenarios, sensitivity and Monte Carlo . . . 253
14.9 The IPO process . . . 254
14.9.1 Preparing: the banks and the prospectus . . . 254
14.9.2 The price range and the IPO discount . . . 255
14.9.3 The roadshow and book-building . . . 255
14.9.4 Pricing and allocation . . . 255
14.9.5 The first day and after . . . 256
14.10 The black magic: what the formulas cannot tell you . . . 256
14.11 Using pm-valuation . . . 257
14.12 Quick start . . . 257
14.13 Sweden or the United States . . . 257
14.14 Sector presets . . . 258
14.15 The interview . . . 261
14.15.1 Levels: how much the interview asks . . . 262
14.15.2 Automatic values and their source . . . 262
14.15.3 Typing values . . . 263
14.15.4 Keys . . . 263
14.16 The live preview . . . 264
14.16.1 Fair value . . . 264
14.16.2 DCF . . . 265
14.16.3 Comps . . . 265
14.16.4 Range . . . 265
14.16.5 Offer price and coverage . . . 266
14.16.6 Verdict . . . 266
14.16.7 Notes . . . 266
14.17 The report . . . 267
14.17.1 How the offer price is chosen . . . 268
14.17.2 The PDF report . . . 268
14.18 Scenario files . . . 268
14.19 Listing the result . . . 269
14.20 Exit status . . . 269
14.21 Options . . . 269
14.22 Field reference . . . 270
14.22.1 Page 1 — Company . . . 270
14.22.2 Page 2 — Market . . . 274
14.22.3 Page 3 — Customers & pricing . . . 275
14.22.4 Page 4 — People . . . 277
14.22.5 Page 5 — Costs . . . 280
14.22.6 Page 6 — Capital & tax . . . 282
14.22.7 Page 7 — Discount rates . . . 285
14.22.8 Page 8 — Offering . . . 289
14.22.9 Page 9 — Investors & sentiment . . . 293
14.22.10 Page 10 — Index . . . 295

14.22.11 Page 11 — Management . . . 297
14.22.12 Page 12 — Simulation . . . 298
14.23 Where to go next . . . 300

15 Session Scheduling and Auctions **301**
15.1 What are auctions? . . . 301
15.1.1 Why auctions exist . . . 301
15.1.2 EduMatcher's auction model . . . 302
15.2 The trading date . . . 302
15.2.1 Setting the session timezone . . . 302
15.2.2 Relationship to `locale.country` . . . 303
15.2.3 Trading days are not always 24 hours . . . 303
15.3 Session phases . . . 303
15.3.1 Order acceptance rules by phase . . . 304
15.3.2 Transition side-effects . . . 304
15.3.3 Valid state transitions . . . 305
15.4 The session scheduler (`pm-scheduler`) . . . 305
15.4.1 Starting the scheduler . . . 306
15.4.2 CLI flag reference . . . 306
15.4.3 After the close: `--on-close` . . . 306
15.4.4 Configuring the schedule . . . 307
15.4.5 Session handling toggle . . . 308
15.4.6 Bank holidays and weekends . . . 309
15.4.7 Wall-clock re-checking . . . 309
15.4.8 Countdown to the next transition . . . 310
15.4.9 Built-in default schedule . . . 310
15.4.10 `--now` mode . . . 310
15.4.11 Compressed mode . . . 310
15.5 Equilibrium price . . . 311
15.5.1 Intuition . . . 311
15.5.2 Algorithm . . . 311
15.5.3 Worked example . . . 312
15.5.4 Uncross execution . . . 312
15.5.5 Published events . . . 313
15.6 Hands-on: driving an opening auction manually . . . 313
15.7 See also . . . 313

16 Risk Controls **315**
16.1 Overview . . . 315
16.2 Instrument halt state . . . 318
16.2.1 What it does . . . 318
16.2.2 State model . . . 318
16.2.3 Halt behaviour by order type . . . 318
16.2.4 Resuming . . . 319
16.2.5 Interaction with auctions . . . 319
16.3 Price collars . . . 319
16.3.1 Motivation . . . 319
16.3.2 Band definitions . . . 319
16.3.3 Validation logic . . . 320
16.3.4 Which orders are checked . . . 320
16.3.5 Configuration . . . 320
16.3.6 Global level profiles (L1/L2/L3 style) . . . 320
16.4 Order size and notional limits . . . 321
16.4.1 Motivation . . . 321
16.4.2 Which orders are checked . . . 322
16.4.3 Reject codes . . . 322
16.4.4 Configuration . . . 322

16.4.5 Querying the effective limits . . . 323
16.5 Circuit breakers . . . 323
16.5.1 Motivation . . . 323
16.5.2 How it works — step by step . . . 323
16.5.3 Rolling reference window . . . 324
16.5.4 Day one (IPO) behaviour . . . 324
16.5.5 Why a halt always reopens with an uncross . . . 325
16.5.6 Automated Corridor Expansion (ACE) . . . 325
16.5.7 Configuration . . . 327
16.5.8 Why there is no selected "default breaker level" . . . 328
16.6 Price collars vs circuit breakers . . . 329
16.6.1 Side-by-side comparison . . . 329
16.6.2 Practical interpretation . . . 329
16.7 Interaction between mechanisms . . . 329
16.7.1 Combined example . . . 330
16.8 ZeroMQ messages . . . 330
16.8.1 Circuit breaker halt . . . 331
16.8.2 Circuit breaker extend (ACE) . . . 331
16.8.3 Circuit breaker resume . . . 331
16.9 ADMIN-role operator controls . . . 331
16.9.1 Prerequisites . . . 332
16.9.2 Triggering an exchange-wide halt . . . 332
16.9.3 Resuming all trading . . . 333
16.9.4 Sequence diagram . . . 334
16.9.5 Key differences from automatic circuit breakers . . . 334
16.9.6 Halting or resuming a single symbol . . . 335
16.9.7 Symbol-level mass cancel . . . 335
16.10 Kill switch . . . 336
16.10.1 Permissions . . . 336
16.10.2 Sending the command . . . 336
16.10.3 What the engine does . . . 336
16.10.4 Reply . . . 336
16.10.5 Differences from a halt . . . 336
16.11 Self-Match Prevention (SMP) . . . 337
16.11.1 Motivation . . . 337
16.11.2 The four actions . . . 337
16.11.3 What the client sees . . . 337
16.11.4 Where SMP is checked — the matching-engine mechanics . . . 338
16.11.5 `FOK` — self-match liquidity is excluded before the pre-check . . . 339
16.11.6 Cancellation is visible, not silent . . . 339
16.11.7 Two ways to specify `smp_action` . . . 339
16.11.8 Precedence: why "omitted" and "explicit `NONE`" are not the same thing . . . 340
16.11.9 Which paths carry an explicit per-request `SMP=`, and which only have the gateway default . . . 341
16.11.10 Worked examples . . . 342
16.11.11 SMP and the other risk controls . . . 343
16.11.12 Configuration reference . . . 343
16.12 Market-maker interaction . . . 343
16.13 See also . . . 344

17 Statistics and Reporting **345**
17.1 Overview — Statistics Architecture . . . 345
17.2 Dates and Timestamps — read this first . . . 345
17.2.1 Setting the session timezone . . . 346
17.2.2 Inspecting what a database was recorded with . . . 346
17.2.3 What `--date` means . . . 347
17.2.4 What `--from` / `--to` accept . . . 347
17.3 Data Folder Location . . . 347

17.4 Prices Are Stored As Integer Ticks . . . 348
17.4.1 The conversion rule . . . 348
17.4.2 Why integers . . . 349
17.4.3 The exceptions . . . 349
17.4.4 `turnover` — stored in ticks, converted like a price . . . 349
17.4.5 What this means for exported data . . . 350
17.4.6 Residual float exposure . . . 350
17.5 The Statistics Database Schema . . . 350
17.5.1 Row identity and pagination cursors . . . 350
17.5.2 `instruments` . . . 350
17.5.3 `stats_meta` . . . 351
17.5.4 `daily_stats` . . . 351
17.5.5 `price_snapshots` . . . 352
17.5.6 `trade_log` . . . 353
17.5.7 `feed_gaps` . . . 354
17.5.8 `order_events` . . . 355
17.5.9 `index_daily_stats` . . . 357
17.5.10 `index_level_snapshots` . . . 357
17.6 Running the Statistics Recorder . . . 358
17.6.1 Retention . . . 360
17.6.2 Exit codes . . . 360
17.6.3 Restarting mid-session . . . 360
17.6.4 One recorder per database . . . 360
17.7 Querying with pm-stats-cli . . . 361
17.7.1 Basic Syntax . . . 361
17.7.2 Row limits and truncation . . . 361
17.7.3 Available Commands . . . 361
17.7.4 Order Lifecycle History Queries . . . 367
17.8 Output Formats . . . 368
17.8.1 Table Format (default) . . . 368
17.8.2 JSON Format . . . 368
17.8.3 CSV Format . . . 369
17.9 Common Analyst Workflows . . . 369
17.9.1 End-of-Day Summary Report . . . 369
17.9.2 Intraday Price Volatility Analysis . . . 369
17.9.3 Trade Flow Analysis . . . 369
17.9.4 Participant Performance Analysis . . . 370
17.9.5 Multi-Day Price Trends . . . 370
17.9.6 Index Level History . . . 370
17.9.7 Getting the EOD index level for a date . . . 370
17.9.8 Validation — Did the Trade Complete Correctly? . . . 371
17.9.9 Order Lifecycle Investigation . . . 372
17.10 Integration with Other Tools . . . 372
17.10.1 Combining with pm-ticker . . . 372
17.10.2 Exporting to BI Tools . . . 372
17.10.3 Python / Pandas Integration . . . 373
17.11 Troubleshooting . . . 373
17.11.1 No data recorded — where did the trades go? . . . 373
17.11.2 No index data recorded — where did the index updates go? . . . 374
17.11.3 Queries return "No rows found" but I know data should exist . . . 374
17.11.4 Database is locked or "unable to open" . . . 375
17.11.5 Snapshot times seem wrong or are missing . . . 375
17.11.6 VWAP calculation looks wrong . . . 375
17.12 See Also . . . 376

18 Market Index (`pm-index`) 377
18.1 What this process is . . . 377

18.2 Prerequisites . . . 377
18.3 Starting pm-index . . . 378
18.3.1 CLI options . . . 378
18.4 Configuration . . . 378
18.4.1 Minimal example . . . 379
18.4.2 Field reference . . . 379
18.4.3 Generating with pm-config-gen . . . 380
18.5 How Calculation Works . . . 380
18.5.1 Cap-weighted formula . . . 380
18.5.2 Divisor initialisation . . . 380
18.5.3 Update trigger . . . 381
18.5.4 Reference prices . . . 381
18.5.5 Publish throttle . . . 381
18.5.6 Intraday OHLC . . . 381
18.6 Corporate Actions . . . 381
18.6.1 Divisor adjustment formula . . . 381
18.6.2 Supported corporate actions . . . 381
18.6.3 Applying corporate actions . . . 382
18.6.4 Constituent changes . . . 383
18.7 State and History . . . 383
18.7.1 State file . . . 383
18.7.2 History file . . . 384
18.8 Getting Index Values . . . 385
18.8.1 From a gateway terminal (INDEX command) . . . 385
18.8.2 From the market-data gateway (CALF) . . . 385
18.8.3 Using pm-index-cli for structural/audit records . . . 386
18.8.4 Querying level/EOD history with pm-stats-cli . . . 387
18.8.5 Via the REST API . . . 387
18.8.6 Reading the structural/audit file directly . . . 387
18.9 Example Configurations . . . 388
18.9.1 Single index, three constituents . . . 388
18.9.2 Two indices with different base values and publish rates . . . 388
18.9.3 Full manual YAML for two indices . . . 389
18.10Operational Checklist . . . 389

19 Index Administration (pm-index-admin-cli) 391
19.1 What this tool is . . . 391
19.2 Why not pm-index-cli? . . . 391
19.3 Why there is no connect() / authentication step . . . 392
19.4 Starting point . . . 392
19.5 Unknown index ID fails fast . . . 392
19.6 Confirmation prompts and --dry-run . . . 393
19.7 Subcommand reference . . . 393
19.7.1 split — Apply a stock split . . . 393
19.7.2 dividend — Apply a cash dividend . . . 394
19.7.3 shares — Set shares outstanding (issuance or buy-back) . . . 394
19.7.4 add — Add a constituent . . . 395
19.7.5 delist — Remove a constituent . . . 395
19.7.6 history — Show recent structural/corp-action history . . . 395
19.8 Output formats . . . 396
19.9 Typical operator workflow . . . 396
19.10Further Reading . . . 397

20 The Market Model (pm-market-sim) 399
20.1 What it does . . . 399
20.2 The model . . . 399
20.3 market_sim.yaml . . . 400

20.4 Running it . . . 400
20.5 Anchoring a market maker . . . 401
20.6 News . . . 401
20.6.1 Generated news . . . 401
20.6.2 Publishing news yourself: `pm-news` . . . 401
20.6.3 Scenarios . . . 402

21 Closing the Day: Settlement, Statements and Housekeeping 403
21.1 The big picture . . . 403
21.1.1 What happens, in order . . . 404
21.1.2 What `pm-eod` does not do . . . 405
21.2 The ideas, from scratch . . . 405
21.2.1 Shares, symbols and trades . . . 405
21.2.2 Positions: long and short . . . 405
21.2.3 Average cost . . . 406
21.2.4 The closing price and "marking to market" . . . 406
21.2.5 Unrealized and realized profit and loss . . . 406
21.2.6 Equity . . . 406
21.2.7 Settlement . . . 406
21.2.8 The central pool (who pays whom) . . . 407
21.2.9 Balance and opening balance . . . 407
21.2.10 Currency, minor units and rounding . . . 407
21.2.11 The trading date . . . 407
21.2.12 Other words used on this page . . . 407
21.3 A complete worked example . . . 408
21.3.1 Day 1 — Monday 2026-11-02 . . . 408
21.3.2 Day 2 — Tuesday 2026-11-03 . . . 408
21.3.3 Day 3 — Wednesday 2026-11-04 (Alice cashes in) . . . 409
21.3.4 The same example, run for real . . . 409
21.4 Setting it up . . . 410
21.4.1 What must be running . . . 410
21.4.2 The roster file . . . 410
21.4.3 Starting the hook . . . 411
21.4.4 Running without the scheduler . . . 412
21.4.5 Where everything lives . . . 412
21.5 A normal day . . . 413
21.5.1 The balances . . . 413
21.5.2 One day in detail . . . 414
21.5.3 Exit codes of `pm-eod run` . . . 414
21.6 The files `pm-eod` writes . . . 414
21.6.1 The statement (`statements/<gateway>.md`) . . . 415
21.6.2 The same statement as JSON and CSV . . . 415
21.6.3 `instructions.csv` — the payment instructions . . . 416
21.6.4 `leaderboard.csv` . . . 416
21.6.5 `summary.md` / `summary.json` — for the instructor . . . 417
21.6.6 Sending the statements . . . 417
21.7 The housekeeping steps in detail . . . 417
21.7.1 `audit` — the audit-trail safety net . . . 417
21.7.2 `logs` — compressing and tidying the fallback logs . . . 418
21.7.3 `backup` — verified copies of the databases . . . 418
21.7.4 Disk space . . . 419
21.8 Correcting mistakes and special situations . . . 419
21.8.1 Run a day again . . . 419
21.8.2 Look before you leap: `--dry-run` . . . 419
21.8.3 The scheduler was down, or the machine was off . . . 419
21.8.4 The engine was restarted during the day . . . 419
21.8.5 A trader joins or leaves mid-way . . . 420

21.8.6 Recomputing a day (`--redo`) . . . 420
21.8.7 After `pm-opctl-cli clear` . . . 420
21.8.8 Rehearsing with a fast clock . . . 420
21.8.9 Weekends and holidays . . . 420
21.9 Troubleshooting . . . 420
21.9.1 Gate failures (exit code 1) . . . 420
21.9.2 Settlement failures (exit code 1) . . . 421
21.9.3 Report failures (exit code 1, settlement kept) . . . 421
21.9.4 Other . . . 422
21.9.5 Housekeeping warnings (exit code 2) . . . 422
21.10 Questions beginners ask . . . 422
21.11 Checklist for your first real day . . . 423
21.12 Command summary . . . 423
21.13 See also . . . 423

22 ALF Gateway — Order Entry 425
22.1 What this process is . . . 425
22.1.1 What this is not . . . 425
22.2 Prerequisites . . . 426
22.3 Configuration . . . 426
22.3.1 One live connection per ID, and the engine heartbeat . . . 427
22.4 Start the gateway . . . 427
22.5 Quick connect test . . . 428
22.6 Session lifecycle . . . 428
22.6.1 Step 1 — Send `HELLO` . . . 429
22.6.2 Step 2 — Authentication round trip . . . 430
22.6.3 Step 3 — Receive `WELCOME` . . . 430
22.6.4 Step 4 — Send commands . . . 430
22.6.5 Step 5 — Disconnect . . . 430
22.7 Command reference . . . 430
22.7.1 `NEW` — submit order . . . 430
22.7.2 `AMEND` — amend resting order . . . 431
22.7.3 `CANCEL` — cancel order / OCO / combo . . . 432
22.7.4 `QUOTE` — submit/replace two-sided quote (MARKET_MAKER role) . . . 432
22.7.5 `QUOTE_CANCEL` — cancel active quote (MARKET_MAKER role) . . . 432
22.7.6 `KILL` — gateway kill-switch . . . 432
22.7.7 `DC` — toggle drop-copy relay . . . 432
22.7.8 `SYMBOLS` — instrument list . . . 432
22.7.9 `ORDERS` — resting order list . . . 433
22.7.10 `QBOOT` — quote bootstrap state . . . 433
22.7.11 `QLEGS` — quote leg snapshot (active + recent) . . . 433
22.7.12 `POS|GW=<gateway_id>` — query another gateway's position . . . 434
22.7.13 `SESSION` — query current trading session state . . . 434
22.7.14 `PING` / `EXIT` . . . 434
22.8 Broadcast events . . . 435
22.9 Engine-scoped events (per-gateway) . . . 435
22.9.1 Unsolicited cancels . . . 436
22.10 Error codes . . . 436
22.11 Example libraries and interactive clients . . . 438
22.11.1 Python . . . 438
22.11.2 C . . . 439
22.11.3 Building your own application . . . 439
22.12 When to use `pm-alf-gwy` vs. the alternatives . . . 440
22.13 Troubleshooting . . . 440
22.13.1 Check whether the port is in use . . . 440
22.13.2 Test the TCP connection from the command line . . . 440
22.13.3 Common problems . . . 441

22.14 See also . . . 441

23 BALF Gateway — Binary Order Entry 443
23.1 What this process is . . . 443
23.1.1 Why BALF instead of ALF? . . . 444
23.1.2 What BALF `1.0.0` does not support . . . 445
23.2 Prerequisites . . . 445
23.3 Configuration . . . 445
23.4 Start the gateway . . . 446
23.5 Quick connectivity test . . . 447
23.6 Session lifecycle . . . 448
23.6.1 Step 1 — Send LOGON . . . 449
23.6.2 Step 2 — Receive LOGON_ACK . . . 449
23.6.3 Step 3 — Send order messages . . . 450
23.6.4 Step 4 — Receive server messages . . . 450
23.6.5 Step 5 — Disconnect . . . 450
23.7 Message reference . . . 450
23.7.1 Client → server messages . . . 450
23.7.2 Server → client messages . . . 450
23.8 Order types and field encoding . . . 451
23.8.1 Supported order types . . . 451
23.8.2 Price encoding . . . 451
23.8.3 ORDER_ACK reject codes . . . 451
23.9 Annotated wire-frame examples . . . 452
23.10 Server-initiated messages . . . 453
23.10.1 HEARTBEAT . . . 453
23.10.2 EXECUTION_REPORT . . . 453
23.10.3 CANCEL_ACK . . . 453
23.10.4 AMEND_ACK . . . 454
23.11 Disconnect behaviour and passive-order policy . . . 454
23.12 Duplicate session policy . . . 454
23.13 Example: minimal Python BALF client . . . 454
23.14 Comparing `pm-balf-gwy` with the alternatives . . . 456
23.15 Troubleshooting . . . 456
23.15.1 Check whether the port is in use . . . 456
23.15.2 Non-interactive TCP reachability test . . . 456
23.15.3 Common problems . . . 456
23.16 See also . . . 458

24 CALF Gateway — Market Data 459
24.1 What this process is . . . 459
24.2 The feed at a glance . . . 461
24.2.1 Message catalogue . . . 461
24.2.2 The envelope on every market-data line . . . 462
24.2.3 The seven channels . . . 462
24.3 Prerequisites . . . 463
24.4 Start the gateway . . . 464
24.5 Quick connect test . . . 465
24.6 What information is available . . . 466
24.6.1 Channel `TOP` — best bid, ask, and last trade . . . 466
24.6.2 Channel `TRADE` — every trade print . . . 467
24.6.3 Channel `STATE` — session and symbol states . . . 467
24.6.4 Channel `INDEX` — index level updates . . . 468
24.6.5 Channel `DEPTH` — aggregated multi-level order book . . . 468
24.6.6 Channel `AUCTION` — auction uncross results . . . 469
24.6.7 Channel `CB` — circuit-breaker detail . . . 470
24.6.8 Channel summary . . . 471

24.7 When to use CALF — protocol comparison . . . 471
24.8 Connecting and subscribing . . . 472
24.8.1 Step 1 — Send HELLO . . . 472
24.8.2 Step 2 — Subscribe . . . 472
24.8.3 Step 3 — Receive snapshots . . . 472
24.8.4 Step 4 — Cancel subscriptions . . . 473
24.8.5 Step 5 — Handle heartbeats . . . 473
24.8.6 Step 6 — Disconnect . . . 473
24.9 Subscribing to a targeted subset . . . 473
24.10 Gap detection and replay recovery . . . 474
24.11 Examples and libraries . . . 474
24.11.1 The production Python client: edumatcher.calf_client . . . 475
24.11.2 Zero-dependency minimal client . . . 476
24.11.3 Using the calf_parser.py library . . . 476
24.11.4 Annotated end-to-end subscriber . . . 476
24.11.5 Run the bundled examples . . . 478
24.11.6 The recovery library in isolation: calf_recovery . . . 478
24.12 Common errors and fixes . . . 479
24.13 Operational checklist . . . 480
24.14 Building tools on top of the feed . . . 480
24.14.1 Anatomy of a robust client . . . 480
24.14.2 Browser clients need a server-side bridge . . . 480
24.15 See also . . . 481

25 RALF Gateway — Post-Trade Dissemination 483
25.1 What this process does . . . 483
25.2 Architecture position . . . 484
25.3 Prerequisites . . . 484
25.4 Quick start . . . 484
25.4.1 1. Start the engine . . . 484
25.4.2 2. Start the RALF gateway . . . 484
25.4.3 3. Connect an external client . . . 484
25.5 Configuration . . . 485
25.6 Generate config with pm-config-gen . . . 485
25.7 What data is available . . . 486
25.7.1 Channel CLEARING — post-trade clearing data . . . 486
25.7.2 Channel DROP_COPY — fill events for risk monitoring . . . 486
25.7.3 Channel AUDIT — full audit trail . . . 487
25.7.4 Channel summary . . . 487
25.8 When to use RALF — protocol comparison . . . 487
25.9 Connecting and subscribing . . . 487
25.9.1 Step 1 — Send HELLO . . . 488
25.9.2 Step 2 — Subscribe . . . 488
25.9.3 Step 3 — Receive the snapshot . . . 488
25.9.4 Step 4 — Cancel subscriptions . . . 488
25.9.5 Step 5 — Handle heartbeats . . . 489
25.9.6 Step 6 — Disconnect . . . 489
25.10 Gap detection and replay . . . 489
25.11 Quick connect test . . . 489
25.12 Python subscriber example . . . 490
25.12.1 Zero-dependency minimal client . . . 490
25.12.2 Using the ralf_parser.py library . . . 490
25.12.3 Annotated end-to-end subscriber . . . 490
25.12.4 Run the bundled examples . . . 492
25.13 Common errors and fixes . . . 492
25.14 Operational notes . . . 492
25.15 Dedicated Gateway Runbook (pm-ralf-gwy) . . . 492

25.15.1 Start commands . . . 492
25.15.2 Minimal client probe . . . 493
25.15.3 Reconnect replay behavior . . . 493
25.15.4 Fast error triage . . . 493
25.15.5 Operator checklist . . . 493
25.16 See also . . . 493

26 API Gateway (REST/WebSocket) 495
26.1 What this process is . . . 495
26.2 Configuration . . . 496
26.3 Start the process . . . 497
26.4 Swagger interface . . . 498
26.5 Authentication principles . . . 498
26.5.1 Auth and roles . . . 498
26.5.2 Key types and provisioning . . . 499
26.6 REST endpoints . . . 501
26.6.1 Order entry in practice . . . 501
26.6.2 Reference data . . . 502
26.6.3 Book snapshots . . . 502
26.6.4 History endpoints . . . 502
26.7 Bootstrap endpoints . . . 503
26.8 Admin endpoints . . . 503
26.8.1 Index administration (`pm-index` bridge) . . . 503
26.8.2 `admin.action` monitor events . . . 503
26.9 Cross-gateway admin views . . . 503
26.9.1 Order cache retention . . . 504
26.10 WebSocket endpoints . . . 504
26.11 Examples . . . 504
26.12 Implementation notes and design deviations . . . 505
26.13 Operational checklist . . . 505
26.14 Minimal MARKET order CLI . . . 505
26.15 Troubleshooting . . . 506
26.15.1 Check whether the port is in use . . . 506
26.15.2 Test HTTP connectivity from the command line . . . 506
26.15.3 Common problems . . . 507

27 Drop-Copy Gateway (`pm-dc-gwy`) 509
27.1 What this process is . . . 509
27.1.1 What this is not . . . 509
27.2 Prerequisites . . . 510
27.3 Configuration . . . 510
27.4 Start the gateway . . . 511
27.5 Quick connect test . . . 511
27.6 Session lifecycle . . . 512
27.6.1 Step 1 — Send `HELLO` . . . 512
27.6.2 Step 2 — Receive `WELCOME` . . . 512
27.6.3 Step 3 — Receive fills . . . 513
27.6.4 Step 4 — Disconnect . . . 513
27.7 Protocol reference . . . 513
27.7.1 Client → Gateway . . . 513
27.7.2 Gateway → Client . . . 513
27.8 Error codes . . . 513
27.9 Minimal zero-dependency client (Python) . . . 514
27.10 When to use `pm-dc-gwy` vs. the alternatives . . . 514
27.11 Troubleshooting . . . 515
27.11.1 Check whether the port is in use . . . 515
27.11.2 Test the TCP connection from the command line . . . 515

27.11.3 Common problems . . . 515
27.12 See also . . . 515

28 Persistence — Data Across Trading Sessions 517
28.1 Data files at a glance . . . 517
28.2 How It Works . . . 524
28.2.1 At Shutdown (Ctrl-C / SIGTERM on the engine) . . . 524
28.2.2 At Startup . . . 525
28.2.3 At the Close (every trading day) . . . 525
28.3 Impact of a Business-Day Change . . . 526
28.4 Operational edge cases . . . 526
28.5 Engine State Files In Detail . . . 527
28.5.1 The `gtc_orders.json` File . . . 527
28.5.2 The `engine_run_seq.json` File . . . 528
28.5.3 The `book_stats.json` File . . . 528
28.5.4 The `gtc_combos.json` File . . . 528
28.6 Working with GTC Orders . . . 529
28.6.1 Submitting a GTC Order . . . 529
28.6.2 Cancelling a GTC Order . . . 529
28.6.3 Order ID Stability . . . 529
28.7 General Startup / Shutdown Sequence . . . 529
28.8 Other Persistent Files . . . 531
28.8.1 `src/data/audit.log` . . . 531
28.8.2 `src/data/clearing.db` . . . 531
28.8.3 `src/data/stats.db` . . . 532
28.9 Summary of All Data Files . . . 534
28.10 See also . . . 534

29 Audit Trail 535
29.1 Overview . . . 535
29.1.1 Why a full event log matters . . . 536
29.2 Data Directory . . . 537
29.3 pm-audit — Event Recorder . . . 537
29.3.1 Startup options . . . 537
29.3.2 Write buffering . . . 538
29.3.3 What can be lost . . . 538
29.3.4 Log format . . . 539
29.3.5 Day parts and archives . . . 540
29.3.6 Proving a day is unchanged: the evidence manifest . . . 541
29.3.7 Subscribed topics . . . 541
29.3.8 Audit completeness — every state-changing decision, not just its outcome . . . 542
29.3.9 Refusals made at a gateway . . . 544
29.3.10 Which clock a line is stamped with . . . 544
29.3.11 Commands the trail may not hold . . . 544
29.3.12 Signal handling and graceful shutdown . . . 544
29.4 pm-audit-cli — Query Tool . . . 544
29.4.1 Why a separate query tool . . . 545
29.4.2 Global options . . . 545
29.4.3 Which files are read . . . 546
29.4.4 `events` — General event search . . . 546
29.4.5 `orders` — Order lifecycle investigation . . . 548
29.4.6 `trades` — Trade execution search . . . 549
29.4.7 `topics` — Topic discovery and statistics . . . 550
29.4.8 `gateways` — Gateway activity summary . . . 551
29.4.9 `timeline` — Chronological event stream . . . 552
29.4.10 `stats` — Log file statistics . . . 553
29.4.11 `index` — Build or update the SQLite index . . . 554

29.4.12 `health` — Recorder and log file health . . . 555
29.5 Cookbook — Common Audit Workflows . . . 557
29.5.1 Investigate a specific order . . . 557
29.5.2 Reconstruct a trade . . . 557
29.5.3 Verify session phases fired correctly . . . 557
29.5.4 Identify the most active gateway . . . 557
29.5.5 Check log health before end of day . . . 557
29.5.6 Export a full session for offline analysis . . . 557
29.5.7 Look at an archived day . . . 558
29.5.8 Watch a live training session . . . 558
29.6 Error Handling . . . 558
29.7 See Also . . . 558

30 Audit Replay (pm-audit-replay) 559
30.1 Background . . . 559
30.2 How it works . . . 560
30.2.1 Four layers . . . 560
30.2.2 Where the causality comes from . . . 560
30.2.3 Ordering . . . 561
30.2.4 The episode index . . . 561
30.3 Subcommands . . . 562
30.3.1 `story` selectors . . . 562
30.3.2 Per-subcommand options . . . 563
30.4 Global options . . . 563
30.4.1 Source . . . 563
30.4.2 Window . . . 564
30.4.3 Filters . . . 564
30.4.4 Output . . . 564
30.4.5 Colour . . . 565
30.4.6 Detection . . . 566
30.4.7 Exit codes . . . 566
30.5 What it can catch . . . 566
30.5.1 Lifecycle . . . 566
30.5.2 Quantity and price conservation . . . 567
30.5.3 Ordering, clocks and the envelope . . . 568
30.5.4 Command reconciliation . . . 569
30.5.5 Coverage — the tool admitting what it does not know . . . 569
30.5.6 What a clean run does not prove . . . 570
30.6 Examples . . . 570
30.6.1 What happened to this order? . . . 570
30.6.2 …and show me the evidence . . . 571
30.6.3 What did the exchange do in the last fifteen minutes? . . . 571
30.6.4 Is anything wrong? . . . 571
30.6.5 Did a kill switch do what it said? . . . 571
30.6.6 Follow one whole causal chain . . . 571
30.6.7 What is worth looking at first? . . . 571
30.6.8 Every rejected order as a table . . . 571
30.6.9 Is the trail itself trustworthy? . . . 572
30.6.10 Re-executing the trail . . . 572
30.6.11 Where was my order in the queue? . . . 573
30.6.12 Feed it to something else . . . 573
30.6.13 Read a log with no index at all . . . 573
30.7 Outputs . . . 573
30.7.1 Detail levels . . . 573
30.7.2 `--format text` . . . 574
30.7.3 `--format ndjson` . . . 574
30.7.4 `--format json` . . . 575

30.7.5 `--format markdown` . . . 575
30.7.6 `--format csv` . . . 575
30.7.7 Which formats each subcommand produces . . . 575
30.8 Tips & tricks . . . 576
30.9 See Also . . . 576

31 Answering a Dispute **579**
31.1 Which command answers which question . . . 579
31.2 The procedure . . . 580
31.3 Anchoring the head hash . . . 580
31.4 What the trail cannot show . . . 581
31.5 See Also . . . 581

32 Centralized Log Server **583**
32.1 Overview . . . 583
32.2 Data Directory . . . 584
32.3 pm-log-srv — Log Collector . . . 584
32.3.1 Startup options . . . 585
32.3.2 Retention . . . 586
32.4 LALF — The Wire Protocol at a Glance . . . 586
32.5 LALF-PS — The ZeroMQ Log Distribution Interface . . . 586
32.5.1 Socket topology . . . 586
32.5.2 `sub_id` — the routing key . . . 587
32.5.3 Two modes: `STREAM` and `NOTIFY` . . . 587
32.5.4 Filtering . . . 587
32.5.5 Backfill — "the last n minutes" . . . 588
32.5.6 Liveness — why leases . . . 588
32.5.7 The other failure: a subscriber that is slow, not dead . . . 589
32.5.8 Server liveness . . . 589
32.5.9 Ordering, durability and the slow-joiner race . . . 589
32.5.10 Error replies . . . 589
32.5.11 A minimal subscriber . . . 590
32.5.12 Tuning . . . 592
32.5.13 Troubleshooting . . . 592
32.6 pm-log-cli — Query/Troubleshooting Tool . . . 592
32.6.1 Global options . . . 593
32.6.2 Exit codes . . . 593
32.6.3 `query` — Filtered historical search . . . 593
32.6.4 `tail` — Follow new rows in real time . . . 594
32.6.5 `processes` — Connected/recently-connected processes . . . 595
32.6.6 `stats` — Server + database health summary . . . 595
32.6.7 `diagnose` — Rule-based troubleshooting report . . . 596
32.6.8 `prune` — Manual retention pruning . . . 597
32.7 Cookbook — Common Log-Server Workflows . . . 597
32.7.1 Watching a live session . . . 597
32.7.2 Investigating an error after the fact . . . 598
32.7.3 Checking whether every process is still reporting in . . . 598
32.7.4 Free-text searching across the whole system . . . 598
32.7.5 Exporting for external analysis . . . 598
32.7.6 Routine maintenance . . . 598
32.8 Error Handling . . . 598
32.9 See Also . . . 599

33 Log Operator Console (`pm-log-ui`) **601**
33.1 Overview . . . 601
33.2 How it relates to the log server . . . 602
33.3 LALF, LALF-PS, and where the console sits . . . 602

33.3.1 What the bridge does on the LALF-PS wire . . . 603
33.4 Running the application . . . 603
33.4.1 Prerequisites . . . 603
33.4.2 Option 1 — The whole stack (recommended) . . . 604
33.4.3 Option 2 — This app alone, in a container . . . 604
33.4.4 Option 3 — A pre-built image artifact . . . 605
33.4.5 Option 4 — Local development . . . 605
33.4.6 Developer commands . . . 606
33.5 A tour of the interface . . . 606
33.5.1 The six views at a glance . . . 606
33.5.2 Dashboard . . . 607
33.5.3 Logs (the Explorer) . . . 607
33.5.4 Alerts . . . 610
33.5.5 Connections . . . 610
33.5.6 Diagnostics . . . 613
33.5.7 Health . . . 613
33.6 Exporting . . . 613
33.7 Configuration reference . . . 616
33.7.1 Network and serving . . . 616
33.7.2 Sources . . . 616
33.7.3 Issues . . . 616
33.7.4 Thresholds . . . 617
33.7.5 Limits . . . 617
33.7.6 How the frontend learns these . . . 617
33.8 Security notes . . . 618
33.9 Troubleshooting . . . 618
33.10 Keeping the console in sync with `pm-log-srv` . . . 619
33.11 See Also . . . 619

34 Health Monitoring (`pm-monitor`) 621
34.1 Overview . . . 621
34.1.1 Which processes beacon . . . 622
34.2 Starting `pm-monitor` . . . 622
34.2.1 In the container . . . 623
34.3 How health is judged . . . 623
34.4 Thresholds and events . . . 624
34.5 Rates and "today" . . . 625
34.6 `metrics.db` . . . 625
34.7 Configuration . . . 626
34.8 `pm-monitor-cli` . . . 627
34.8.1 `health` . . . 627
34.8.2 `status` . . . 628
34.8.3 `events` . . . 628
34.8.4 Cookbook . . . 628
34.9 Troubleshooting . . . 629
34.10 See also . . . 629

35 Operator Dashboard (`pm-dash`) 631
35.1 Overview . . . 631
35.2 Running the dashboard . . . 633
35.2.1 Option 1 — The whole stack (recommended) . . . 633
35.2.2 Option 2 — This app alone, in a container . . . 633
35.2.3 Option 3 — Local development . . . 633
35.3 Reading the dashboard . . . 634
35.3.1 The top bar . . . 634
35.3.2 Time . . . 634
35.3.3 Colours . . . 634

35.3.4 Rates and their windows . . . 634
35.3.5 The nine rows . . . 634
35.3.6 Reading engine latency . . . 635
35.3.7 Drawers: one process, participant or book . . . 636
35.3.8 Settings . . . 636
35.4 Configuration reference . . . 636
35.5 Security notes . . . 637
35.6 Troubleshooting . . . 637
35.7 Where to go next . . . 638

36 FAQ: Setup and Operation **641**
36.1 Starting the exchange . . . 641
36.1.1 Why did the seed MM quote disappear after the second day? . . . 641
36.1.2 Why did `pm-scheduler` exit immediately? . . . 641
36.2 Seeing what is happening . . . 641
36.2.1 How do I see what messages are flowing between processes? . . . 641
36.3 Participants . . . 642
36.3.1 Why is my order rejected with "Gateway not authenticated"? . . . 642
36.4 Configuration and data files . . . 642
36.4.1 The engine says it cannot find its configuration. Where should the file be? . . . 642
36.4.2 Where are my data files stored? . . . 642

CHAPTER 1

How to Use This Book

The Operator's Guide is about **running an exchange**: installing it, configuring it, starting and supervising its processes, running a trading day, operating the gateways, and finding out afterwards what happened — including recovering from a crash.

Who it is for	Instructors running EduMatcher for a class or a demonstration, and anyone operating an EduMatcher exchange
What it assumes	You have finished the Quick Start Guide: you have run the exchange, made a trade and deployed a configuration
What it leaves to other books	Trading as a participant (Participant Guide); every configuration field and command option (Reference Manual); protocol formats (Protocols and Clients); source-level design (Architecture and Developer Guide)

1.1 How the book is organized

Part	What you learn
I. Install and deploy	Every installation mode, the container networking, every directory the system uses, and the VM runtime image
II. Configure	Writing, verifying, deploying and inspecting a configuration, the form-based Configuration GUI, and ready-made examples
III. Run	Starting, monitoring, restarting and shutting down the process set, and the operator console
IV. Run a market	Listing symbols, valuation, sessions and auctions, risk controls, statistics, the market index, and closing the day (settlement and statements)
V. Gateways	Starting, configuring and troubleshooting each external gateway
VI. Observe and recover	Persistence, the audit trail and replay, and the central log server
Appendix	An FAQ on seed quotes, the scheduler, the participant allowlist, and where the configuration and data files live

1.2 A first class session, chapter by chapter

If you are preparing to run EduMatcher for a class, this route covers what you need:

1. Installation — pick the mode that suits your room; the container mode is the usual choice.
2. The Configuration Workflow and Example Engine Configs — choose or build the market your students will trade.
3. Running the Exchange — start it, check it, and know what to do when a process fails.
4. Session Scheduling and Auctions and Risk Controls — run the trading day and keep it orderly.
5. The Admin Console and Exchange Commands — halt a symbol, cancel orders, disconnect a participant.
6. Statistics and Reporting and the Log Operator Console — show the class what happened.

The Training Guide, chapters 00–19, is the companion set of exercises for the same material.

CHAPTER 2

Installation

Learning objectives

After reading this page you will understand:

- Which of the five installation modes fits what you are trying to do
- How the container deployment is wired, and why the web GUIs reach the exchange without any published port being involved
- What `127.0.0.1` and `0.0.0.0` mean in each of the three separate places they appear, which of them actually protects you, and how the core plane and the service layer come by their different defaults
- Every flag that controls a build or a start, and every directory the system reads or writes

New to EduMatcher? The Quick Start's Install and Start is the short version of this chapter: one route, step by step. This chapter is the complete reference. How a release is produced is described in the Architecture and Developer Guide's The Release Process.

2.1 Choosing an installation mode

The first question is whether you want to **run** EduMatcher or **change** it.

Mode	Best for	Needs on your machine	What you get	Command style
Containers, one command	Running a venue: classroom, demo, self-study	Podman or Docker	The exchange **and** all six web GUIs	`./edumatcher.sh start`
Containers from source	Changing the code and seeing it run as a system	Podman/Docker, Poetry, Node	The same, built from your checkout	`make up-all`
VM bootstrap	Workshops where host setup must stay untouched	Multipass, `curl`	Backend inside a Multipass VM	`multipass shell ems`
pipx	Students running processes by hand, one per terminal	Python 3.14, `pipx`	`pm-*` commands on your PATH	`pm-engine`
pipx + GUI containers	Running the exchange natively but still wanting the browser applications	Python 3.14, `pipx`, Podman or Docker	The pipx exchange, plus the six web GUIs as ready-built containers	`./edumatcher.sh start`
Poetry checkout	Developing, running the test suite	Python 3.14, Poetry	Repository plus dev dependencies	`poetry run pm-engine`

The container modes give you the whole system — engine, gateways, REST API and the six browser applications — with one command. The pipx and Poetry modes give you the individual processes to start yourself, which is what the rest of this guide assumes when it shows a bare `pm-*` command. In Poetry mode, prefix those commands with `poetry run`. The browser applications are not part of the Python package; Web applications with a pipx exchange shows how to add them to a pipx install as containers.

If you are not sure

Use the one-command container install. It is the fastest route to a running market, and nothing it installs is outside a single directory you can delete.

2.2 Containers: the whole system in one command

Everything this install creates lives in **one directory**, `~/.edumatcher` by default. Nothing is written anywhere else — no system paths, no service registered, no change to your PATH — so removing that directory removes the installation.

Inside it	Holds
`~/.edumatcher/data`	**Every trade, order book, log and database the exchange produces.** Mounted into the containers, so it is on your disk rather than inside one, and survives stop, start and update
`~/.edumatcher/config`	An engine configuration of your own, when you supply one
`~/.edumatcher/compose.yaml`, `.env`, `edumatcher.sh`	The deployment itself

Pick a different location with `--dir`; the layout is the same underneath it. Two Compose volumes sit outside that directory, in your container engine's storage — the Log Operator Console's acknowledgements and TapeDeck's failover log. Neither holds exchange data, and `./edumatcher.sh uninstall` removes both.

/.edumatcher/data is the directory to back up, and the one to delete

`./edumatcher.sh uninstall` keeps it; `uninstall --data` is what erases a venue's history. `./edumatcher.sh mounts` shows which directory is behind each path inside each container if you are ever unsure.
With that established:

```
curl -fsSL https://raw.githubusercontent.com/johan162/EduMatcher/main/deployment/curl/install.sh | bash
```

Then open http://localhost:8090.

That starts a complete exchange plus six web applications, from images built and published by the project. Nothing is compiled on your machine: no Python, no Node, no checkout. The only requirement is Podman or Docker.

Application	URL	What it is
TapeDeck market display (`pm-terminal`)	http://localhost:8090	Live order books, trades and market data
Log Operator Console (`pm-log-ui`)	http://localhost:8091	The centralized log, searchable, with acknowledgements
Configuration GUI (`config-gui`)	http://localhost:8092	Author an `engine_config.yaml` in the browser
Trading GUI (`pm-trading-ui`)	http://localhost:8093	Submit and manage orders as a trading participant
Order Book Viewer (`pm-book`)	http://localhost:8094	One symbol's full order book, session statistics and trade tape — the browser companion to `pm-viewer`

Application	URL	What it is
Operator Dashboard (`pm-dash`)	http://localhost:8095	The health of every process, trading activity, participants, books and gateways, live and over time — see Operator Dashboard
REST API docs	http://localhost:8080/docs	Swagger UI for the `desk` API gateway

2.2.1 What the installer does

1. Checks that Podman or Docker is present, preferring Podman when both are.
2. Resolves the newest release (or the one you named with `--version`).
3. Downloads `compose.yaml`, `compose.zmq.yaml`, `edumatcher.sh` and `.env.example` **from that release's tag**, so the files and the images always come from one commit.
4. Writes `.env`, creating `data/` and `config/` beside it.
5. Pulls the six images and starts them.

Step 3 is worth noting: the support files come from the release tag being installed, not from a branch, so the compose file and the images can never describe different systems.

2.2.2 Installer options

Option	Default	Effect
`--version X.Y.Z`	newest release	Install one specific release. All six images carry this tag, so it pins the whole system
`--config NAME`	`s10-basic`	Deploy a bundled example configuration
`--config FILE`	—	…or a path to an `engine_config.yaml` of your own
`--dir PATH`	`~/.edumatcher`	Where to install
`--sim`	—	Install the AI-trader simulation instead, into `~/.edumatcher-sim`
`--guis`	—	Install only the web GUIs, for an exchange run with pipx, into `~/.edumatcher-guis` (details)
`--no-start`	—	Fetch, configure and download the images, but do not start anything. `./edumatcher.sh start` then needs no network
`--help`	—	Show the options

Because the script is read from a pipe, options need `bash -s --` so that the shell hands them to the script rather than consuming them itself:

```
curl -fsSL .../install.sh | bash -s -- --config s10-nominal --version 0.76.5
```

Two environment variables are also honoured: `REPO_OWNER` (which GitHub account and GHCR namespace to use) and `REPO_REF` (fetch the support files from a branch instead of the release tag, for testing an unreleased installer).

2.2.3 Configuring it: the `.env` file

`~/.edumatcher/.env` is where every choice about a running install lives. The installer writes it from `.env.example` on the first run and **keeps your copy afterwards**, so a later `./edumatcher.sh update` re-pins the version without discarding anything else you changed.

Compose reads that file automatically because it sits beside `compose.yaml`. Nothing else does: there is no separate configuration for the containers, and the compose file itself contains no settings, only `${VARIABLE}` references into this file.

Variable	Default	What it does
`EM_VERSION`	*(the installed release)*	Which release to run. All six images carry this tag, so one value pins the whole system. `latest` follows the newest release
`GHCR_OWNER`	`johan162`	The GHCR namespace the images are pulled from. Change it only for a fork
`EM_CONFIG`	`s10-basic`	Which bundled example configuration the exchange deploys
`EM_CONFIG_FILE`	*(empty)*	Set to `/config/engine_config.yaml` when you run a configuration of your own. Non-empty wins over `EM_CONFIG`
`EM_PROFILE`	`mm-demo`	Which processes start: `mm-demo` (the full stack plus a slow, passive market-maker bot quoting as `MM01`), `ai-swarm` (`mm-demo` with its market maker quoting around the market model's values, plus the model and twenty AI traders), `default`, `mini` or `micro`. See Processes, Environment and Ports
`TZ`	`Europe/Stockholm`	Container timezone. Set it to match the trading calendar in your configuration
`BIND_ADDR`	`127.0.0.1`	Which host interface the published ports listen on. See the warning below
`EDUMATCHER_GATEWAY_BIND_HOST`	`0.0.0.0`	Bind host for the service-layer listeners *inside* the container — the four protocol gateways, `pm-log-srv`, `pm-api-gwy` and `pm-monitor`. It is what makes them reachable from the GUI containers, and it wins over any `bind_address:` in the deployed configuration. Not a host-exposure setting; that is `BIND_ADDR`
`EM_ZMQ`	`0`	`1` also publishes the raw ZeroMQ bus (5555-5559, 5601/5602, and `pm-monitor`'s 5611) and tells the engine and `pm-index` to bind the container interface, so tools on your machine can attach. The equivalent of `make up ZMQ=1`
`TERMINAL_GUI_PORT`	`8090`	Host port for TapeDeck (`pm-terminal`)
`LOG_GUI_PORT`	`8091`	Host port for the Log Operator Console (`pm-log-ui`)
`CONFIG_GUI_PORT`	`8092`	Host port for the configuration builder
`TRADER_GUI_PORT`	`8093`	Host port for the trader GUI
`BOOK_GUI_PORT`	`8094`	Host port for the order book viewer
`DASH_GUI_PORT`	`8095`	Host port for the operator dashboard (`pm-dash`)

Variable	Default	What it does
`LOG_GUI_URL` / `BOOK_GUI_URL`	`http://localhost:<port>`	Where the dashboard's drawers link to the log console and the order book viewer, as your browser reaches them — set when you open the GUIs from another machine

Two settings have their own commands, because editing them by hand is easy to get half-right — `EM_CONFIG` and `EM_CONFIG_FILE` must agree, and `EM_VERSION` needs an image pull to take effect:

```
./edumatcher.sh config s10-nominal    # sets EM_CONFIG, clears EM_CONFIG_FILE
./edumatcher.sh config ./mine.yaml    # copies the file, sets EM_CONFIG_FILE
./edumatcher.sh update 0.26.2         # sets EM_VERSION, pulls, restarts
```

Everything else is a plain edit followed by `./edumatcher.sh restart`.

BIND_ADDR=0.0.0.0 puts an unauthenticated exchange on your network

It is the right setting for a classroom where students connect to the instructor's machine, and the wrong one on a network you do not control: the protocol gateways have no password. The default keeps everything on this machine.

Values you will not find in `.env`

TapeDeck needs `API_GATEWAY_URL` and `PM_TERMINAL_API_KEY`, and the order book viewer `API_GATEWAY_URL` and `PM_BOOK_API_KEY`, and the operator dashboard `API_GATEWAY_URL` and `PM_DASH_API_KEY` — the same read-only key. Those are **resolved at startup and injected**, not stored: the read-only API key is generated per engine configuration — a different one in each bundled example — and lives on a different gateway instance than the trading credentials. `./edumatcher.sh start` reads the deployed configuration and passes them to all three applications. Setting them in `.env` would only go stale the next time you switched configuration.

2.2.4 Everyday commands

Everything is driven by one script in the install directory:

```
cd ~/.edumatcher
./edumatcher.sh status              # containers, plus the exchange process table
./edumatcher.sh logs terminal-gui   # follow one service
./edumatcher.sh shell               # a shell inside the exchange container
./edumatcher.sh shell pm-config-show      # ...or one command in it
./edumatcher.sh urls                # the application table, with your ports
./edumatcher.sh mounts              # which directory is behind each container path
./edumatcher.sh stop                # stop everything; ./data is kept
./edumatcher.sh start               # bring it back
./edumatcher.sh upgrade             # newest release: asks first if running, replaces the images
./edumatcher.sh update              # pull the newest release and restart, without asking
./edumatcher.sh update 0.20.5       # ...or move to an exact one
./edumatcher.sh uninstall           # remove containers and volumes, keep data
./edumatcher.sh uninstall --data    # remove everything
```

Picking up a new web application

`update` pulls new images for the files you already have; it does not re-download `compose.yaml`. When a release adds an application — the order book viewer on port 8094 and the operator dashboard on 8095 arrived this way — re-run the one-line installer instead. It keeps your `.env` and `data/`, and fetches the release's `compose.yaml` and `edumatcher.sh`.

2.2.5 Choosing what the exchange trades

EduMatcher ships 30 ready-made configurations: 1, 3, 10, 30 or 150 order books, each as a `basic`, `nominal` or `complex` variant, and each of those with or without seeded market-maker quotes. They are inside the backend image, so no download is involved in switching.

```
./edumatcher.sh config s10-nominal
./edumatcher.sh restart
```

The names are `s1`, `s3`, `s10`, `s30` and `s150` combined with `basic`, `nominal` and `complex`, each with an optional `-nomm` suffix for the variant with empty order books instead of seeded market-maker quotes (for example `s3-basic-nomm`). The `s150-*` configs also seed `AI001-AI020` for a 20-bot `pm-ai-swarm` run unchanged. `./edumatcher.sh config` lists them all if you mistype one. See Example Engine Configs for what each contains.

To run a configuration of your own, give the same command a path instead:

```
./edumatcher.sh config ./my-market.yaml
./edumatcher.sh restart
```

The file is copied into `~/.edumatcher/config/`, mounted read-only into the container, and deployed on every start — so editing it and restarting is the whole edit-test loop. Switching back to a bundled example is `./edumatcher.sh config s10-basic`. The configuration builder at http://localhost:8092 is the easy way to author one; see Configuration GUI.

2.3 Containers from the source checkout

Use this when you are changing EduMatcher and want to see your change running as a whole system. It builds the same six images from your working tree.

```
git clone https://github.com/johan162/EduMatcher.git
cd EduMatcher/deployment/docker
make build          # builds a wheel from the checkout, then the image
make up-all         # exchange + terminal-, log-, trader-, book- and dash-gui
```

`make help` lists every target. The two that matter are `up` (the exchange alone) and `up-all` (the exchange plus the GUIs).

2.3.1 Flags that control the build

`make build` decides **where the EduMatcher package comes from**:

Flag	Result
(none)	Build a wheel from this checkout with Poetry and install that. This is the default deliberately: a rebuild after editing source must pick up the edit
`PYPI=1`	Ignore the checkout; install the newest release from PyPI
`VERSION=x.y.z`	Install exactly that PyPI release (implies `PYPI=1`)
`DEV=1`	Accepted and ignored — it used to select the local wheel, which is now the default
`CONTAINER_ENGINE=docker`	Force Docker when Podman is also installed

Confirm which package went in

A successful build prints a line of the form `Installing local wheel: /tmp/wheel/edumatcher-<version>-<build-tag>.whl` (the exact filename comes from whatever Poetry built, so it may include a platform or Python-tag suffix beyond the version). If instead you see pip *downloading* `edumatcher`, the image is running a published release rather than your code — and a source change you are hunting for will appear not to take effect.

2.3.2 Flags that control a start

Flag	Default	Effect
`CONFIG=<name>`	`s10-basic`	Deploy a bundled example

Flag	Default	Effect
`CONFIG=<file>`	—	Deploy an `engine_config.yaml` of your own; the file is copied to `deployment/docker/config/` and mounted read-only
`PROFILE=<name>`	`default`	Which processes `pm-opctl-cli` starts: `default`, `mini`, `micro`, `mm-demo` or `ai-swarm`. See Processes, Environment and Ports
`ZMQ=1`	off	Also publish the raw ZeroMQ bus (5555-5559, 5601-5602) to the host, and set the engine and index sockets to bind `0.0.0.0` inside the container
`SSH=1`	off	Run `sshd` in the container on `SSH_PORT`, authorised by your `~/.ssh/*.pub`
`CONFIG_GUI=1`	off	Include the configuration builder in `up-all`. It is opt-in because it talks to nothing — it is a standalone authoring tool

Flags combine: `make up-all CONFIG=s10-complex PROFILE=mini ZMQ=1 CONFIG_GUI=1`.

2.3.3 Settings in `.env`

`deployment/docker/.env` is created from `.env.example` on the first build. Compose reads it automatically because it sits beside the compose files, and it is git-ignored — so it is the right place for choices specific to your machine, as opposed to a `make` flag, which applies to one invocation only.

When the same setting is available in more than one place, the nearer one wins:

```
make flag  →  shell environment  →  .env  →  compose ${VAR:-default}  →  image ENV
```

That is why `make up-all CONFIG=s10-nominal` does not edit `.env`, and why a plain `make up-all` afterwards goes back to whatever `.env` says.

Variable	Default	Meaning
`COMPOSE_PROJECT_NAME`	`edumatcher`	Compose project name; decides container and network naming
`EDUMATCHER_VERSION`	*(empty)*	PyPI version for `PYPI=1` builds; empty means newest
`WITH_SSH`	`1`	Install `openssh-server` into the image at build time
`EM_CONFIG`	`s10-basic`	Bundled example to deploy
`EM_CONFIG_FILE`	*(empty)*	Path **inside the container** to a configuration of your own; set for you by `CONFIG=<file>`
`EM_PROFILE`	`default`	Process profile to start
`TZ`	`Europe/Stockholm`	Container timezone — match the trading calendar in your configuration
`BIND_ADDR`	`127.0.0.1`	Which host interface the published ports listen on. See below

Variable	Default	Meaning
EDUMATCHER_GATEWAY_BIND_HOST	0.0.0.0	Bind host for the service-layer listeners *inside* the container — the four protocol gateways, pm-log-srv, pm-api-gwy and pm-monitor. It is what makes them reachable from the GUI containers, and it wins over any bind_address: in the deployed configuration. Not a host-exposure setting; that is BIND_ADDR
SSH_PORT	2222	Host port forwarded to sshd
TERMINAL_GUI_PORT	8090	Host port for TapeDeck (pm-terminal)
LOG_GUI_PORT	8091	Host port for the Log Operator Console (pm-log-ui)
CONFIG_GUI_PORT	8092	Host port for the configuration builder
TRADER_GUI_PORT	8093	Host port for the trader GUI
BOOK_GUI_PORT	8094	Host port for the order book viewer
DASH_GUI_PORT	8095	Host port for the operator dashboard (pm-dash)
LOG_GUI_URL / BOOK_GUI_URL	http://localhost:<port>	Where the dashboard's drawers link to the log console and the order book viewer, as your browser reaches them — set when you open the GUIs from another machine
IMAGE_NAME / IMAGE_TAG	edumatcher / local	Image naming
CONTAINER_NAME	edumatcher	Container name

Beyond these, the compose files read a few variables that have sensible defaults and no .env entry: CORS_ORIGIN, MAX_WS_CLIENTS, CALF_CLIENT_ID and INDEX_IDS for TapeDeck, LOG_SRV_ENABLED for its logging uplink, and PIP_INDEX_URL as a build argument. Export any of them in your shell before make up-all if you need to.

API_GATEWAY_URL and PM_TERMINAL_API_KEY are read the same way but should be left alone: up-all resolves them from the deployed configuration and injects them, for the reasons in The read-only API key.

2.3.4 Image build arguments

Passed with --build-arg, or through the compose files' args: blocks. You need these mainly behind a corporate proxy.

The backend image (deployment/docker/Dockerfile):

Argument	Default	Purpose
PYTHON_VERSION	3.14	Base image tag for both build and runtime stages
EDUMATCHER_VERSION	*(empty)*	PyPI version to install when no local wheel is present
PIP_INDEX_URL	*(empty)*	Point pip at an internal index such as Artifactory
WITH_SSH	1	Set to 0 for a smaller image with no openssh-server

Each web GUI image (web-apps/*/Dockerfile):

Argument	Default	Purpose
`WEB_PORT`	per app	Port the server listens on inside the container
`NPM_REGISTRY`	`https://registry.npmjs.org/`	Internal npm mirror. The Dockerfiles accept this argument, but the compose files' `args:` blocks do not currently pass it through, so setting it in your shell has no effect via `make up-all`/`make build-guis` — pass `--build-arg NPM_REGISTRY=...` directly to `podman build`/`docker build` instead
`NPM_STRICT_SSL`	`true`	Set `false` when a proxy re-signs TLS
`HTTP_PROXY` / `HTTPS_PROXY` / `NO_PROXY`	*(empty)*	Honoured by npm during the build
`USE_PROXY_CA` / `CA_CERT_FILE`	*(empty)*	Declared only by the unused `web-apps/config-gui/Dockerfile.proxy`, not the `Dockerfile` that `compose.config-gui.yaml` actually builds — setting these currently has no effect on `make up-all CONFIG_GUI=1`
`EDUMATCHER_VERSION`	*(empty)*	`log-gui` only: the PyPI release of the bundled `pm-log-cli` when `web-apps/log-gui/.wheel/` holds no wheel. `make build-guis` fills that drop box from your checkout by the same rule as `make build`, and empties it for `PYPI=1` or `VERSION=x.y.z`

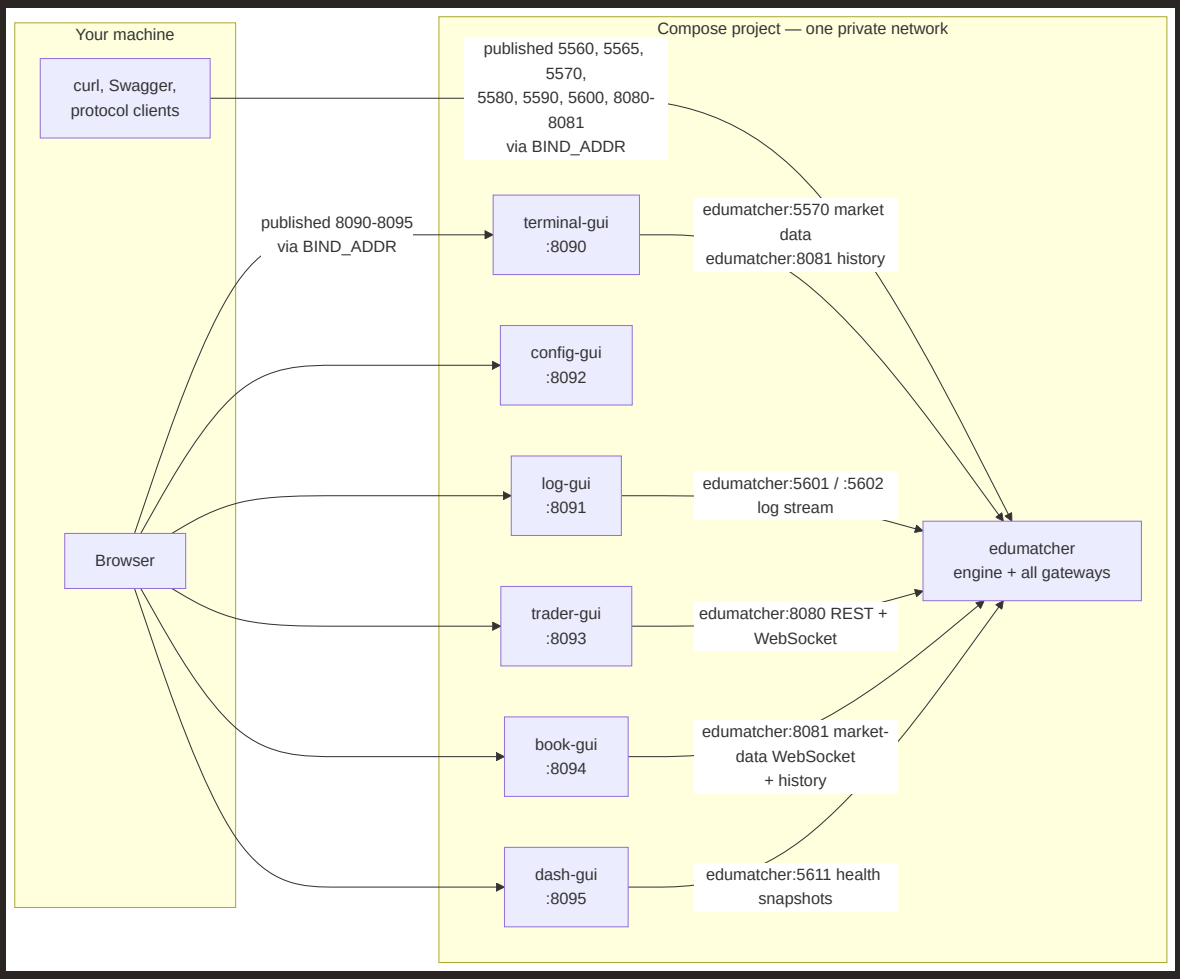
2.4 How the containers are wired

For a step-by-step walkthrough of the images, compose files, pods, volumes and ports behind this section, with diagrams and the commands to rebuild everything by hand, see Inside the Container Deployment.

This section is worth reading even if the install "just worked", because the one thing that reliably confuses people here — the difference between `127.0.0.1` and `0.0.0.0` — appears in three separate places that mean three different things, and only one of them is a security boundary.

2.4.1 One compose project, one private network

All six containers belong to a single Compose project, so Compose puts them on one private network and gives each a name on it. The backend answers to the hostname `edumatcher`.



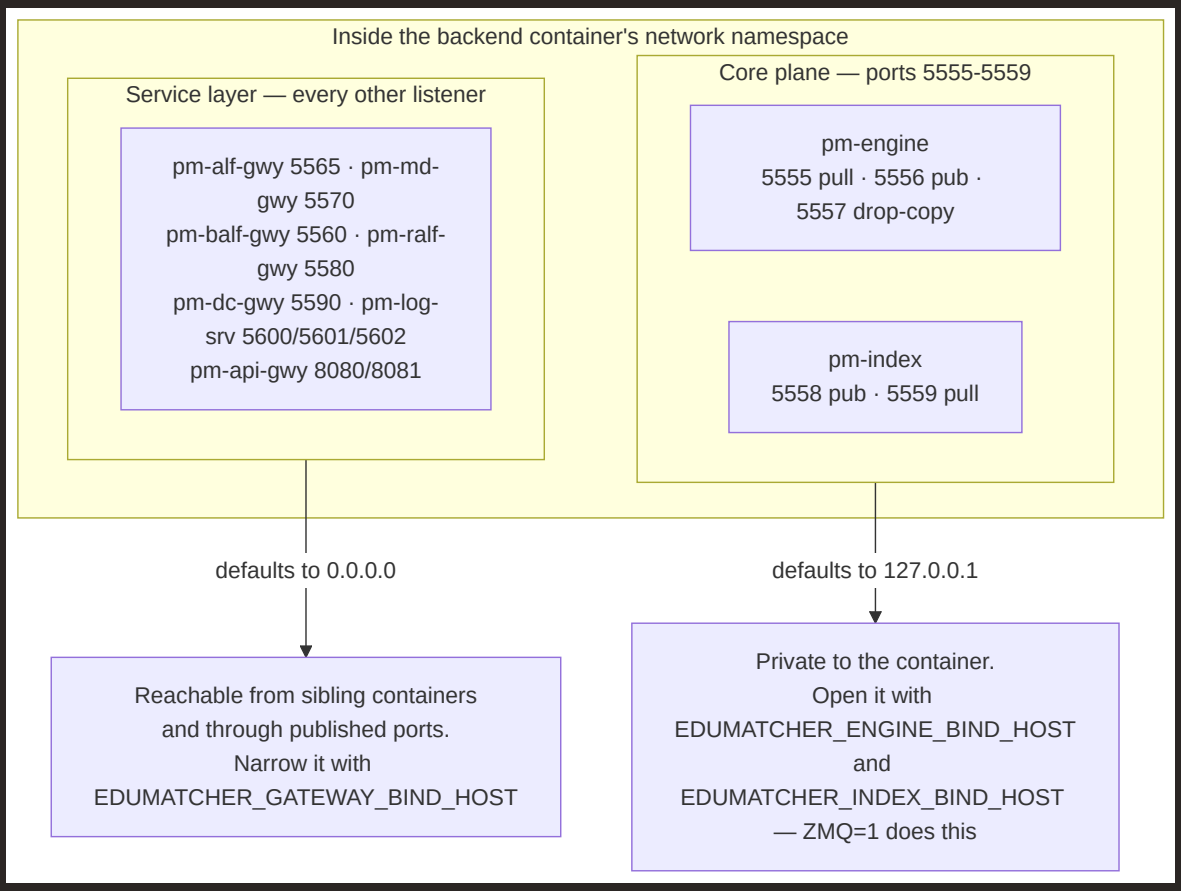
Two things follow, and both are easy to get wrong:

- **The GUIs do not use the published ports.** Their traffic never leaves the private network. The ports published on your machine exist for *you* — your browser, `curl`, Swagger, the protocol example clients.
- **Nothing needs `host.docker.internal`.** Older setups pointed the GUIs at the host and back in again, which behaved differently on Docker Desktop, Linux Docker and Podman. Sharing one project removes the problem rather than working around it.

The configuration builder is the exception: it talks to nothing at all, which is why it is opt-in with `CONFIG_GUI=1` when building from source.

2.4.2 The two planes, and where each address is decided

Every listening socket in EduMatcher belongs to one of two planes, and the two have deliberately different defaults.



The core plane is the internal bus. `pm-engine` and `pm-index` speak ZeroMQ to the other EduMatcher processes and to nothing else, so they default to `127.0.0.1` and a stock install keeps the bus inside the container. Starting with `ZMQ=1` sets `EDUMATCHER_ENGINE_BIND_HOST` and `EDUMATCHER_INDEX_BIND_HOST` to `0.0.0.0` and publishes 5555–5559, so you can attach your own ZeroMQ client from the host.

The service layer is what clients talk to. All seven gateway processes default to `0.0.0.0`, because every practical deployment needs them reachable from somewhere else — a sibling GUI container, a student's laptop in a classroom, a protocol example running on your host.

Each service listener resolves its bind host in this order, first match wins:

Precedence	Source	Example
1	The process's own command-line flag — `--bind` on the four protocol gateways, `--host` on `pm-log-srv` and `pm-api-gwy`	`pm-md-gwy --bind 127.0.0.1`
2	The `EDUMATCHER_GATEWAY_BIND_HOST` environment variable	`EDUMATCHER_GATEWAY_BIND_HOST=127.0.0.1`
3	`bind_address:` (or `host:` for `pm-api-gwy`) in the engine configuration	`bind_address: 10.0.0.5`
4	The built-in default	`0.0.0.0`

`EDUMATCHER_GATEWAY_BIND_HOST` deliberately sits *above* the configuration file. It is a deployment-wide switch: one environment variable closes every service listener at once, without editing a configuration that may not be yours to edit.

2.4.3 What `0.0.0.0` does and does not expose

Where	Set by	`127.0.0.1` means	`0.0.0.0` means
Published ports — the host side	`BIND_ADDR` in `.env`	Only this machine can reach the exchange	Anyone on your network can
Service listeners — inside the container	default, or `EDUMATCHER_GATEWAY_BIND_HOST`, or `bind_address:` / `host:`	Only processes *in that container* — not even a sibling container	Any container on the Compose network, and anything reaching a published port
Core plane — inside the container	default, or `EDUMATCHER_ENGINE_BIND_HOST` / `EDUMATCHER_INDEX_BIND_HOST`; set by `ZMQ=1`	Same: container-internal only	Reachable from the Compose network and publishable

Only the first row is a security boundary. The second and third are inside a network namespace that already isolates the container: binding `0.0.0.0` there exposes nothing to the outside world, because there is no route in except through a port you chose to publish. Whether the exchange is visible on your LAN is decided entirely by `BIND_ADDR`.

BIND_ADDR=0.0.0.0 puts an unauthenticated exchange on your network

It is the right setting for a classroom where students connect to the instructor's machine. It is the wrong setting on a network you do not control. There is no password on the order-entry gateways — anyone who can open the socket can submit orders.

On bare metal there is no namespace

A `pipx` or Poetry install has no container around it, so a service listener on `0.0.0.0` really is on your network. If that is not what you want, export `EDUMATCHER_GATEWAY_BIND_HOST=127.0.0.1` — one variable, every gateway, no configuration edits. This is a change in default behaviour; see the note in `CHANGELOG.md`.

2.4.4 Why there is no longer a loopback rewrite

Earlier versions shipped the bundled examples with `bind_address: 127.0.0.1` and had the container entrypoint rewrite them to `0.0.0.0` on the way in. That rewrite worked, but it meant the deployed configuration was not the configuration you wrote, and `pm-config-show` described one thing while the processes did another.

The gateways now bind `0.0.0.0` themselves, every bundled example says `0.0.0.0`, and the entrypoint rewrites nothing: **the deployed configuration is the configuration you supplied**, and `make config-show` describes what the processes actually do.

2.4.5 Two things that look like evidence and are not

Both of these will tell you a gateway is reachable when it is not:

"I can reach the port from the host." Podman's rootless port forwarding delivers a published connection to *loopback inside the container's namespace*. A listener bound to `127.0.0.1` therefore answers host traffic perfectly, while a sibling container connecting across the private network gets `ECONNREFUSED`. Host reachability says nothing about container-to-container reachability. The stock defaults avoid this, but you can still walk into it by setting `EDUMATCHER_GATEWAY_BIND_HOST=127.0.0.1`, by passing `--host` to a process, or by supplying a configuration that pins `bind_address:` to loopback.

"`make status` is all green." `pm-opctl-cli list` shows `tcp connect to 127.0.0.1:5570 ok`, but that check also runs inside the container, over the same loopback. A healthy process table is not evidence that anything outside that container can connect.

The check that settles it is the compiled configuration, which is the file the processes actually read:

```
podman exec edumatcher python3 -c \
  "import json; print(json.load(open('/data/ref_data/engine_config.json'))['market_data_gateway'])"
```

and, from a GUI container, an actual connection attempt:

```
podman exec edumatcher-terminal-gui node -e '
const net = require("net");
const s = net.connect(5570, "edumatcher", () => { console.log("OPEN"); s.end(); });
s.on("error", e => console.log("FAIL", e.code));'
```

2.4.6 The read-only API key

One value cannot be a fixed default. TapeDeck reads historical data, and the order book viewer reads everything, through `pm-api-gwy` using the credential whose `gateway_id` is `null`. That key is **generated per engine configuration** — every bundled example has a different one — and it is issued on the `dashboards` gateway instance (port 8081), not the `desk` instance (8080) that carries the trading credentials.

Both start paths therefore run in two phases: bring up the exchange, read the key out of the deployed configuration, then start the GUIs with it. If a configuration has no such credential the start still succeeds and says so — the TapeDeck's live market-data feed needs no key, only its history panels do; the order book viewer, which reads all its data through `pm-api-gwy`, then has nothing to show.

2.5 Directories and paths

Every location the system reads or writes, and what controls it.

2.5.1 The data directory

One variable decides where an exchange keeps its state:

Path	Controlled by	Contains
`<DATA_DIR>`	`EDUMATCHER_DATA_DIR`; see Environment variables in the Reference Manual for how the default is chosen	Everything below
`<DATA_DIR>/ref_data/engine_config.json`	`pm-setup`, `pm-config-deploy`	The **compiled artifact every process reads**
`<DATA_DIR>/ref_data/engine_config.yaml`	same	The authored source it was compiled from, kept for provenance
`<DATA_DIR>/emo/`	`pm-opctl-cli`	One log per process, PID files, and the active profile name
`<DATA_DIR>/log.db`	`pm-log-srv`	The centralized log
`<DATA_DIR>/stats.db`	`pm-stats`	Statistics and history behind the REST API
`<DATA_DIR>/clearing.db`	`pm-clearing`	Positions and P&L
`<DATA_DIR>/eod.db`, `settlement/`, `backup/`	`pm-eod`	The settlement ledger, the per-day trader statements, and compressed database backups — see Closing the Day
`<DATA_DIR>/ref_data/roster.yaml`	you (optional)	Which person is behind which gateway, for the statements
`<DATA_DIR>/audit.log`, `audit_index.db`	`pm-audit`	The audit trail and its index
`<DATA_DIR>/gtc_orders.json`, `gtc_combos.json`, `book_stats.json`	`pm-engine`	Resting orders and book state across restarts
`<DATA_DIR>/logs/`	log clients	Failover logs written when `pm-log-srv` is unreachable

Configured relative paths such as `data/stats.db` resolve under `<DATA_DIR>`, so they mean the same file no matter which directory a process was started from. Absolute paths in the configuration remain explicit overrides.

2.5.2 Where that lands in each mode

Mode	<DATA_DIR> is	Controlled by
One-command containers	~/.edumatcher/data on your disk, mounted at /data in the container	--dir at install time
Containers from source	deployment/docker/data, mounted at /data	The ./data:/data mount in compose.yaml
pipx	~/.local/share/edumatcher	EDUMATCHER_DATA_DIR
Poetry checkout	<repo>/src/data/	EDUMATCHER_DATA_DIR
VM	/home/ubuntu/.local/share/edumatcher inside the VM (same installed default as pipx)	mknode.sh, via EDUMATCHER_DATA_DIR in .bashrc

The container images set EDUMATCHER_DATA_DIR=/data internally, so the container's view and your directory are the same files. Your data is on your disk, not inside a container: it survives stop, start, rebuild and update.

2.5.3 Supporting directories

Path	Created by	Purpose
~/.edumatcher/	the installer (--dir to change)	compose.yaml, .env, edumatcher.sh, data/, config/
~/.edumatcher/config/	./edumatcher.sh config <file>	Your own configuration, mounted read-only at /config
deployment/docker/.env	first make build or make up	Machine-specific settings; git-ignored
deployment/docker/config/	make up-all CONFIG=<file>	As above, for the source-built stack
deployment/docker/.wheel/	make build	The locally built wheel the image installs from
deployment/docker/.ssh/	make keys	authorized_keys assembled from your ~/.ssh/*.pub, for SSH=1
/backend-data (in log-gui and dash-gui)	the ./data:/backend-data:ro mount	Where the Log Operator Console reads log.db, and the operator dashboard metrics.db and the deployed configuration — read-only by construction, so neither can write the exchange's files
log-gui-acks, terminal-gui-logs, book-gui-logs, dash-gui-logs	Compose named volumes	Log acknowledgements; the terminal's, the order book viewer's and the dashboard's failover logs

2.6 VM bootstrap — a ready-to-run Multipass VM

Use this when you want the fewest assumptions about your host machine. Your host needs Multipass and curl; Python and EduMatcher are installed inside the VM. This mode installs the backend only — the web GUIs are part of the container deployment. Provisioning runs pm-setup, so the VM comes up with a deployed configuration and pm-opctl-cli ready to start the stack.

```
curl -fsSL https://raw.githubusercontent.com/johan162/EduMatcher/main/deployment/vm/curl_setup_vm.sh | \
    bash -s -- --version 0.76.5 --snapshot

multipass shell ems
cd /home/ubuntu/session
pm-opctl-cli start
```

Option	Default	Purpose
`--name <vm>`	`ems`	Name the VM
`--version X.Y.Z`	`dev`	Install a specific EduMatcher release. The default installs a local wheel, so pass this unless you have a checkout
`--dev`	—	Install the wheel from the repository's `dist/` instead of PyPI
`--cpus` / `--memory` / `--disk`	`4` / `4G` / `6G`	Size the VM
`--image <name>`	`lts`	Base Multipass image
`--snapshot-name <name>`	`clean`	Name of the snapshot taken after provisioning
`--snapshot`	already on	Snapshots are taken by default; the flag only makes that explicit
`--ssh-key <path>`	`~/.ssh/<vm>_ed25519`	Private key whose public half is installed for passwordless login

To read the script before running it:

```
curl -fsSL https://raw.githubusercontent.com/johan162/EduMatcher/main/deployment/vm/curl_setup_vm.sh -o curl_setup_vm.sh
less curl_setup_vm.sh
bash curl_setup_vm.sh --version 0.76.5 --snapshot
```

2.7 pipx — commands on your own machine

Use this when you want to run the processes yourself, one per terminal, which is how most of this guide is written.

Requirement	Notes
Python 3.14 or later	Check with `python --version`
`pipx`	Installs command-line applications into isolated environments
Several terminals	Or `tmux` / `screen`; one process per pane is normal

```
# macOS with Homebrew
brew install pipx
pipx ensurepath

# Linux / generic Python install
python -m pip install --user pipx
python -m pipx ensurepath
```

Then install and bootstrap a session directory:

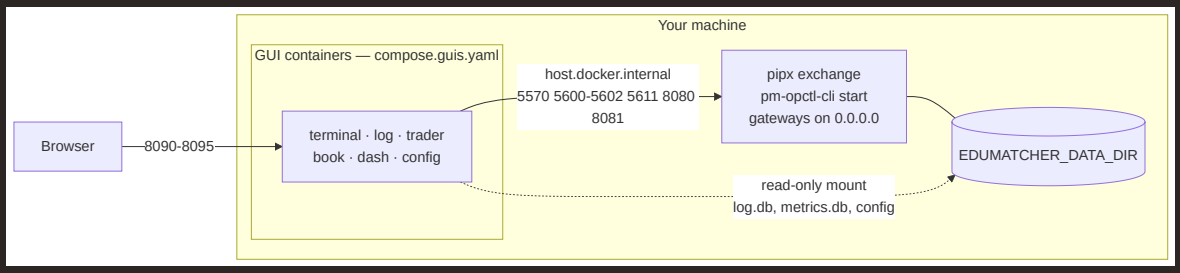
```
pipx install edumatcher
mkdir edumatcher-session
cd edumatcher-session
pm-setup
```

`pm-setup` creates the data directory, deploys a bundled example configuration and prints the `EDUMATCHER_DATA_DIR` line to add to your shell profile. Open a new terminal afterwards so every `pm-*` command sees the same data directory.

```
pm-setup --config s10-nominal   # a different bundled example
pm-setup --data-dir ~/my-venue  # an explicit location
pm-setup --force                # replace an already-deployed configuration
pm-setup --no-config            # create the directory only
```

2.8 Web applications with a pipx exchange

The browser applications are separate projects and are not in the Python package. If you run the exchange with `pipx` you can still have all six of them: they are published as ready-built container images, and a small installer runs just those, pointed at the exchange on your own machine. It works the same for a plain exchange and for the AI-trader simulation.



You need Podman or Docker with Compose in addition to the pipx requirements above. Nothing else is installed: no Node, no checkout.

2.8.1 Step 1: the exchange

Set it up as in pipx, then start it as you normally would, with one addition: `EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0`. The protocol gateways and the REST API already listen on every interface, but `pm-monitor`, which feeds the Operator Dashboard, listens on `127.0.0.1` unless this variable says otherwise, and a container cannot reach loopback. A plain exchange:

```
pipx install edumatcher
pm-setup                        # deploys s10-basic into ~/.local/share/edumatcher
export EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0
pm-opctl-cli start
```

or the AI-trader simulation (see A Complete Set-up by Hand for what each step does, and `EDUMATCHER_SCHEDULER_SPEED` for an accelerated day):

```
export EDUMATCHER_DATA_DIR=~/edumatcher-sim
export EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0
pm-setup --config s300-sim
pm-opctl-cli start ai-sim
```

Wait until `pm-opctl-cli list` shows every process running before you open the GUIs. The installer in the next step needs the deployed configuration that `pm-setup` creates, but not a running exchange.

2.8.2 Step 2: the GUI containers

```
curl -fsSL https://raw.githubusercontent.com/johan162/EduMatcher/main/deployment/curl/install.sh \
| bash -s -- --guis
cd ~/.edumatcher-guis
```

This downloads `compose.guis.yaml`, `edumatcher.sh` and a `.env` into `~/.edumatcher-guis`, pulls the six images from `ghcr.io` and starts them. The installer reads the exchange's data directory from `EDUMATCHER_DATA_DIR` in the shell you run it from, and from `~/.local/share/edumatcher` when that is not set, so run it in a shell where your `pm-*` commands work. If you set the variable only in this one shell, repeat it for later starts or put `EM_DATA_DIR=<directory>` in `.env`.

`--version X.Y.Z` pins the images to a release. Use the version of your pipx package (`pm-engine --version`): the GUIs and the exchange are released together.

The addresses are the same as in the container install:

URL	Application
http://localhost:8090	TapeDeck (`pm-terminal`)
http://localhost:8091	Log Operator Console (`pm-log-ui`)
http://localhost:8092	Configuration GUI

URL	Application
http://localhost:8093	Trading GUI
http://localhost:8094	Order Book Viewer (`pm-book`)
http://localhost:8095	Operator Dashboard (`pm-dash`)

`./edumatcher.sh` works as in the container install, with these differences: `status` lists the GUI containers and then runs `pm-opctl-cli list` on your own machine; `stop` stops the GUIs and leaves the exchange alone; `shell`, `config` and `speed` are refused, because the exchange is yours and its commands are already on your PATH. `share on` and `update` behave as documented in the curl installer README.

2.8.3 What the GUIs need from the exchange

Three things connect the containers to your pipx exchange. The installer sets two of them up for you; the third is yours to check.

The host address. Inside a container, `127.0.0.1` is the container itself. The GUIs therefore reach the exchange as `host.docker.internal`, which `compose.guis.yaml` maps to your machine with `host-gateway`. Docker Desktop, Docker 20.10 or later and Podman 5 all honour this. If the name clashes with something on your system, set `EM_HOST` in `.env`.

The read-only API key. TapeDeck's history panels, the Order Book Viewer and the dashboard's session schedule use the credential whose `gateway_id` is `null`. It is different in every configuration, so `./edumatcher.sh start` reads it from `ref_data/engine_config.json` in the data directory and hands it to the containers. Run `./edumatcher.sh restart` after you deploy another configuration. The Trading GUI needs no key: you type your own trading credentials into its login screen.

The service listeners must be on `0.0.0.0`. The gateways and the REST API default to it, and `EDUMATCHER_GATEWAY_BIND_HOST=0.0.` (Step 1) extends it to `pm-monitor`. A configuration that pins `bind_address: 127.0.0.1`, or an exported `EDUMATCHER_GATEWAY_BIND_HOST=1` closes them to the containers while every check on your machine still looks healthy. If a GUI shows nothing, check this first; without the variable the symptom is a dashboard that reports the monitor as down while everything else works.

Container	Reaches on your machine	Used for
TapeDeck, Order Book Viewer	5570, 8081, 5600	Market data, history, log forwarding
Trading GUI	8080	Orders and account data
Log Operator Console	5601, 5602, `log.db`	The log stream and its history
Operator Dashboard	5611, `metrics.db`, 8081, 5600	Process health and the session schedule
Configuration GUI	nothing	Standalone editor

The raw ZeroMQ ports 5555-5559 are not used, so no `EM_ZMQ` equivalent is needed.

The gateways have no password

On a pipx install a listener on `0.0.0.0` is on your network, not behind a container namespace. The GUIs' own published ports follow `BIND_ADDR` in `.env` (`127.0.0.1` by default), but your exchange's gateways are open to whatever your firewall lets through. A container can only reach a listener that is not bound to loopback, so there is no loopback-only setup for this combination. On a network you do not control, keep the exchange behind the host firewall.

A container exchange on the same machine

The GUI containers use the host ports 8090-8095, the same as the full container install, so the two cannot run together. If a container exchange or simulation is running and publishing 5570, 8080 and the rest on `127.0.0.1`, these GUIs cannot reach it either: a port published to loopback is not reachable through `host-gateway`. Use that install's own GUIs.
If something does not work, run `./edumatcher.sh mounts` (it shows which directory the GUIs are reading) and test a port from inside a container:

```
podman exec edumatcher-host-terminal-gui node -e '
const net = require("net");
const s = net.connect(5570, "host.docker.internal", () => { console.log("OPEN"); s.end(); });
s.on("error", e => console.log("FAIL", e.code));'
```

`ECONNREFUSED` means nothing listens on a non-loopback address (check the bind host above, and that the process is running; test 5611 for the dashboard). A hang or `EHOSTUNREACH` usually means the host firewall: allow connections from the container network to the exchange's ports (on Fedora and other `firewalld` systems, put the Podman network in a trusted zone).

To remove the GUIs, `./edumatcher.sh uninstall` deletes the containers and volumes; the exchange and its data are not touched.

2.9 Poetry checkout — developing EduMatcher

```
git clone https://github.com/johan162/EduMatcher.git
cd EduMatcher
poetry config virtualenvs.in-project true
poetry install --with dev,docs

poetry run pm-engine --verbose
poetry run pm-alf-console --id TRADER01
```

Developer mode uses the repository-local data directory. When exact behaviour matters, use the same deployed-configuration flow as installed mode: author YAML, run `poetry run pm-config-deploy ...`, then restart the processes.

2.10 Shell completion

Every `pm-*` command completes its subcommands, options and `choices=` values in bash and zsh — `pm-audit-cli ev<Tab>` completes to `events`, `pm-engine --log-level <Tab>` lists `CRITICAL ERROR WARNING INFO DEBUG`, and so on. `pm-help --completion bash|zsh` prints the script; nothing needs to be generated by hand.

Container: nothing to do — the backend image builds completion in at image-build time and every interactive `make shell` / `make ssh` session already has it.

pipx or `pip install` into a venv: add one line to your shell profile —

```
# ~/.bashrc
command -v pm-help >/dev/null && eval "$(pm-help --completion bash)"
```

```
# ~/.zshrc, after compinit
(( $+commands[pm-help] )) && eval "$($commands[pm-help] --completion zsh)"
```

Poetry checkout: the venv is usually not active when your shell profile runs, so point at it by its absolute path —

```
# ~/.bashrc
EM=/path/to/EduMatcher/.venv/bin/pm-help
[ -x "$EM" ] && eval "$("$EM" --completion bash)"
```

```
# ~/.zshrc, after compinit
EM=/path/to/EduMatcher/.venv/bin/pm-help
[[ -x $EM ]] && eval "$($EM --completion zsh)"
```

An editable install (`poetry install`) reads the completion scripts straight from the checkout, so they follow whichever branch is checked out. Calling `pm-help` at every shell start (rather than sourcing a saved copy) keeps completion matched to the installed version, at the cost of `pm-help`'s own ~0.2s start-up per new shell. To avoid that, cache it once per upgrade instead:

```
pm-help --completion bash > ~/.local/share/edumatcher/pm-completion.bash
# ~/.bashrc:  source ~/.local/share/edumatcher/pm-completion.bash
```

```
pm-help --completion zsh > ~/.local/share/edumatcher/pm-completion.zsh
# ~/.zshrc:   eval "$(<~/.local/share/edumatcher/pm-completion.zsh)"
```

zsh's own completion system (`compinit`) must already be initialised — the snippets above assume the usual `autoload -Uz compinit && compinit` earlier in `~/.zshrc`. Loading the file with `source` instead of `eval` silently registers nothing; always use `eval` for the zsh script.

2.11 Where to go next

- Inside the Container Deployment — how the container images, compose files, volumes and ports fit together
- Quick Start Guide — what EduMatcher is, and your first trade
- Engine Configuration — authoring an `engine_config.yaml`
- Configuration GUI — doing it in the browser instead
- Running the Exchange — starting processes and keeping them healthy
- Processes, Environment and Ports — what each `pm-*` process is for

CHAPTER 3

VM Runtime Image with Pinned PyPI Release

Learning objectives

After reading this page you will understand:

- How to build a Multipass VM preinstalled with a chosen EduMatcher release
- Why the image uses pip in a dedicated venv rather than pipx
- What provisioning does, and how every `pm-*` command reaches `/usr/local/bin`
- How to operate, snapshot, upgrade and reset the VM
- When to reach for the VM instead of the container deployment

3.1 Summary

`deployment/vm/` builds a Multipass VM carrying one chosen EduMatcher release, with every `pm-*` command on the system PATH and a configuration already deployed. It is the option to pick when you want a *machine* to log into — a workshop image you can snapshot and reset between classes, or an environment where processes are started by hand.

Script	Role
`deployment/vm/Makefile`	The everyday interface — `build`, `up`, `shell`, `ssh`, `status`, `restore`, `clean`
`deployment/vm/mknode.sh`	Creates and provisions the VM
`deployment/vm/install_edumatcher.sh`	The in-VM installer `mknode.sh` runs
`deployment/vm/curl_setup_vm.sh`	Repository-free entry point; fetches the two scripts above and forwards its arguments
`deployment/vm/mkupd.sh`	Updates an existing VM in place
`deployment/vm/mkdevnode.sh`	A larger *development workstation* VM — a different thing, see below

VM or container?

Use `deployment/docker/` when you want the whole system — exchange plus all five web GUIs — running with one command, or when you are changing code and need a fast rebuild. Use the VM when you want a real machine, a snapshot you can roll back to, or a demo that survives being handed to somebody else. The VM installs the backend only; the web GUIs are part of the container deployment.

3.2 Why pip in a venv instead of pipx

For a shared VM runtime image, pip in a dedicated virtual environment fits better than pipx:

1. **Deterministic system paths** — every `pm-*` command resolves from `/usr/local/bin`, the same for every user on the machine.

2. **One controlled runtime location** — `/opt/edumatcher/.venv`, easy to inspect, replace or reason about.
3. **Easier service wiring** — unit files and scripts can reference stable absolute paths.

pipx remains the right choice for a per-user workstation install. This image has a system-wide command-link requirement that pipx does not serve as directly.

3.3 Build the VM

From a checkout, through the Makefile:

```
cd deployment/vm
make build                              # VM named 'ems', local wheel
make build NAME=ems-026 VERSION=0.26.0  # a pinned PyPI release
make build DEV=1                        # explicitly the wheel in ../../dist
make help                               # every target and variable
```

`make build` writes the VM name to `.vm-name`, so `make shell`, `make ssh`, `make status` and the rest need no arguments afterwards.

Without a repository checkout:

```
curl -fsSL https://raw.githubusercontent.com/johan162/EduMatcher/main/deployment/vm/curl_setup_vm.sh \
    | bash -s -- --version 0.76.5
```

`curl_setup_vm.sh` downloads `mknode.sh` and `install_edumatcher.sh` into a temporary directory and forwards everything after `--` to `mknode.sh`. Set `REPO_OWNER`, `REPO_NAME` or `REPO_REF` to pull them from a fork or branch.

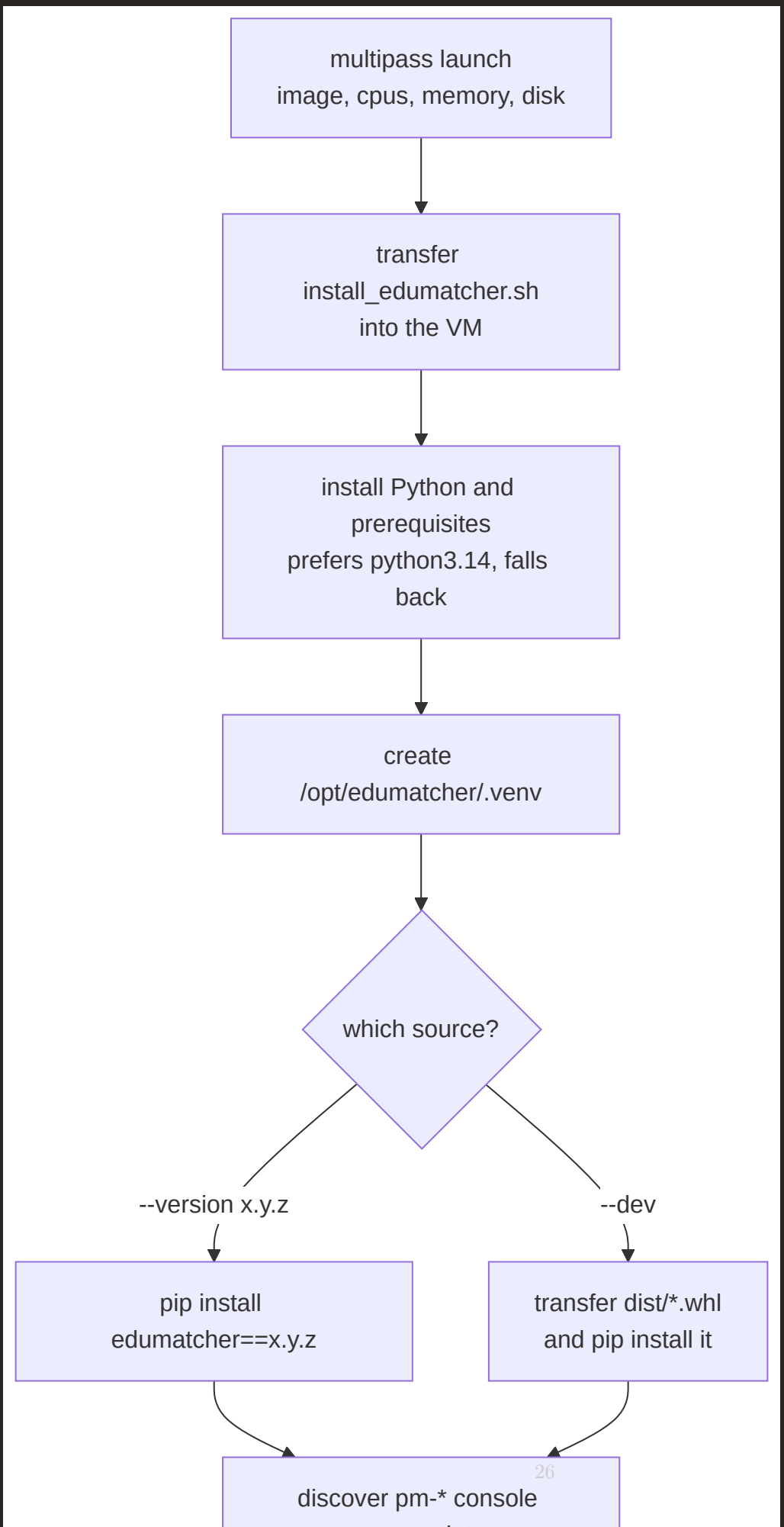
3.3.1 `mknode.sh` options

Option	Default	Description
`--name <vm-name>`	`ems`	VM instance name
`--image <image>`	`lts`	Base Multipass image
`--cpus <count>`	`4`	Virtual CPUs
`--memory <size>`	`4G`	RAM
`--disk <size>`	`6G`	Disk
`--version <x.y.z>`	`dev`	PyPI release to install
`--dev`	—	Install the local wheel from the repository's `dist/`. This is what happens by default when no `--version` is given
`--snapshot`	already enabled	Snapshot after provisioning. Snapshots are on by default, so this only makes it explicit — there is no `--no-snapshot`
`--snapshot-name <name>`	`clean`	Snapshot name
`--ssh-key <path>`	`~/.ssh/<vm-name>_ed25519`	Private key whose public half is installed for passwordless login
`--help`	—	Usage

The default is -dev, not a release

`DEFAULT_VERSION` is `dev`, so a bare `mknode.sh` or `make build` installs the wheel from the repository's `dist/` directory and fails if none is there. Run `poetry build` first, or pass `--version`/`VERSION=`.

3.4 What provisioning does



Two details are easy to miss:

- **`pm-setup` runs from `/home/ubuntu/session`, but that is not the data directory.** The deployed configuration goes to `/home/ubuntu/.local/share/edumatcher`, which provisioning exports as `EDUMATCHER_DATA_DIR` in `.bashrc`. `~/session` is only a convenient working directory.
- **The API keys are printed once, at the end.** `mknode.sh` finishes by showing the VM's IP address and the admin, TRADER01, TRADER02 and MM01 keys from the deployed configuration. Keep that output, or recover them later with `pm-config-show`.

3.4.1 Verify the install

Inside the VM:

```
ls -1 /usr/local/bin/pm-*                # the linked commands
ls -1 /opt/edumatcher/.venv/bin/pm-*     # their targets
cat /opt/edumatcher/EDUMATCHER_VERSION   # what was installed
```

3.5 Operate the VM

```
cd deployment/vm
make shell            # or: multipass shell ems
make ssh              # passwordless, as user 'ubuntu'
make status           # pm-opctl-cli list — one row per process
make health           # exit 0 when every process is OK
make info             # VM name and multipass state
make up / make down   # start and stop the VM
```

Inside the VM, the process manager starts the whole stack:

```
pm-opctl-cli start          # the 'default' profile
pm-opctl-cli list
pm-opctl-cli stop
```

To run processes by hand instead — the arrangement the Operator's Guide assumes — open a shell per process with `multipass shell ems` and start them separately:

Terminal	Command
1	`pm-engine --verbose`
2	`pm-audit --terminal`
3	`pm-clearing`
4	`pm-viewer --symbol AAPL`
5	`pm-alf-console --id TRADER01`

See Processes, Environment and Ports for what each one does and what the `default`, `mini` and `micro` profiles contain.

3.6 Snapshots, upgrades and reset

Provisioning takes a snapshot named `clean` by default. That is what makes this image worth using for teaching: after a class has filled the order books with nonsense, one command puts the VM back to freshly-provisioned.

```
make restore                # roll back to the 'clean' snapshot
make clean                  # multipass delete --purge
```

For a new release, prefer an immutable rebuild under a new name so the old VM stays available:

```
make build NAME=ems-026 VERSION=0.26.0
```

To reprovision an existing VM in place:

```
./mkupd.sh --vm-name ems
```

3.7 Development nodes are a different thing

`mkdevnode.sh` builds a larger *development workstation* VM from a checkout, with SSH access and the toolchain `scripts/verify_setup.` checks for. It is not a runtime image and not a substitute for `mknode.sh` or the curl path — use it when you want to develop EduMatcher inside a VM rather than run it.

3.8 Reference

- `deployment/vm/README.md` — the same material as a quick reference beside the scripts
- Container and Network Setup — the container deployment, which runs the same backend plus the web GUIs
- Installation — all five installation modes from a user's perspective

CHAPTER 4

Inside the Container Deployment

Learning objectives

After reading this chapter you will understand:

- What each file in `deployment/docker/` and `deployment/curl/` is for, and how the two directories relate to each other
- How the images are built, layer by layer, and how to rebuild any of them by hand without the helper scripts
- How the compose files stack on top of each other, and what each one adds: containers, environment variables, volumes and ports
- What actually happens when the backend container starts and stops
- Which networks, pods and volumes your container engine creates, and what they are called
- Every published port and why it exists, and every mount and why it is read-only or read-write
- How the `make` targets and `edumatcher.sh` commands map onto plain `podman` / `docker` commands

This chapter assumes you know roughly what an image and a container are, and nothing more. Installation is the *how to install* chapter and the complete flag reference; this one is the *how it works* walkthrough behind it. The developer-level view, including the release workflow and network debugging, is in the Architecture and Developer Guide's Container and Network Setup.

4.1 Five container terms, as they apply here

Container tools use a handful of words in very specific ways. Here is what each one means in an EduMatcher deployment.

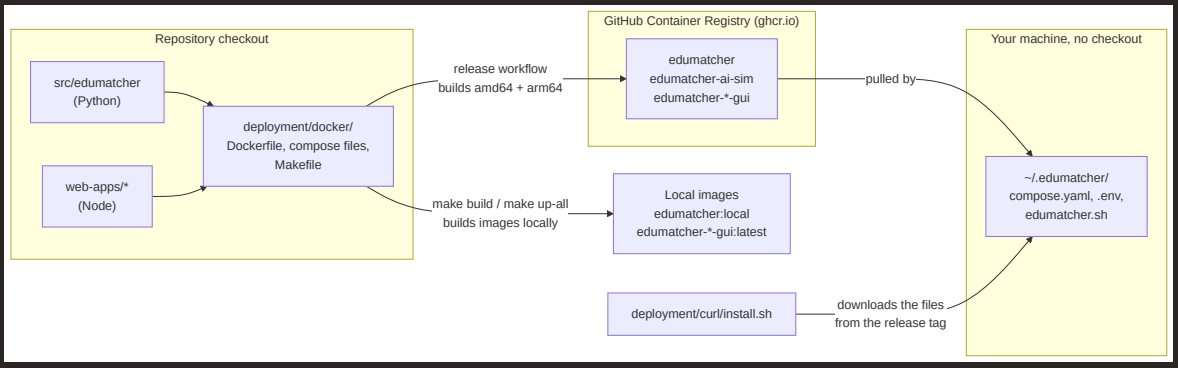
Term	In general	In EduMatcher
Image	A read-only template: a file system plus a default command	Seven of them: `edumatcher` (the whole exchange), `edumatcher-ai-sim` (the same exchange preset to trade by itself) and one per web GUI
Container	A running instance of an image, with its own processes, network interface and writable layer	Up to seven run at once: the exchange container and six GUI containers
Compose file	A YAML file that describes several containers and how they connect, so you start them with one command instead of six	`compose.yaml` and its overlays. Several files can be combined, each adding to the last
Bind mount and **named volume**	Two ways to give a container storage that outlives it. A bind mount is a directory on your disk; a named volume is storage the container engine manages for you	Everything the exchange produces lives in a bind mount (`./data`), so you can see it. The GUIs keep small private state in named volumes

Term	In general	In EduMatcher
Published port	A port on your machine forwarded to a port in a container	How *you* — your browser, `curl`, a protocol client — reach the system. The containers do not use them to reach each other

Two more appear further down: a **compose project** (the group of containers one compose command manages, which also gets its own private network) and a Podman **pod** (a named group of containers). Both are explained where they come up in Projects, networks and pods.

4.2 The two directories

The deployment lives in two directories. They describe **the same system**: one builds it from source, the other runs the released images.



	`deployment/docker/`	`deployment/curl/`
For	Changing EduMatcher and seeing the change run	Running a released version
Images come from	Built on your machine from the checkout	Pulled from `ghcr.io`, never built
Needs	Podman or Docker, plus Poetry and Node for the builds	Podman or Docker only
Driven by	`make` targets (`Makefile`)	`./edumatcher.sh`
Lives in	The repository	One directory, `~/.edumatcher` by default

Because they describe one system, the compose files in `deployment/curl/` are **twins** of those in `deployment/docker/`. They differ in only two ways: they name images on GHCR instead of having `build:` sections, and they put all six GUIs in one file.

4.2.1 What is in each directory

`deployment/docker/`:

File	Purpose
`Dockerfile`	Builds the backend image, in three stages: `builder`, `runtime` (the exchange) and `ai-sim` (the self-trading exchange)
`entrypoint.sh`	The script every backend container runs at start: deploys the configuration, starts the processes, waits, and shuts them down

File	Purpose
`compose.yaml`	The base compose file: the exchange container alone, with its gateway ports
`compose.zmq.yaml`	Overlay: also publish the internal ZeroMQ bus
`compose.ssh.yaml`	Overlay: run an SSH server in the exchange container
`compose.guis.yaml`	Overlay: the five web GUIs that talk to the exchange
`compose.config-gui.yaml`	Overlay: the configuration builder, which talks to nothing
`compose.ai-sim.yaml`	A separate, self-contained stack: the AI-trader simulation and five GUIs
`Makefile`	Every build and run command, with the right files and variables filled in
`.env.example`	The default settings, copied to `.env` on first use
`.dockerignore`	Keeps everything except the wheel and the entrypoint out of the build
`.wheel/`	A drop box: a Python wheel placed here is what the image installs
`README.md`	A summary of this chapter, beside the files

`deployment/curl/`:

File	Purpose
`install.sh`	The one-line installer: resolves a release, downloads the files below, writes `.env` and starts everything
`edumatcher.sh`	The control script copied into the install directory: `start`, `stop`, `status`, `config`, `update` and the rest
`compose.yaml`	The exchange and all six GUIs, from GHCR images
`compose.zmq.yaml`	Overlay: publish the ZeroMQ bus (`EM_ZMQ=1` in `.env`)
`compose.ai-sim.yaml`	The AI-trader simulation, from GHCR images (`install.sh --sim`)
`.env.example` / `.env.sim.example`	The default settings for the exchange and for the simulation
`README.md`	Usage notes for a released install

4.3 The images

4.3.1 Eight images, one backend Dockerfile

Image	Built from	Listens on (in the container)	Contains
`edumatcher`	`deployment/docker/Dockerfile`, stage `runtime`	All exchange ports (see Ports)	Python 3.14 and the `edumatcher` package with every `pm-*` command
`edumatcher-ai-sim`	same Dockerfile, stage `ai-sim`	same	The `runtime` image with different defaults: `s300-sim`, the `ai-sim` profile, Stockholm time

Image	Built from	Listens on (in the container)	Contains
`edumatcher-terminal-gui`	`web-apps/terminal-gui/Dockerfile`	8090	TapeDeck: a React front end served by a small Node server (the *bridge*)
`edumatcher-log-gui`	`web-apps/log-gui/Dockerfile`	8091	The Log Operator Console, plus a Python copy of `edumatcher` for its diagnostics view
`edumatcher-config-gui`	`web-apps/config-gui/Dockerfile`	8092	The configuration builder
`edumatcher-trader-gui`	`web-apps/trader-gui/Dockerfile`	8093	The trading GUI
`edumatcher-book-gui`	`web-apps/book-gui/Dockerfile`	8094	The order book viewer
`edumatcher-dash-gui`	`web-apps/dash-gui/Dockerfile`	8095	The operator dashboard

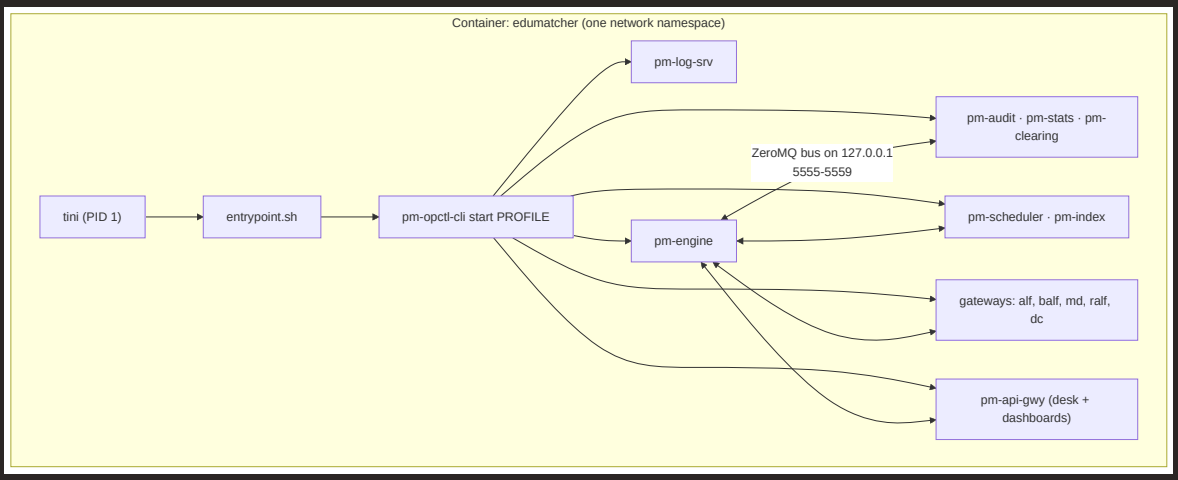
Locally built images are tagged `edumatcher:local` (from `IMAGE_NAME` and `IMAGE_TAG` in `.env`) and `edumatcher-<app>:latest`. Released images are `ghcr.io/johan162/<image>:<version>`, with `latest` following the newest release. One version tag pins all eight images to the same release.

4.3.2 Why the whole exchange is one container

It is tempting to expect one container per process: one for `pm-engine`, one for each gateway, and so on. EduMatcher deliberately does not do that.

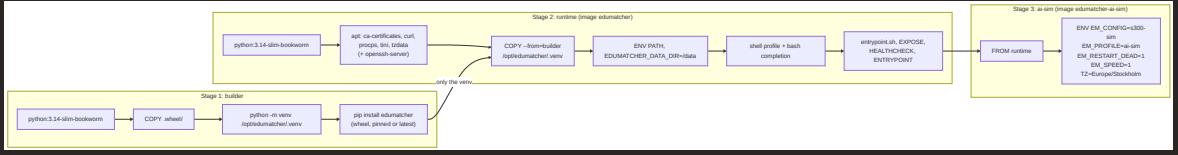
The exchange's processes talk to each other over a ZeroMQ message bus, and the engine binds that bus to `127.0.0.1`, the loopback interface. Loopback is private to one *network namespace*. Each container normally has its own, so processes in different containers could not see the bus. Putting every `pm-*` process in one container gives them one shared loopback, exactly as they have when you run them by hand on one machine or in the VM runtime image.

Inside that container, `pm-opctl-cli` is the process manager: it starts the processes of a *profile*, watches them and stops them. The container engine sees only one container; `pm-opctl-cli` manages what runs inside it.

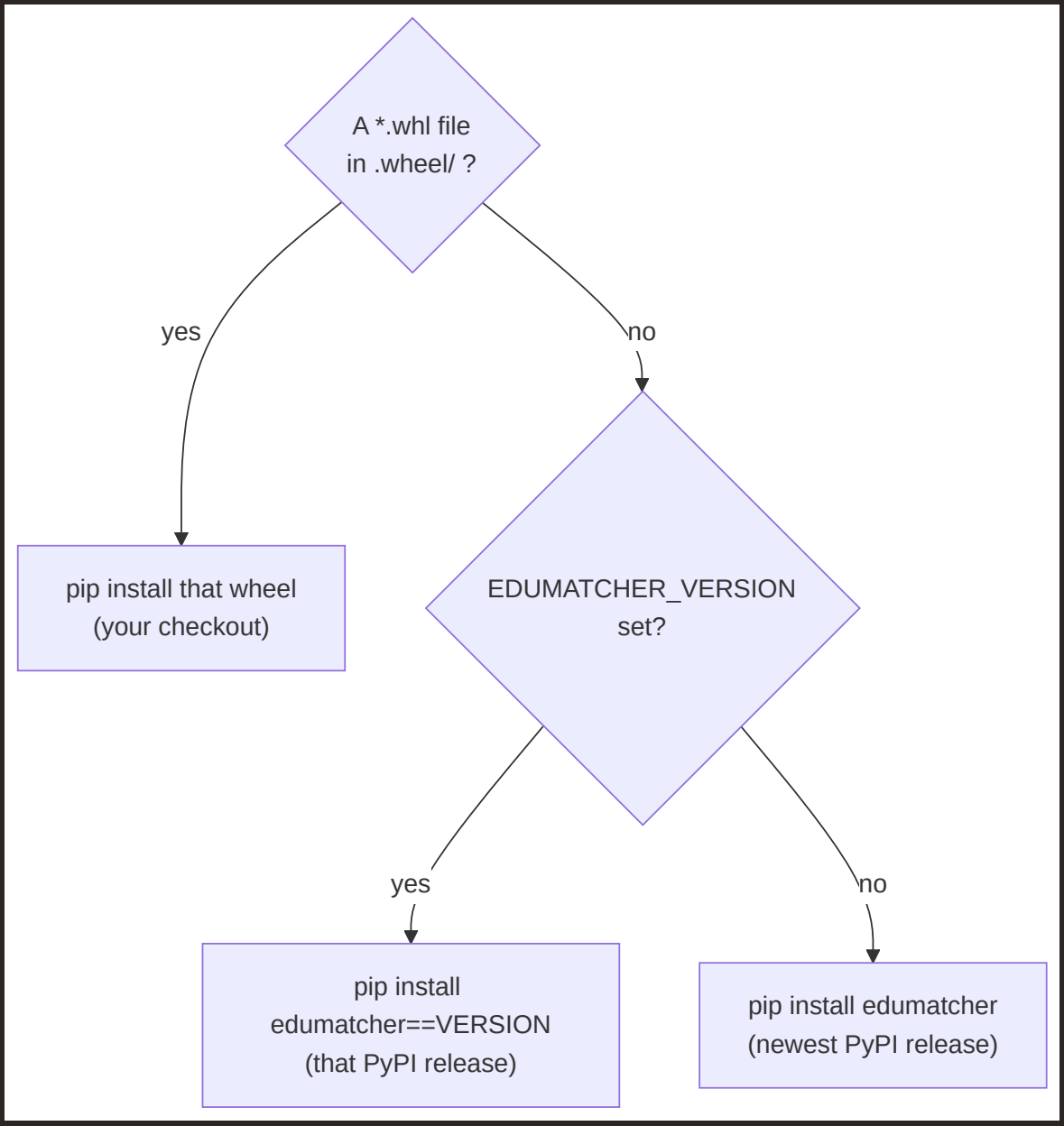


4.3.3 The backend Dockerfile, stage by stage

A *multi-stage* Dockerfile builds in steps. Each `FROM` starts a fresh stage, and a later stage copies only the results it needs out of an earlier one. Build tools, pip caches and temporary files therefore never reach the image you run.



Stage 1 — **`builder`.** Creates a Python virtual environment and installs EduMatcher into it. Where the package comes from is decided by what is in the `.wheel/` drop box and by one build argument:



`make build` puts a freshly built wheel in `.wheel/` unless you pass `PYPI=1` or `VERSION=`, in which case it empties the drop box first. That is what makes "build from my checkout" the default. A build log line `Installing local wheel: /tmp/wheel/edumatcher-...whl` confirms your code went in.

Stage 2 — **`runtime`.** A clean copy of the same base image, plus five small OS packages, each there for a reason:

Package	Why it is needed
`tini`	Runs as PID 1. `pm-opctl-cli` starts processes detached; when one exits, someone has to reap it, and a shell script is not good at that
`procps`	`pgrep` and `ps`, which `pm-opctl-cli` and the `*-cli health` commands use to find processes
`tzdata`	The time zone database. The scheduler and market calendars need real zone rules
`ca-certificates`, `curl`	HTTPS from inside the container, for diagnostics

Package	Why it is needed
`openssh-server`	Only when `WITH_SSH=1` (the default), for `make up SSH=1`

It then copies in the virtual environment from stage 1 and sets two environment variables every process depends on: `PATH`, so `pm-engine` and the rest are plain commands, and `EDUMATCHER_DATA_DIR=/data`, so every process reads and writes the mounted data directory. The same two settings are also written to `/etc/profile.d/` and `/root/.bashrc`. An SSH login does not inherit the image's environment, so it needs them there.

The last lines of the stage set up how the container runs:

- **`EXPOSE`** lists the ports the processes use. It is documentation only: nothing becomes reachable because of it. The compose files decide what is published.
- **`HEALTHCHECK`** opens a TCP connection to the engine's order-intake socket on 5555 every 30 seconds, after a 60-second grace period. `podman ps` or `docker ps` then show the container as `healthy` or `unhealthy`. For a check of every process, use `make health` (`pm-opctl-cli health`).
- **`ENTRYPOINT`** runs `tini`, which runs `entrypoint.sh`. See What happens when the exchange container starts.

Stage 3 — `ai-sim`. Adds nothing but default settings on top of `runtime`. The `edumatcher-ai-sim` image is the same software, preset to deploy the `s300-sim` example, run the `ai-sim` profile on Stockholm trading hours, and restart any process that dies. A plain `podman build` with no `--target` builds this last stage. That is why every compose file that builds the exchange names `target: runtime` explicitly.

4.3.4 Build arguments of the backend image

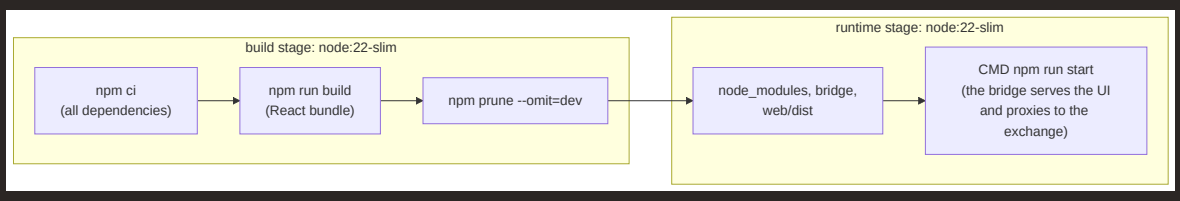
Build arguments are values you can pass at build time with `--build-arg`. They affect what goes into the image, not how it runs.

Argument	Default	Effect
`PYTHON_VERSION`	`3.14`	Tag of the `python:*-slim-bookworm` base image, for both stages
`EDUMATCHER_VERSION`	*(empty)*	PyPI release to install when `.wheel/` holds no wheel. Empty means the newest
`PIP_INDEX_URL`	*(empty)*	A PyPI mirror, such as a corporate Artifactory, for builds behind a firewall
`WITH_SSH`	`1`	`0` leaves out `openssh-server` for a smaller image

`compose.yaml` passes the last three from the variables of the same name in your `.env` or shell.

4.3.5 The web GUI images

All six GUI Dockerfiles follow one pattern, also in two stages:



Each image runs one small Node server, the **bridge**. It serves the browser application and relays its requests to the exchange, so the browser only ever talks to one port. The bridges have no durable state of their own. The named volumes mounted into them hold a failover log, used while `pm-log-srv` is unreachable, and the log console's acknowledgements.

Their build arguments are mostly for corporate networks: `HTTP_PROXY`, `HTTPS_PROXY`, `NO_PROXY`, `NPM_STRICT_SSL` (`false` when a proxy re-signs TLS), `NPM_REGISTRY` (an npm mirror) and `WEB_PORT`. Image build arguments in the installation chapter lists which of them the compose files actually pass on.

The log console carries a copy of EduMatcher too

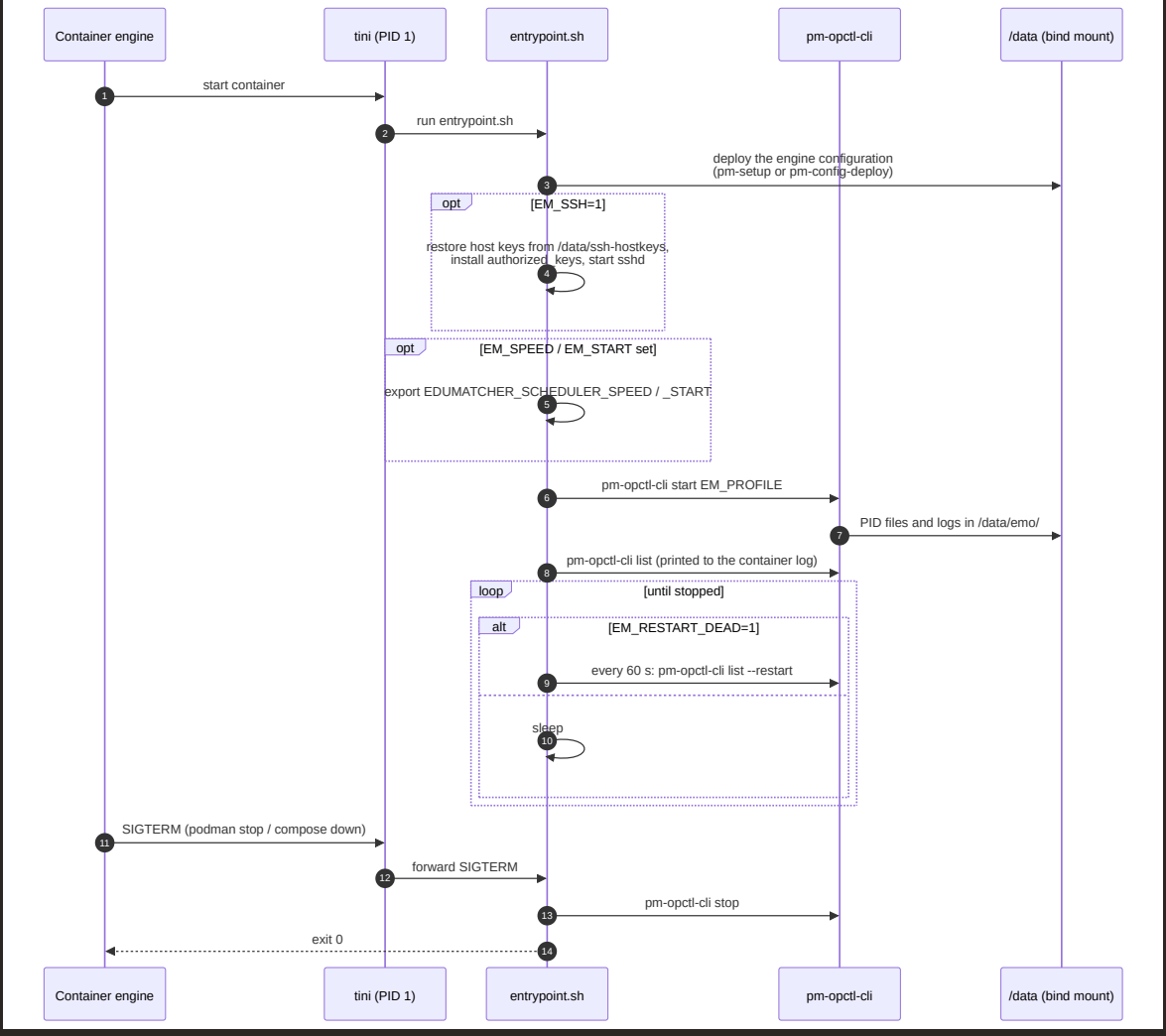
`edumatcher-log-gui` is the one GUI image with Python in it. Its diagnostics view runs `pm-log-cli` rather than reimplementing that tool in TypeScript. It picks its `edumatcher` package the same way the backend does, from a wheel in `web-apps/log-gui/.wheel/`, and `make build-guis` fills that drop box by the same rule as `make build`: a wheel built from your checkout, or a PyPI release with `PYPI=1` or `VERSION=x.y.z` (passed to the build as `EDUMATCHER_VERSION`).

4.3.6 How released images are built

You never need to do this yourself, but it explains what you pull. Publishing a GitHub release runs `.github/workflows/publish-images`. It builds all eight images (`edumatcher`, `edumatcher-ai-sim` and the six GUIs) natively on an `amd64` and an `arm64` runner, and joins each pair into one multi-arch manifest. Both backend images and the log console get the release's own wheel in their `.wheel/` drop box, so their Python code is exactly the release. The same tag is pushed for every image; that is why one `EM_VERSION` pins the whole system.

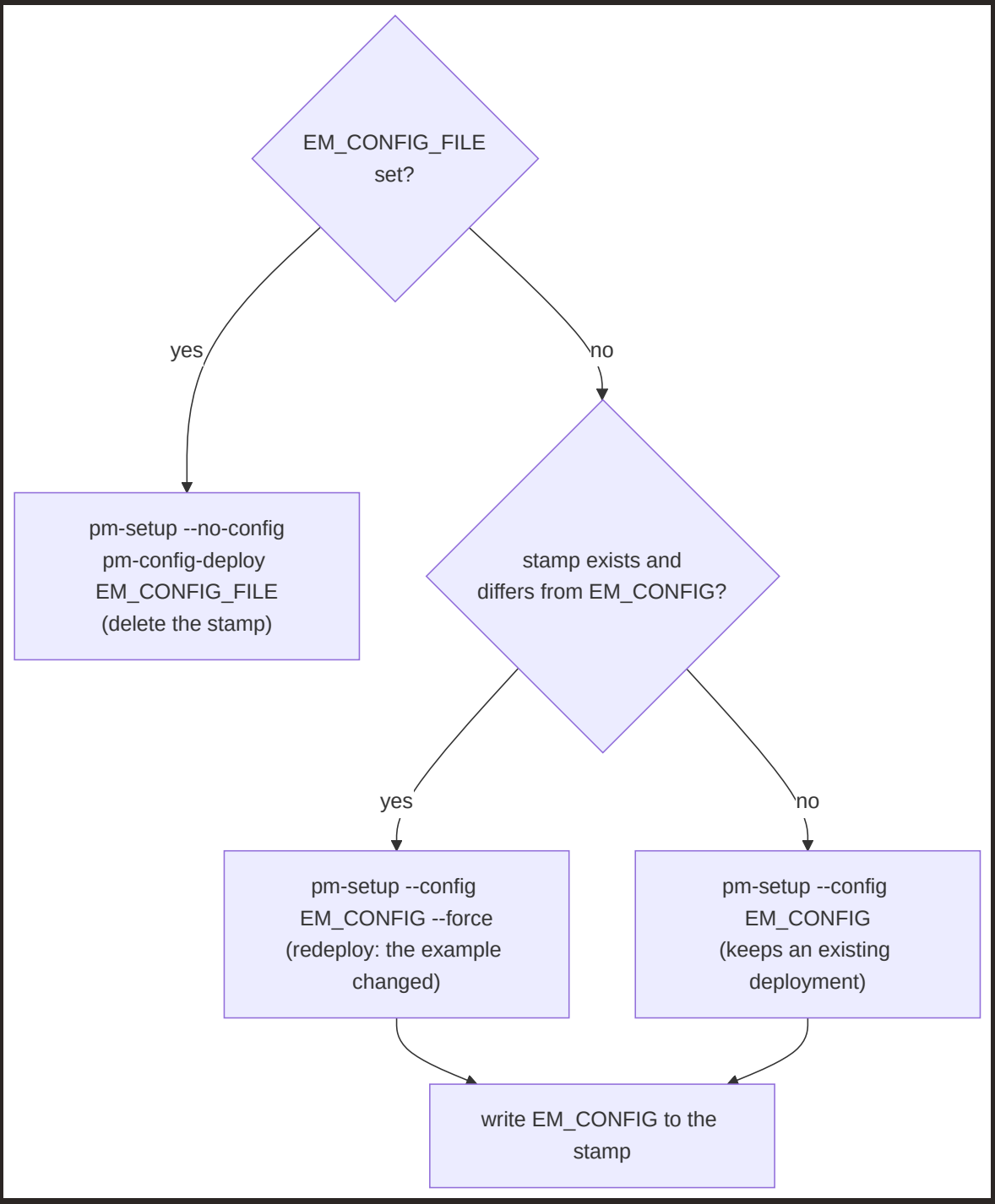
4.4 What happens when the exchange container starts

The container's command is `tini -- entrypoint.sh`. `entrypoint.sh` reads a handful of environment variables (all set by the compose files from your `.env`) and runs four steps.



4.4.1 Step 1: deploying the configuration

Which configuration is deployed depends on two variables and on a small stamp file, `/data/.container-config`, which records the example name the data directory was built from:



In practice:

- **A configuration file of your own** (`EM_CONFIG_FILE`) is deployed on *every* start. Editing the file and restarting is the whole edit-and-test loop.
- **A bundled example** (`EM_CONFIG`) is deployed once and then kept across restarts. That is what you want when you have edited `data/ref_data/engine_config.yaml` by hand. Naming a *different* example redeploys.

Either way the result is `/data/ref_data/engine_config.json`, the compiled configuration every process reads.

4.4.2 Step 4: stopping cleanly

`podman stop`, `docker stop` and `compose down` send the container `SIGTERM`, then wait for a grace period before killing it. `tini` forwards the signal to `entrypoint.sh`, whose handler runs `pm-opctl-cli stop`. That stops every process of the profile in order, so each one can flush its files. `stop_grace_period` in the compose files (30 seconds, 60 for the simulation) gives that walk enough time before the engine resorts to `SIGKILL`.

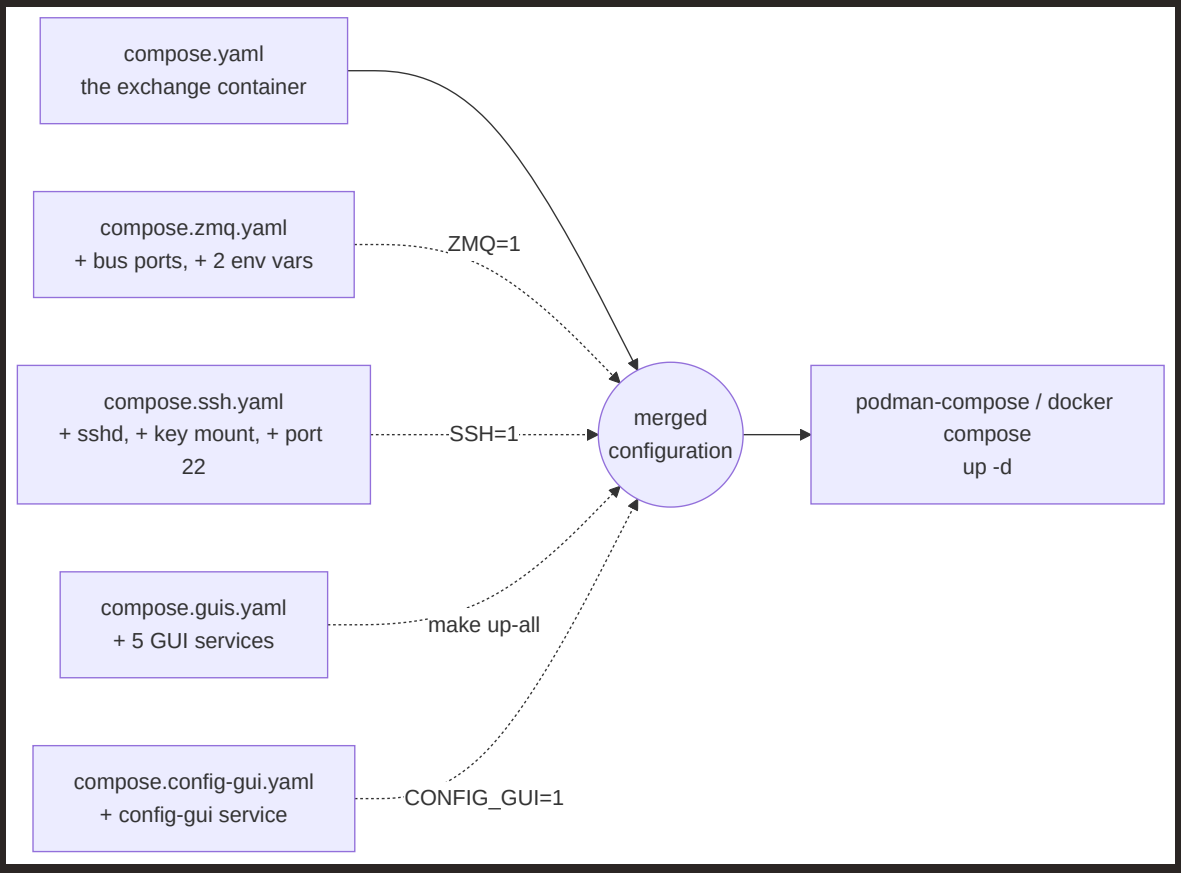
The container log is the entrypoint's log

`make logs`, `podman logs edumatcher` or `./edumatcher.sh logs edumatcher` show the `==>` lines above: which configuration was deployed, which profile started, and the process table. Each process writes its own log to `data/emo/<name>.log`.

4.5 The compose files, one by one

4.5.1 How compose files combine

Compose reads one or more files, given with `-f`, and **merges** them in order. A later file can add services, add entries to a service's environment, ports or volumes, or override single values. The base file stays small; each optional feature is a separate *overlay* you add only when you want it.



The `make` flags only choose which `-f` arguments are passed:

Command	Files passed
`make up`	`compose.yaml`
`make up ZMQ=1 SSH=1`	`compose.yaml compose.zmq.yaml compose.ssh.yaml`
`make up-all`	`compose.yaml compose.guis.yaml`
`make up-all ZMQ=1 CONFIG_GUI=1`	`compose.yaml compose.zmq.yaml compose.guis.yaml compose.config-gui.yaml`
`make up-sim`	`compose.ai-sim.yaml` alone, as project `edumatcher-sim`

To see the result of a merge without starting anything, ask compose to print it:

```
cd deployment/docker
podman-compose -f compose.yaml -f compose.zmq.yaml config    # or: docker compose ... config
```

4.5.2 `${VARIABLE:-default}`: where the values come from

The compose files contain almost no fixed settings. Nearly every value is a reference such as `${BIND_ADDR:-127.0.0.1}`, meaning *the variable `BIND_ADDR`, or `127.0.0.1` if it is unset.* Compose fills these in from your shell environment first, then from the `.env` file beside the compose file. When a value can be set in several places, the nearest one wins:

```
make flag  →  shell environment  →  .env  →  ${VAR:-default} in the compose file  →  ENV in the image
```

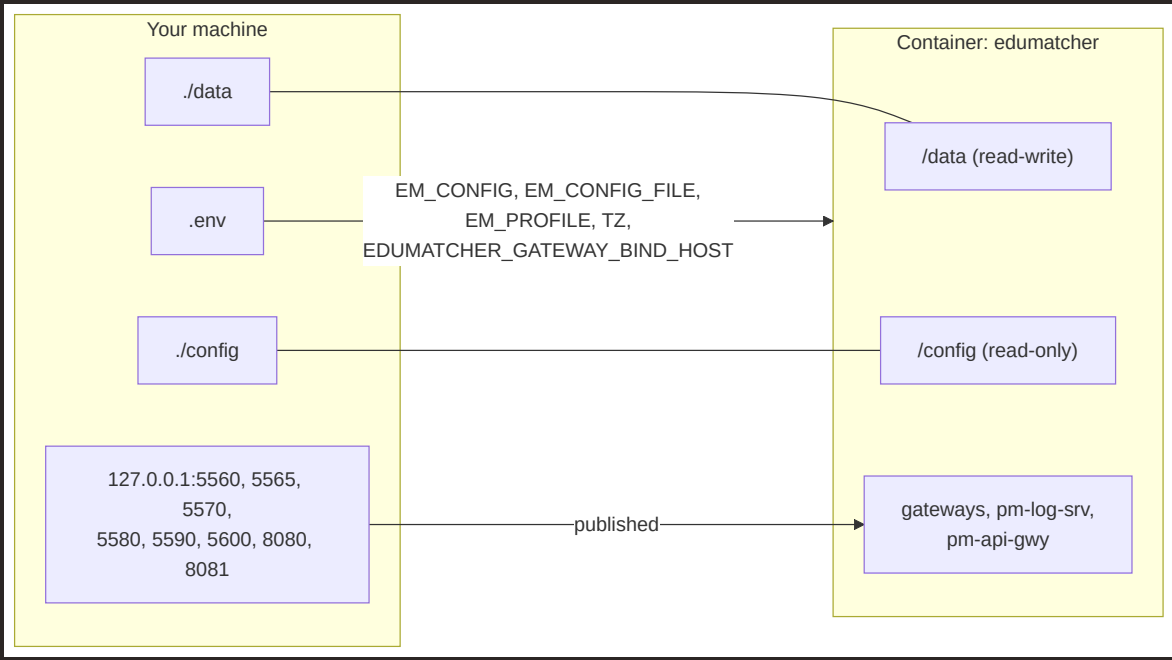
So `make up CONFIG=s10-nominal` affects that one start only and does not edit `.env`. The next plain `make up` goes back to what `.env` says.

Two kinds of variable

Some variables are passed **into** a container as its environment (they appear under `environment:`), for example `EM_CONFIG`. Others are used only by compose itself to fill in the file and never reach a container, for example `BIND_ADDR`, the GUI host ports and `IMAGE_NAME`. The tables below say which is which.

4.5.3 `compose.yaml` — the exchange

The base file defines one service, `edumatcher`, built from the Dockerfile's `runtime` stage.



Section	Setting	Why
`build`	`context: .`, `target: runtime`, three `args`	Build the exchange (not the AI-sim stage) from this directory
`image`	`${IMAGE_NAME:-edumatcher}:${IMAGE_TAG:-dev}`	The name the built image is tagged with
`container_name` / `hostname`	`edumatcher`	A fixed name, so `podman exec edumatcher ...` works and the GUIs can find it
`restart`	`unless-stopped`	Restart after a crash, but not after you stopped it deliberately

Section	Setting	Why
`stop_grace_period`	`30s`	Time for `pm-opctl-cli stop` to stop every process cleanly
`environment`	see below	What `entrypoint.sh` and the processes read
`volumes`	`./data:/data`, `./config:/config:ro`	See Volumes and mounts
`ports`	eight client-facing ports	See Ports

Environment passed into the container:

Variable	Default	Read by	Effect
`EM_CONFIG`	`s10-basic`	`entrypoint.sh`	Bundled example configuration to deploy
`EM_CONFIG_FILE`	*(empty)*	`entrypoint.sh`	Path *inside the container* to a configuration of your own; wins over `EM_CONFIG`
`EM_PROFILE`	`default`	`entrypoint.sh`	Which `pm-opctl-cli` profile to start: `default`, `mini`, `micro`, `mm-demo`, `ai-swarm`, `ai-sim`
`EDUMATCHER_GATEWAY_BIND_HOST`	`0.0.0.0`	every gateway, `pm-log-srv`, `pm-api-gwy`	The interface the client-facing listeners bind inside the container. `0.0.0.0` is what lets the GUI containers reach them
`TZ`	`Europe/Stockholm`	everything	The container's time zone. Match it to your trading calendar

`entrypoint.sh` also reads `EM_SSH`, `EM_SPEED`, `EM_START` and `EM_RESTART_DEAD`, which the overlays and the AI-sim stack set.

4.5.4 `compose.zmq.yaml` — the raw message bus

Adds seven ports and two environment variables to the `edumatcher` service, so tools on your machine (`pm-calf-spy`, `pm-dc-spy`, `pm-index-cli`, your own ZeroMQ clients) can attach to the bus directly.

Adds	Value	Why
`EDUMATCHER_ENGINE_BIND_HOST`	`0.0.0.0`	The engine binds its bus to `127.0.0.1` by default; publishing the port alone would forward to nothing
`EDUMATCHER_INDEX_BIND_HOST`	`0.0.0.0`	Same for `pm-index`
ports	5555–5559, 5601, 5602	The engine's, the index's and the log server's ZeroMQ sockets

The bus has no authentication

Anything that can reach 5555 can submit orders as if it were a gateway. Keep `BIND_ADDR=127.0.0.1` while this overlay is on.

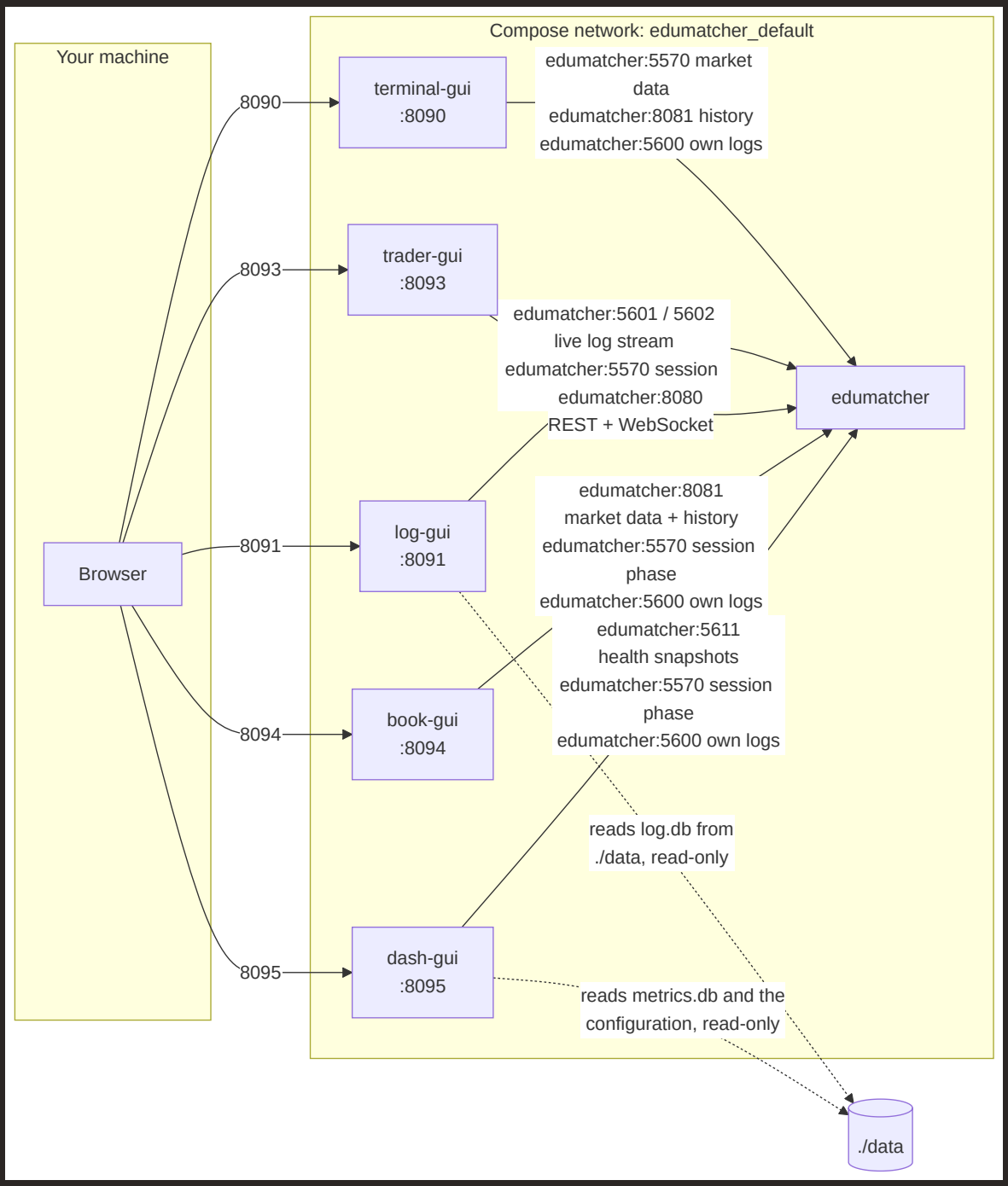
4.5.5 `compose.ssh.yaml` — SSH into the exchange

Adds	Value	Why
`EM_SSH`	`1`	Tells `entrypoint.sh` to start `sshd`
volume	`./.ssh/authorized_keys:/run/host-authorized_keys:ro`	Your public keys, collected by `make keys` from `~/.ssh/*.pub`
port	`${SSH_PORT:-2222}` → 22	`ssh -p 2222 root@localhost`

The keys are mounted read-only and *copied* into place, because `sshd` refuses key files it does not own with mode `0600`. The container's host keys are generated once and kept in `data/ssh-hostkeys/`, so recreating the container does not trigger a "host key changed" warning. No key material is ever built into the image.

4.5.6 `compose.guis.yaml` — the four connected web GUIs

Adds four services in the **same compose project** as the exchange. That is the important part: being in one project puts them on one private network, where the exchange answers to the hostname `edumatcher`.



Service	Talks to	Environment that wires it	Storage
`terminal-gui`	`pm-md-gwy` (CALF, 5570), `pm-api-gwy` dashboards (8081), `pm-log-srv` (5600)	`CALF_HOST`/`CALF_PORT`, `API_GATEWAY_URL`, `PM_TERMINAL_API_KEY`, `LOG_SRV_HOST`/`LOG_SRV_PORT`, `INDEX_IDS`, `MAX_WS_CLIENTS`	named volume `terminal-gui-logs`
`log-gui`	`pm-log-srv` LALF-PS (5601 PUB, 5602 PULL), `pm-md-gwy` (CALF, 5570)	`LOG_SRV_HOST`, `LOG_SRV_PUB_PORT`, `LOG_SRV_PULL_PORT`, `CALF_HOST`, `LOG_DB_PATH`, `ACK_STORE_PATH`	`./data` read-only at `/backend-data`; named volume `log-gui-acks`
`trader-gui`	`pm-api-gwy` desk (8080)	`API_PROXY_TARGET`	none

Service	Talks to	Environment that wires it	Storage
book-gui	pm-api-gwy dashboards (8081), pm-md-gwy (CALF, 5570), pm-log-srv (5600)	API_GATEWAY_URL, PM_BOOK_API_KEY, CALF_HOST, LOG_SRV_HOST/LOG_SRV_PORT	named volume book-gui-logs
dash-gui	pm-monitor (5611 PUB), pm-md-gwy (CALF, 5570), pm-api-gwy dashboards (8081), pm-log-srv (5600)	MONITOR_HOST/MONITOR_PUB_PORT, CALF_HOST, API_GATEWAY_URL, PM_DASH_API_KEY, METRICS_DB_PATH, CONFIG_ARTIFACT_PATH, LOG_GUI_URL/BOOK_GUI_URL, LOG_SRV_HOST/LOG_SRV_PORT	./data read-only at /backend-data; named volume dash-gui-logs

Every GUI also gets HOST=0.0.0.0 (listen on the container's interface), PORT (its own port) and CORS_ORIGIN. depends_on: edumatcher only controls start order. It does not wait for the exchange to be ready, which is one reason make up-all starts in two phases (see below).

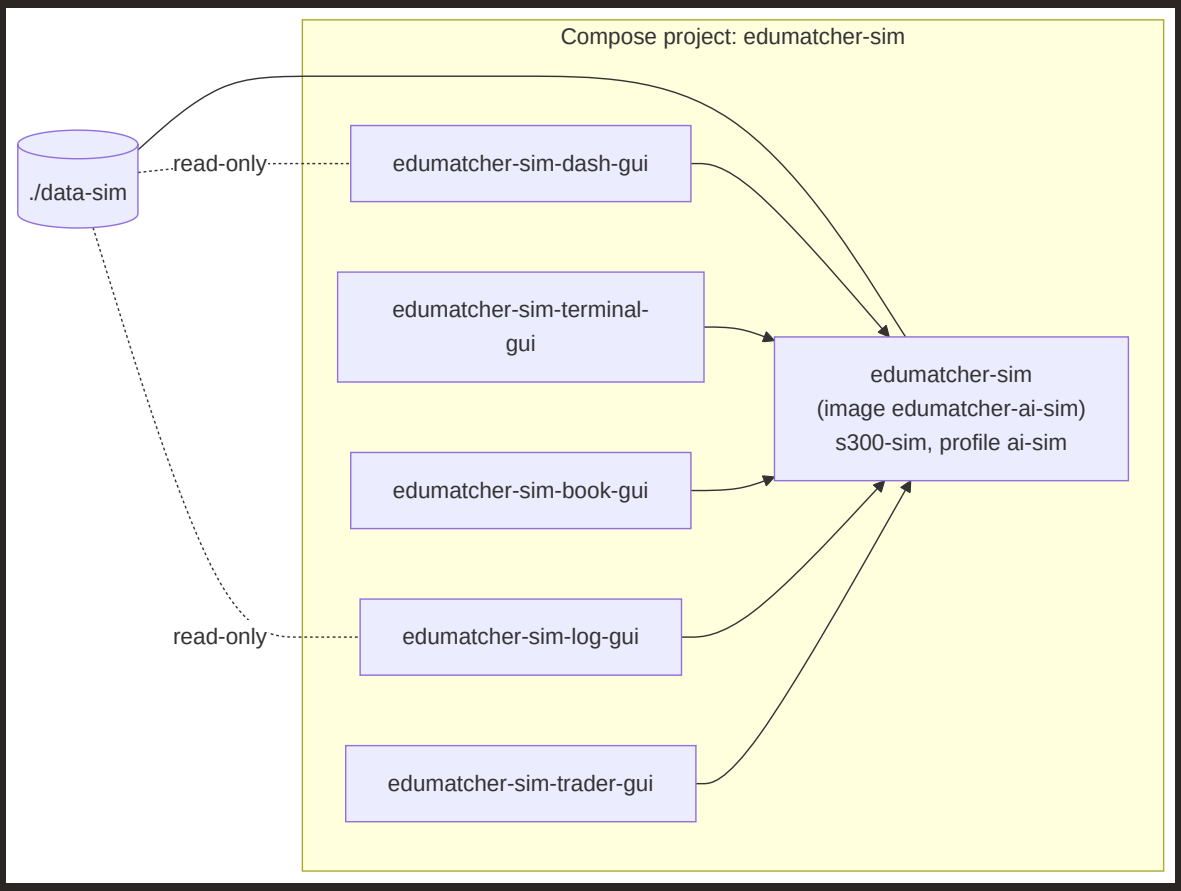
Note what is **not** involved: none of the GUI-to-exchange traffic uses a published port. The exchange's published ports are there for you; the GUIs reach it across the private network. The log console's live stream on 5601/5602 therefore works without ZMQ=1. pm-log-srv binds those sockets on 0.0.0.0 inside the container, and the network carries the traffic. The same holds for the dashboard and pm-monitor's 5611, which EDUMATCHER_GATEWAY_BIND_HOST opens inside the container.

4.5.7 compose.config-gui.yaml — the configuration builder

One service, config-gui, on port 8092. It writes configuration files in the browser and talks to nothing else, so it is kept out of compose.guis.yaml and added only with CONFIG_GUI=1. Unlike the other GUIs it has no depends_on.

4.5.8 compose.ai-sim.yaml — the self-trading market

A different kind of file: **self-contained**, not an overlay. It sets its own project name (name: edumatcher-sim) and pulls every image from GHCR by default. It needs no .env and no build.



Its differences from the plain exchange:

Setting	Value	Why
backend image	`${AISIM_IMAGE:-ghcr.io/.../edumatcher-ai-sim:${EM_VERSION#v}-latest}`	make up builds a locally built `edumatcher-ai-sim:local`
`EM_SPEED` / `EM_START`	`1` / *(empty)*	Run the trading calendar faster than real time, from a chosen start
data	`./data-sim`	Its own data directory, so it never touches the exchange's `./data`
`stop_grace_period`	`60s`	100 AI traders and the market model take longer to stop
read-only key	`PM_READONLY_API_KEY`, default `key-readonly-9vaho7`	Fixed, because the configuration is fixed: no two-phase start needed
GUIs	four (no config-gui)	Nothing to configure in a preset simulation

It publishes **the same host ports** as the exchange stack, so only one of the two can run at a time. Running the simulation is covered in the AI Traders book's The Simulation Container.

4.6 Projects, networks and pods

When you run `up`, the container engine creates more than containers. Knowing the names helps when you list things with `podman ps`, `podman network ls` or `podman volume ls`.

4.6.1 The compose project and its names

Every compose run belongs to a **project**. The project name prefixes the network and the named volumes it creates. It comes from, in order: `-p` on the command line, `name:` in the compose file, `COMPOSE_PROJECT_NAME` in `.env`, and finally the directory name.

Stack	Project name	Why
`deployment/docker`, `make up` / `up-all`	`edumatcher`	`COMPOSE_PROJECT_NAME=edumatcher` in `.env`. Without a `.env` it would be `docker`, the directory name
curl install in `~/.edumatcher`	`edumatcher`	The directory name, with the leading dot dropped
either AI-sim stack	`edumatcher-sim`	`name:` in the file; the `Makefile` also passes `-p edumatcher-sim`

What a project named `edumatcher` creates:

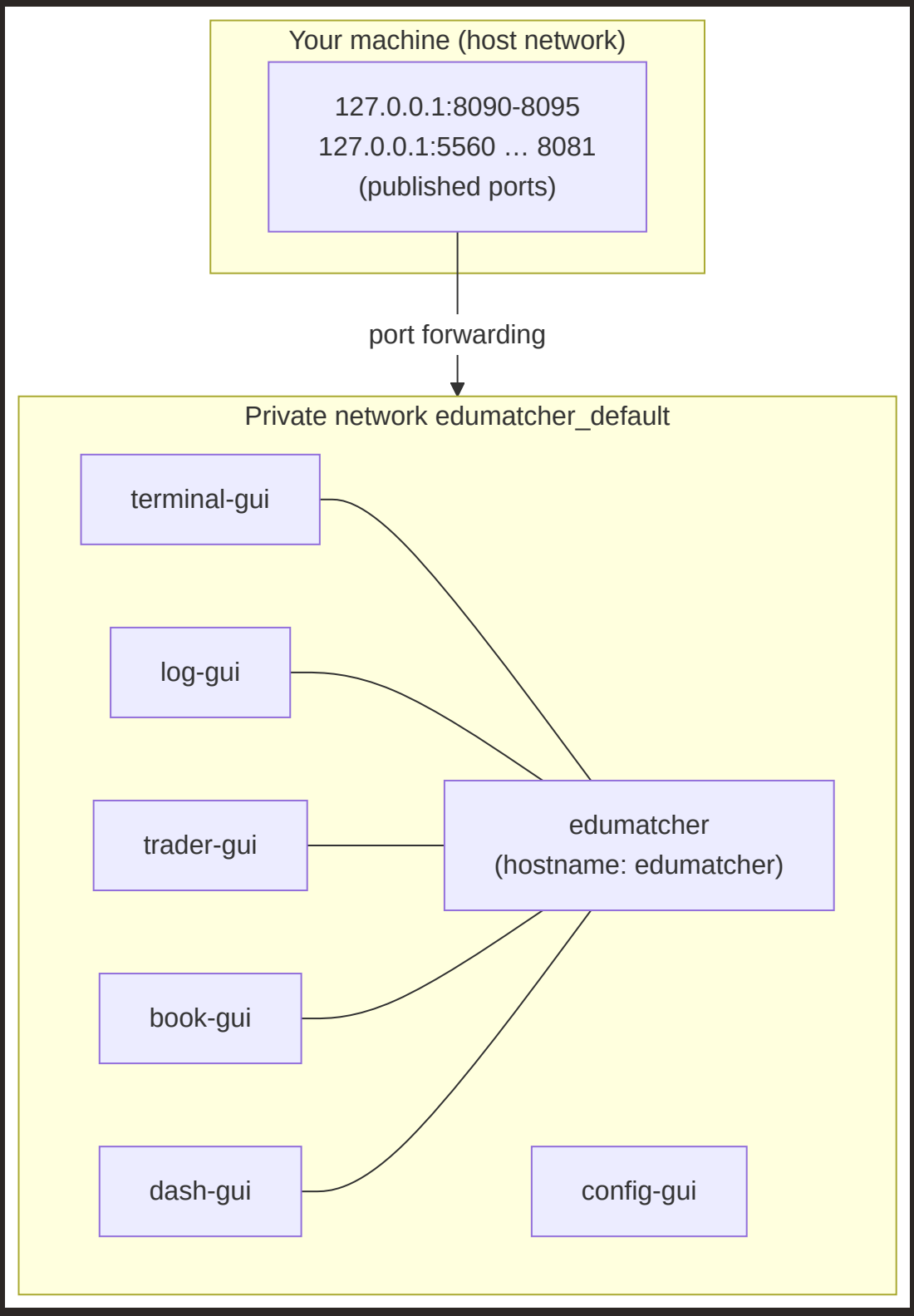
Object	Name	Created by
Network	`edumatcher_default`	Compose, for every project
Containers	`edumatcher`, `edumatcher-terminal-gui`, `edumatcher-log-gui`, `edumatcher-trader-gui`, `edumatcher-book-gui`, `edumatcher-dash-gui`, `edumatcher-config-gui`	`container_name:` in the files
Named volumes	`edumatcher_log-gui-acks`, `edumatcher_terminal-gui-logs`, `edumatcher_book-gui-logs`, `edumatcher_dash-gui-logs`	`volumes:` at the bottom of the GUI files
Pod (Podman only)	`pod_edumatcher`	`podman-compose`

One stack at a time

A source-built stack and a curl install have the **same project name, the same container names and the same host ports**. If one is running, the other would quietly attach to its containers. Both tools check for this and refuse to start: `make up-all` checks via `check-foreign`, and `./edumatcher.sh start` checks that the existing `edumatcher` container mounts *its own* `data` directory. Stop one before starting the other. `make mounts` and `./edumatcher.sh mounts` show which directory each running container really uses.

4.6.2 The network

Compose gives each project one private network and attaches every service to it, registering the **service name** as a hostname on that network. The exchange's service is called `edumatcher`, so `edumatcher:8081` from inside `terminal-gui` reaches the exchange's REST API.



Each container has its own network namespace and its own `127.0.0.1`. A process listening on `127.0.0.1` inside `edumatcher` is invisible to the GUI containers. That is why the client-facing listeners bind `0.0.0.0`: *every interface of this container*, which still means only this private network plus whatever ports you publish. How the containers are wired covers the three meanings of `127.0.0.1` / `0.0.0.0` in detail.

4.6.3 Pods (Podman only)

A **pod** is Podman's way of grouping containers. Docker has no pods; with Docker this section does not apply.

`podman-compose` puts all containers of a project into one pod named `pod_<project>`. EduMatcher's stacks get `pod_edumatcher`

or `pod_edumatcher-sim`. It creates that pod with `--infra=false --share=`, which means the containers in it **share no namespaces**: each keeps its own network stack, and they talk over the compose network exactly as described above. The pod is a label that lets you manage the group as a whole:

```
podman pod ls                       # pod_edumatcher, with its container count
podman pod ps                       # (same, older syntax)
podman ps --pod                     # containers with their pod
podman pod stop pod_edumatcher      # stop the whole group
podman pod logs pod_edumatcher      # every container's output
```

You never need to create the pod yourself: `up` creates it, `down` removes it. Use `make down-all` or `./edumatcher.sh stop` rather than removing the pod by hand, so compose also cleans up the network.

podman compose versus podman-compose

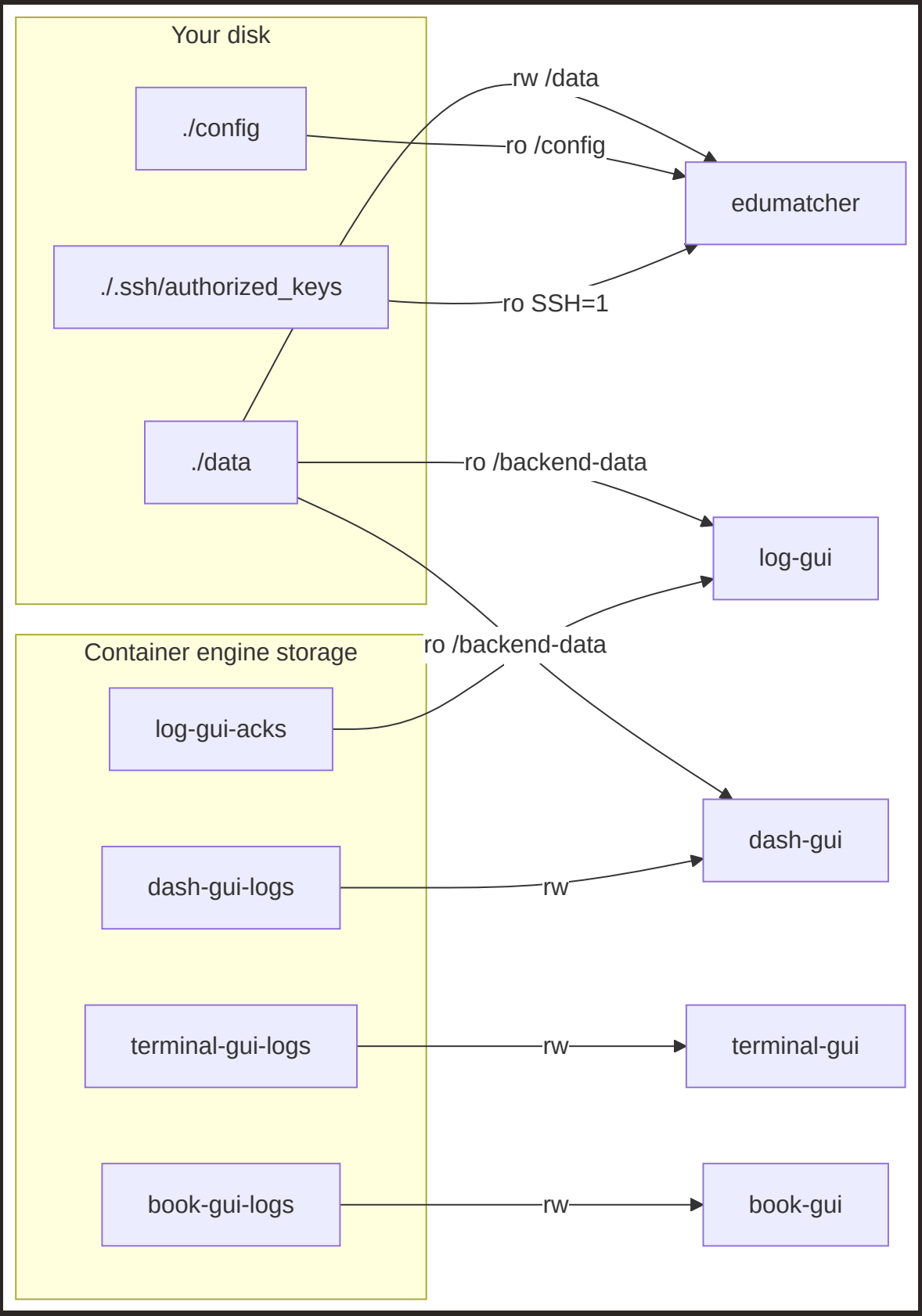
`podman compose` (with a space) is a thin wrapper that runs whichever compose program it finds first, which may be Docker's `docker-compose`. The `Makefile` and `edumatcher.sh` prefer the standalone `podman-compose` when it is installed. `make info` shows which one was picked.

4.7 Volumes and mounts

Every piece of storage a container uses, and why it is set up the way it is:

Container	Container path	Host side	Type	Mode	Why
`edumatcher`	`/data`	`./data` (curl: `~/.edumatcher/data`)	bind mount	read-write	The exchange's whole state: configuration, databases, audit log, process logs. On your disk so it survives every rebuild and you can read it directly
`edumatcher`	`/config`	`./config`	bind mount	**read-only**	A configuration file of your own, deployed from here at every start. Read-only because the exchange has no business changing your source file
`edumatcher`	`/run/host-authorized_keys`	`~/.ssh/authorized_keys`	bind mount, `SSH=1` only	**read-only**	Your public keys; copied into place by the entrypoint
`log-gui`	`/backend-data`	`./data`	bind mount	**read-only**	The console reads `log.db` directly for history, and must never write the log server's database
`log-gui`	`/app/ack-data`	`<project>_log-gui-acks`	named volume	read-write	Which log entries an operator has acknowledged

Container	Container path	Host side	Type	Mode	Why
`terminal-gui`	`/app/logs`	`<project>_terminal-gui-logs`	named volume	read-write	Failover log, written while `pm-log-srv` is unreachable
`book-gui`	`/app/logs`	`<project>_book-gui-logs`	named volume	read-write	Same, for the order book viewer
`dash-gui`	`/backend-data`	`./data`	bind mount	**read-only**	The dashboard reads `metrics.db` (history) and `ref_data/engine_config.` (the threshold bands), and must never write either
`dash-gui`	`/app/logs`	`<project>_dash-gui-logs`	named volume	read-write	Failover log, for the operator dashboard
AI-sim `edumatcher-sim`	`/data`	`./data-sim` (curl `--sim`: `~/.edumatcher-sim/data`)	bind mount	read-write	The simulation's own state, separate from the exchange's



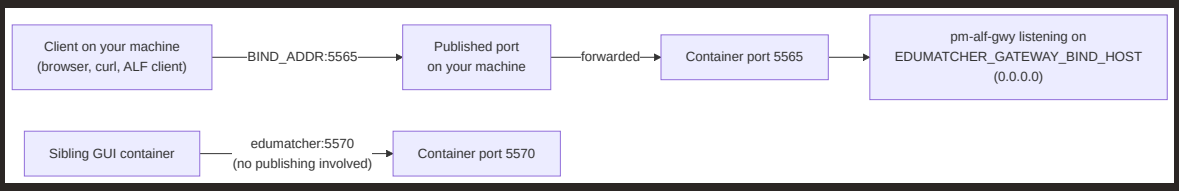
The split is deliberate. **Exchange data is in a bind mount** because it is yours: you back it up, inspect it with `sqlite3`, delete it to start over. The GUI volumes hold **disposable private state** that nothing else needs, so they live where the container engine keeps such things. They are removed by `./edumatcher.sh uninstall` or `compose down --volumes`; `./data` is not.

The :z on every bind mount

Every compose file adds `:z` to its bind mounts (`./data:/data:z`, `./data:/backend-data:ro,z`). On SELinux systems such as Fedora and RHEL, `:z` asks the container engine to relabel the directory so containers may use it; without it the exchange cannot write `./data` at all. Elsewhere it has no effect. Keep it in a compose file of your own.
What lives in `./data` is listed in Directories and paths.

4.8 Ports and why each one exists

A port appears in up to three places. The **container port** is where the process listens. The **published port** is the matching port on your machine, created by `ports:` in a compose file. `BIND_ADDR` decides which of your machine's interfaces it opens on. Unpublished ports are reachable only inside the container, or across the compose network if the process binds `0.0.0.0`.



4.8.1 The exchange container

	Port	Process	Kind	Published	Why it exists
	5560	`pm-balf-gwy`	TCP	always	Binary ALF order entry for trading clients
	5565	`pm-alf-gwy`	TCP	always	Text ALF order entry for trading clients and the protocol examples
	5570	`pm-md-gwy`	TCP	always	Market data (CALF). Also what `terminal-gui` reads its live feed from
	5580	`pm-ralf-gwy`	TCP	always	Post-trade feed (RALF) for clearing and drop-copy clients
	5590	`pm-dc-gwy`	TCP	always	Drop copy (DCLF)
	5600	`pm-log-srv`	TCP	always	Log ingest (LALF). Clients outside the container, including the GUI bridges, send their logs here
	8080	`pm-api-gwy`	HTTP	always	REST and WebSocket API, `desk` instance: trading credentials. Swagger UI at `/docs`. `trader-gui` uses it

Port	Process	Kind	Published	Why it exists
8081	`pm-api-gwy`	HTTP	always	REST and WebSocket API, `dashboards` instance: the read-only key. `terminal-gui` and `book-gui` use it
5555	`pm-engine`	ZeroMQ PULL	`ZMQ=1`	Order intake: every gateway sends orders to the engine here
5556	`pm-engine`	ZeroMQ PUB	`ZMQ=1`	Event and order-book feed
5557	`pm-engine`	ZeroMQ PUB	`ZMQ=1`	Drop-copy feed
5558	`pm-index`	ZeroMQ PUB	`ZMQ=1`	Index values
5559	`pm-index`	ZeroMQ PULL	`ZMQ=1`	Index commands
5601	`pm-log-srv`	ZeroMQ PUB	`ZMQ=1`	Live log broadcast (LALF-PS). `log-gui` reads it across the network either way
5602	`pm-log-srv`	ZeroMQ PULL	`ZMQ=1`	Log console control channel
5611	`pm-monitor`	ZeroMQ PUB	`ZMQ=1`	Health snapshots and events. `dash-gui` reads it across the network either way; published for `pm-monitor-cli` or a dashboard run from source
5610	`pm-monitor`	ZeroMQ PULL	never	Health beacons from every process inside the container. Never published: anything that reached it could report fake health
22	`sshd`	TCP	`SSH=1`, as `SSH_PORT` (2222)	Shell access with your SSH keys
5561–5564	`pm-stats`, `pm-audit`, `pm-clearing`, `pm-index`	ZeroMQ REP	never	Health pings answered by each process, used by `pm-*-cli health` and `pm-opctl-cli list` inside the container
5553, 5554	`pm-market-sim`	ZeroMQ	never	The market model's true values and instructor commands: kept away from students on purpose

The client-facing ports are published because something outside the container has to reach them. The bus and health ports

are not, because only the exchange's own processes use them, over the container's loopback interface. `ZMQ=1` is the deliberate exception, for when you want to attach your own tools to the bus.

4.8.2 The GUI containers

Host port (variable)	Container port	Service	Why
`TERMINAL_GUI_PORT` (8090)	8090	`terminal-gui`	TapeDeck in your browser
`LOG_GUI_PORT` (8091)	8091	`log-gui`	Log Operator Console
`CONFIG_GUI_PORT` (8092)	8092	`config-gui`	Configuration builder
`TRADER_GUI_PORT` (8093)	8093	`trader-gui`	Trading GUI
`BOOK_GUI_PORT` (8094)	8094	`book-gui`	Order book viewer
`DASH_GUI_PORT` (8095)	8095	`dash-gui`	Operator dashboard

Only the host side of a GUI port is configurable. If 8090 is already taken on your machine, set `TERMINAL_GUI_PORT=9090` in `.env` and restart; the container still listens on 8090.

`make ports` (or `podman port edumatcher`) lists what is actually published on a running container.

BIND_ADDR is the only setting that decides who can connect

With the default `127.0.0.1`, every published port is reachable from your machine only. `BIND_ADDR=0.0.0.0` opens all of them to your network, which is right for a classroom and wrong anywhere you do not trust: the order-entry gateways have no password.

4.9 Every variable, in one place

4.9.1 `deployment/docker/.env`

Variable	Default	Used by	Goes into a container?
`COMPOSE_PROJECT_NAME`	`edumatcher`	compose	no: names the project, network, pod and volumes
`EDUMATCHER_VERSION`	*(empty)*	backend build	build argument: PyPI release when no wheel is present
`WITH_SSH`	`1`	backend build	build argument: include `openssh-server`
`PIP_INDEX_URL`	*(not in `.env`)*	backend build	build argument: PyPI mirror
`EM_CONFIG`	`s10-basic`	`entrypoint.sh`	yes
`EM_CONFIG_FILE`	*(empty)*	`entrypoint.sh`	yes; set by `make up-all CONFIG=<file>`
`EM_PROFILE`	`default`	`entrypoint.sh`	yes
`TZ`	`Europe/Stockholm`	all processes	yes
`BIND_ADDR`	`127.0.0.1`	compose `ports:`	no: the host interface ports open on
`EDUMATCHER_GATEWAY_BIND_HOST`	`0.0.0.0`	gateways, `pm-log-srv`, `pm-api-gwy`, `pm-monitor`	yes
`SSH_PORT`	`2222`	`compose.ssh.yaml`, `make ssh`	no

Variable	Default	Used by	Goes into a container?
`TERMINAL_GUI_PORT` … `DASH_GUI_PORT`	`8090` … `8095`	GUI `ports:`	no
`LOG_GUI_URL` / `BOOK_GUI_URL`	`http://localhost:<port>`	`dash-gui`	yes: the drawers' links to log-gui and book-gui
`IMAGE_NAME` / `IMAGE_TAG`	`edumatcher` / `local`	compose `image:`	no
`CONTAINER_NAME`	`edumatcher`	compose, `Makefile`	no

Read from your shell when set, with no `.env` entry: `CORS_ORIGIN`, `MAX_WS_CLIENTS`, `CALF_CLIENT_ID`, `INDEX_IDS` and `LOG_SRV_ENABLED` for the GUIs.

`API_GATEWAY_URL`, `PM_TERMINAL_API_KEY`, `PM_BOOK_API_KEY` and `PM_DASH_API_KEY` are also variables of `compose.guis.yaml`, but **leave them unset**: `make up-all` resolves them from the deployed configuration at every start (see the two-phase start).

4.9.2 `~/.edumatcher/.env` (curl install)

The same idea, with three differences. `EM_VERSION` and `GHCR_OWNER` choose which released images to pull. `EM_ZMQ=1` adds the ZeroMQ overlay (there is no `make` to pass `ZMQ=1`). There are no build variables. The defaults also differ in one place: `EM_PROFILE=mm-demo` (the full stack plus a slow market-maker bot, so a fresh install has prices to look at). The full table is in Configuring it: the `.env` file.

4.9.3 `~/.edumatcher-sim/.env` (curl `--sim` install)

`EM_STACK=sim` (which tells `edumatcher.sh` to drive `compose.ai-sim.yaml`), `EM_VERSION`, `GHCR_OWNER`, `EM_SPEED`, `EM_START`, `EM_CONFIG_FILE`, `TZ`, `BIND_ADDR`, `EDUMATCHER_GATEWAY_BIND_HOST`, `PM_READONLY_API_KEY` and the four GUI ports.

4.9.4 Variables only `entrypoint.sh` reads

Variable	Set by	Effect
`EM_SSH`	`compose.ssh.yaml`	`1` starts `sshd`
`EM_SPEED`	AI-sim image and compose file	Not `1`: export `EDUMATCHER_SCHEDULER_SPEED` so the calendar runs faster
`EM_START`	AI-sim compose file	Export `EDUMATCHER_SCHEDULER_START`: where an accelerated calendar begins
`EM_RESTART_DEAD`	AI-sim image	`1`: every minute, restart any process of the profile that has died
`EDUMATCHER_DATA_DIR`	the image (`/data`)	Where everything is read and written; do not change it in a container

4.10 Using the helper scripts

4.10.1 The `Makefile` in `deployment/docker`

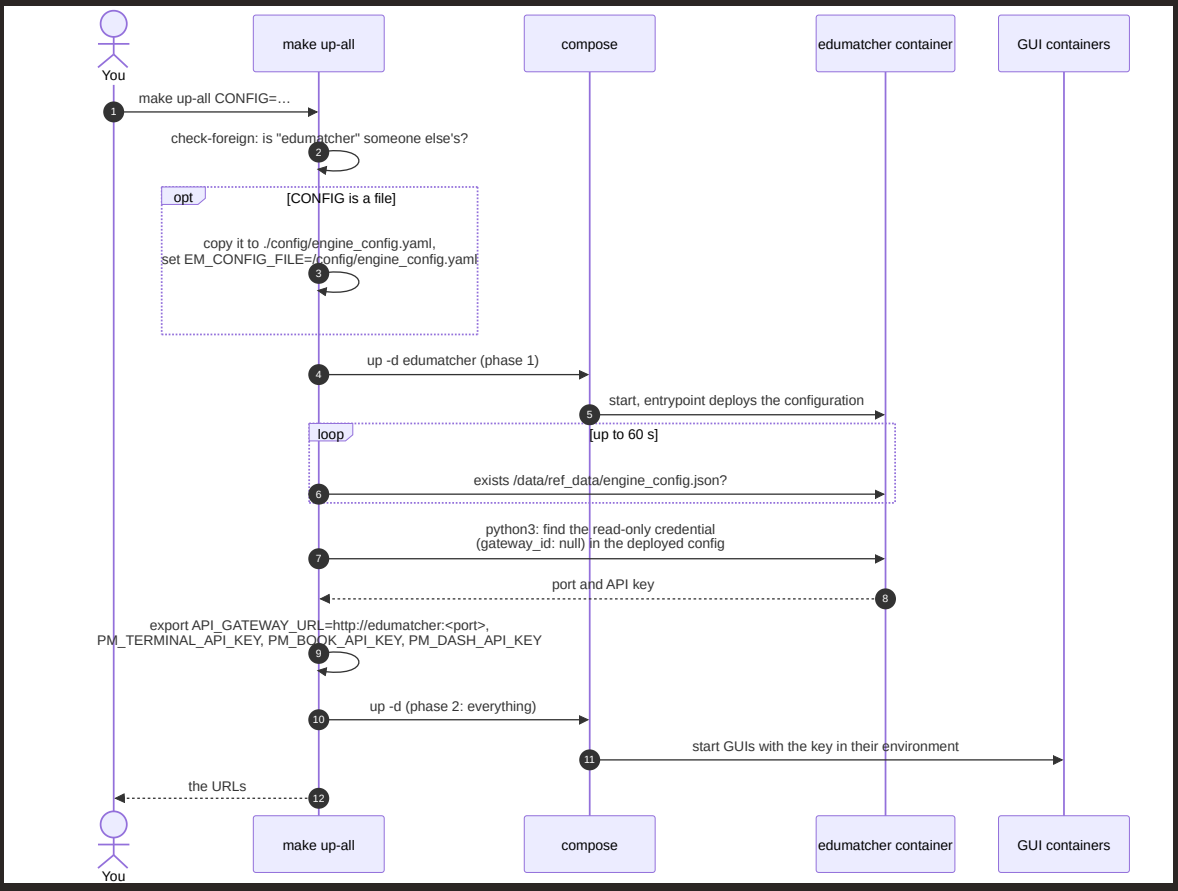
The `Makefile` first detects your tools. It uses Podman if it is installed, otherwise Docker (`CONTAINER_ENGINE=docker` forces Docker). It then uses `podman-compose`, `podman compose`, `docker compose` or `docker-compose`, whichever exists. Every target then runs ordinary compose commands with the right files.

Group	Target	What it does
Setup	`make init`	Create `.env` from `.env.example`; also happens automatically on first `build`/`up`
	`make info`	Show the detected engine, compose tool and image name
Build	`make build`	Build a wheel from your checkout with Poetry, place it in `.wheel/`, build the exchange image
	`make build PYPI=1` / `VERSION=x.y.z`	Build the exchange image from a PyPI release instead
	`make build-guis [GUI=<name>]`	Rebuild all six GUI images, or one. `log-gui` gets a wheel from your checkout (or PyPI with `PYPI=1` / `VERSION=`)
	`make build-sim`	Build `edumatcher-ai-sim:local` from your checkout
Run	`make up [ZMQ=1] [SSH=1] [CONFIG=…] [PROFILE=…]`	Start the exchange alone
	`make up-all [BUILD=1] [CONFIG_GUI=1] …`	Start the exchange and the GUIs in two phases
	`make down` / `make down-all`	Stop and remove the containers (data is kept)
	`make restart`	Restart the exchange container
	`make up-sim [LOCAL=1] [SPEED=N] [START=…]` / `make down-sim`	Start or stop the AI-trader simulation
Inspect	`make status` / `make health`	`pm-opctl-cli list` / `pm-opctl-cli health` inside the container
	`make logs` / `make proc-logs P=engine`	Follow the container log / one process log in `data/emo/`
	`make ports` / `make mounts`	Published ports / which directory backs each container path
	`make sim-status`	The simulation's process table, data size and free disk
Access	`make shell` / `make sim-shell`	A root shell in the exchange / simulation container
	`make ssh`	Log in over SSH (after `make up SSH=1`)
	`make keys`	Collect `~/.ssh/*.pub` into `.ssh/authorized_keys`
Configure	`make config-deploy`	Recompile `data/ref_data/engine_config.yaml`; then `make restart`
	`make config-show`	Show the deployed configuration
Develop	`make dev-env GUI=<name>`	Print the environment for running one GUI from source against this container
Publish	`make ghcr-push [TAG=…]`	Build every image from your checkout and push it to GHCR (single architecture; prefer the workflow)

Group	Target	What it does
Clean up	`make clean` / `make clean-data` / `make clean-all`	Remove container and image / delete `./data` (asks first) / both

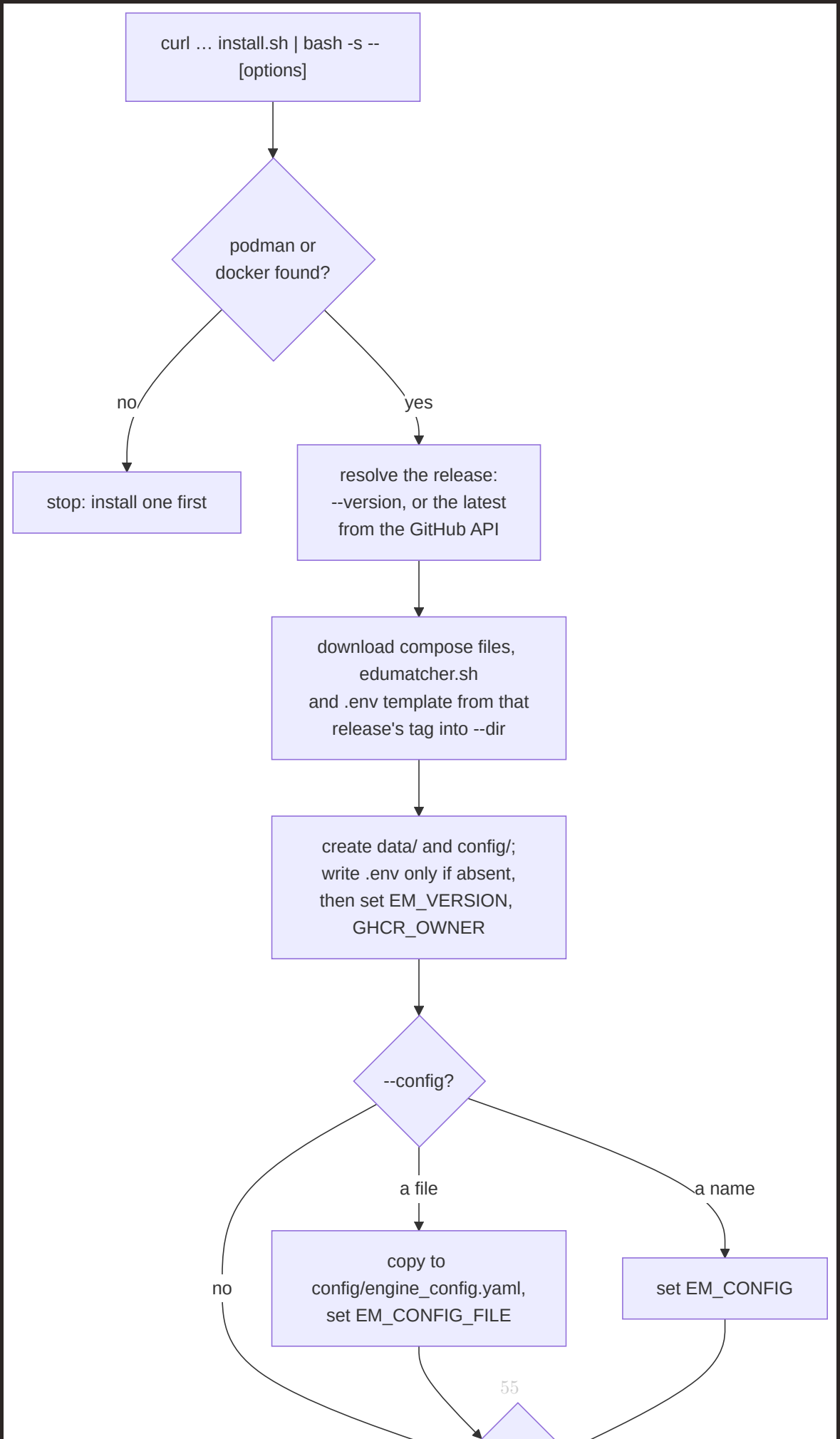
`make up-all` step by step

`up-all` cannot start everything in one go. `terminal-gui` and `book-gui` need the read-only API key, and that key is only known once the exchange has deployed its configuration: every bundled example has a different one. So it runs in two phases:



`./edumatcher.sh start` does exactly the same for a curl install. The AI-sim stack skips it, because its configuration and key are fixed.

4.10.2 `install.sh` (curl)



Re-running the installer into an existing directory is safe: it keeps your `.env` and `data/`, and refreshes the compose files and script from the release. That is how you pick up a GUI added by a newer release, which `./edumatcher.sh update` alone would not (it pulls new images for the files you already have). `--sim` installs the simulation into `~/.edumatcher-sim` instead.

4.10.3 `edumatcher.sh` (curl)

The control script reads `.env` to decide what it drives. `EM_STACK=sim` selects `compose.ai-sim.yaml`; otherwise it uses `compose.yaml`, plus `compose.zmq.yaml` when `EM_ZMQ=1`.

Command	Equivalent	Notes
`./edumatcher.sh start`	`make up-all`	Two-phase for the exchange, one phase for the simulation; refuses if another install owns the containers
`./edumatcher.sh stop`	`make down-all`	Data kept
`./edumatcher.sh restart`	stop + start	How `.env` edits take effect
`./edumatcher.sh status`	`compose ps` + `make status`	
`./edumatcher.sh logs [service]`	`compose logs -f`	
`./edumatcher.sh shell [cmd…]`	`make shell`	With a command, runs just that command
`./edumatcher.sh config <name or file>`	`CONFIG=`	Edits `.env` (and copies a file, plus a neighbouring `market_sim.yaml`); apply with `restart`
`./edumatcher.sh speed N [start]`	`make up-sim SPEED=N START=…`	Simulation only
`./edumatcher.sh upgrade [-y]` (or `--upgrade`)	—	Find the newest release, pull it, then — if the system was running, and after a warning and a question (`-y` skips it) — stop, replace the images, remove the old ones and restart. A stopped system is only upgraded, not started. A failed pull changes nothing
`./edumatcher.sh pull`	—	Download the images, start nothing
`./edumatcher.sh update [version]`	—	Set `EM_VERSION`, pull, restart, without asking
`./edumatcher.sh urls` / `mounts`	— / `make mounts`	
`./edumatcher.sh uninstall [--data]`	`compose down --volumes`	`--data` also deletes `data/` and `config/`

4.11 Doing it by hand

Everything the scripts do can be done with plain commands, which is useful when a script is not available, or to understand what it does. The examples use `podman` and `podman-compose`; replace them with `docker` and `docker compose` if you use Docker. Run them from `deployment/docker`.

4.11.1 Rebuilding the exchange image

```
cd deployment/docker

# 1. Build a wheel from the checkout and put it in the drop box
(cd ../.. && poetry build --format wheel)
rm -f .wheel/*.whl
cp "$(ls -t ../../dist/*.whl | head -1)" .wheel/

# 2. Build the runtime stage, tagged the way compose.yaml expects
podman build --target runtime -t edumatcher:local .

# Variants:
podman build --target runtime -t edumatcher:local \
    --build-arg WITH_SSH=0 .                        # no sshd
rm -f .wheel/*.whl && podman build --target runtime -t edumatcher:local \
    --build-arg EDUMATCHER_VERSION=0.50.5 .         # a PyPI release
podman build --target runtime -t edumatcher:local \
    --build-arg PIP_INDEX_URL=https://pypi.example.corp/simple .   # a mirror

# 3. The self-trading variant: same steps, other target
podman build --target ai-sim -t edumatcher-ai-sim:local .
```

Always give -target

Without it, the build produces the *last* stage, `ai-sim`, tagged with whatever name you gave it. An `edumatcher:local` that quietly starts the 300-symbol simulation is a confusing thing to debug.
Add `--no-cache` to rebuild every layer, for example after a base image security update. Otherwise unchanged layers are reused and a rebuild after a code change only redoes the `pip install` and what follows.

4.11.2 Rebuilding a GUI image

Each GUI builds from its own directory:

```
cd web-apps/terminal-gui
podman build -t edumatcher-terminal-gui:latest .

# behind a corporate proxy
podman build -t edumatcher-terminal-gui:latest \
    --build-arg HTTP_PROXY=http://proxy.example:8080 \
    --build-arg HTTPS_PROXY=http://proxy.example:8080 \
    --build-arg NPM_STRICT_SSL=false .
```

The names compose expects are `edumatcher-terminal-gui`, `edumatcher-log-gui`, `edumatcher-config-gui`, `edumatcher-trader-gui`, `edumatcher-book-gui` and `edumatcher-dash-gui`, each with the tag `latest`. From `deployment/docker`, `podman-compose -f compose.yaml -f compose.guis.yaml -f compose.config-gui.yaml build terminal-gui` does the same. This is what `make build-guis GUI=terminal-` runs.

The log console needs one extra step by hand, because it bundles the Python package. Put a wheel in its drop box first, exactly as for the exchange image:

```
(cd ../.. && poetry build --format wheel)          # from deployment/docker
rm -f ../../web-apps/log-gui/.wheel/*.whl
cp "$(ls -t ../../dist/*.whl | head -1)" ../../web-apps/log-gui/.wheel/
cd ../../web-apps/log-gui && podman build -t edumatcher-log-gui:latest .
```

With an empty drop box it installs a PyPI release instead: the newest, or the one named with `--build-arg EDUMATCHER_VERSION=x.y.z`.

A rebuilt image is not used until its container is recreated. Running `up -d` again does that for every container whose image changed.

4.11.3 Starting with compose directly

```
cd deployment/docker
cp -n .env.example .env
mkdir -p data config

# The exchange alone (make up), optionally with overlays (ZMQ=1 SSH=1)
podman-compose -f compose.yaml up -d
podman-compose -f compose.yaml -f compose.zmq.yaml up -d

# One start with another example (make up CONFIG=s10-nominal)
EM_CONFIG=s10-nominal podman-compose -f compose.yaml up -d
```

The whole system needs the two phases by hand:

```
# Phase 1: the exchange
podman-compose -f compose.yaml -f compose.guis.yaml up -d edumatcher

# Wait for it to deploy its configuration, then read the read-only key
until podman exec edumatcher test -f /data/ref_data/engine_config.json; do sleep 1; done
podman exec edumatcher python3 -c '
import json
cfg = json.load(open("/data/ref_data/engine_config.json"))
for gw in cfg.get("api_gateways", {}).values():
    for c in gw.get("credentials") or []:
        if c.get("gateway_id") is None:
            print(gw.get("port", 8080), c["api_key"]); raise SystemExit
'
# prints e.g.:  8081 key-readonly-xxxxxx

# Phase 2: everything, with the key
export API_GATEWAY_URL=http://edumatcher:8081
export PM_TERMINAL_API_KEY=key-readonly-xxxxxx
export PM_BOOK_API_KEY=$PM_TERMINAL_API_KEY
export PM_DASH_API_KEY=$PM_TERMINAL_API_KEY
podman-compose -f compose.yaml -f compose.guis.yaml up -d
```

And to stop, with the same files you started with:

```
podman-compose -f compose.yaml -f compose.guis.yaml down
```

4.11.4 The released stack by hand

A curl install is just its directory. The same commands work there with no build step, since every image comes from GHCR:

```
cd ~/.edumatcher
podman-compose -f compose.yaml pull          # fetch the images for EM_VERSION
podman-compose -f compose.yaml up -d edumatcher
# … read the key as above, export the three variables …
podman-compose -f compose.yaml up -d
```

4.11.5 Make target to raw command

`make`	Raw command (from `deployment/docker`)
`make build`	`poetry build` → copy wheel to `.wheel/` → `podman-compose -f compose.yaml build`
`make up ZMQ=1`	`podman-compose -f compose.yaml -f compose.zmq.yaml up -d`
`make down`	`podman-compose -f compose.yaml down`
`make shell`	`podman exec -it edumatcher bash`

make	Raw command (from deployment/docker)
make status	podman exec -it edumatcher pm-opctl-cli list --no-restart
make health	podman exec edumatcher pm-opctl-cli health
make logs	podman logs -f edumatcher
make ports	podman port edumatcher
make config-deploy	podman exec edumatcher pm-config-deploy /data/ref_data/engine_config.yaml
make up-sim	podman-compose -p edumatcher-sim -f compose.ai-sim.yaml up -d

4.12 Common situations

You want to	Do this
See your code change running	make build && make up-all. Check the build log says Installing local wheel
See a GUI change running	make up-all BUILD=1 GUI=<name>
Change which example runs	make up-all CONFIG=s10-nominal, or ./edumatcher.sh config s10-nominal && ./edumatcher.sh restart
Run your own configuration	make up-all CONFIG=./mine.yaml, or ./edumatcher.sh config ./mine.yaml && ./edumatcher.sh restart
Edit the deployed configuration in place	Edit data/ref_data/engine_config.yaml, then make config-deploy && make restart
Let students on your network connect	BIND_ADDR=0.0.0.0 in .env, then restart. Mind the warning above
Attach a ZeroMQ tool from your machine	make up ZMQ=1 / EM_ZMQ=1 in .env
Start over with an empty exchange	make down-all && make clean-data && make up-all
Find out whose data a container is using	make mounts / ./edumatcher.sh mounts
Keep the stack running after a reboot (Podman)	Containers with restart: unless-stopped are restarted at boot only if podman-restart.service is enabled: systemctl --user enable podman-restart.service

For problems, the troubleshooting sections of deployment/docker/README.md and of the Architecture and Developer Guide's Container and Network Setup go deeper, including how to test a connection from inside a GUI container.

CHAPTER 5

The Configuration File, Block by Block

Learning objectives

After reading this chapter you will understand:

- how `engine_config.yaml` is organised into top-level blocks, and which process reads which block
- the smallest file that gives you a working exchange, and what each later block adds to it
- every parameter of every block: its type, its default, and what it changes at runtime
- which mistakes the tools catch for you, and which they do not

`engine_config.yaml` has grown into a large file. It describes the instruments, who may trade, when the exchange is open, how prices are protected, how liquidity is seeded, and how every gateway and service listens. Read top to bottom it is easy to lose the thread, so this chapter does two things:

1. It shows the **shape** of the file first — a map of the top-level blocks and what each one is for.
2. It then **builds a complete file in nine steps**. Step 1 is the smallest file the engine accepts; every later step adds one concern, explains every parameter of it, and ends with a file that still passes `pm-cverifier`. Step 9 is a full specification you can use as a template.

Every YAML listing in this chapter was verified: each step's cumulative file was run through `pm-cverifier`, and the final file was compiled with `pm-config-deploy` and started on a real `pm-engine`.

Two ways to reach the same file

You do not have to type the file by hand. `pm-config-gen` generates it from command-line options and the Configuration GUI builds it in a form. Both write the structure described here. Knowing the structure still matters: you will read, review and hand-edit generated files.

5.1 The shape of the file

`engine_config.yaml` is a single YAML mapping. Each top-level key is a **block**. Only two blocks are required; every other block is optional and has built-in defaults.

```
engine_config.yaml
│
├── CORE — who and what ──────────────────────────────────────
│   ├── symbols                   required   the instruments that can trade
│   ├── participants              required   the gateway IDs that may log in
│   └── participant_defaults                 defaults inherited by participants
│
├── TRADING DAY — when ───────────────────────────────────────
│   ├── sessions_enabled                     scheduler-driven phases on/off
│   ├── schedule                             the five phase times, per weekday
│   └── locale                               country (holidays) and timezone
│
├── PRICE PROTECTION — how prices are guarded ────────────────
│   ├── enforce_collars                      global collar switch
│   ├── enforce_circuit_breakers             global circuit-breaker switch
│   ├── risk_controls                        named collar levels
│   └── circuit_breaker_defaults             the halt ladder and reopening rules
│
├── LIQUIDITY — market makers and seeds ──────────────────────
│   ├── mm_obligation_defaults               quote-quality policy
│   ├── require_mm_seed_quotes               every symbol needs a seed quote?
│   └── market_maker_combos                  startup multi-leg combo orders
│
├── ENGINE TUNING ────────────────────────────────────────────
│   ├── engine_tuning                        memory and snapshot knobs
│   └── auction_indicative_interval_sec      indicative-price refresh rate
│
├── DERIVED DATA ─────────────────────────────────────────────
│   └── indices                              market indices (pm-index)
│
├── ACCESS POINTS — how the outside world connects ───────────
│   ├── alf_gateway                          pm-alf-gwy   text order entry
│   ├── balf_gateway                         pm-balf-gwy  binary order entry
│   ├── market_data_gateway                  pm-md-gwy    market data (CALF)
│   ├── post_trade_gateway                   pm-ralf-gwy  post-trade (RALF)
│   ├── dc_gateway                           pm-dc-gwy    drop copy
│   └── api_gateways                         pm-api-gwy   REST / WebSocket
│
└── OPERATIONS — watching the exchange ───────────────────────
    ├── log_server                           pm-log-srv   central log collector
    └── monitor                              pm-monitor   health beacons
```

The same map as a table, with the process that reads each block and the step of this chapter that introduces it:

Block	Required	Read by	Step
`include`	no	every tool that reads the authored file	Splitting the file
`symbols`	**yes**	`pm-engine`, `pm-index`	1, 2, 4, 5
`participants`	**yes**	`pm-engine`, `pm-alf-gwy`, `pm-balf-gwy`	1, 5
`participant_defaults`	no	`pm-engine`	5
`sessions_enabled`	no	`pm-engine`	1, 3
`schedule`	no	`pm-scheduler`, `pm-engine`	3
`locale`	no	`pm-scheduler`, `pm-stats`, `pm-clearing`, `pm-monitor`	3
`enforce_collars`, `enforce_circuit_breakers`	no	`pm-engine`	4

Block	Required	Read by	Step
`risk_controls`	no	`pm-engine`	4
`circuit_breaker_defaults`	no	`pm-engine`	4
`mm_obligation_defaults`	no	`pm-engine`	5
`require_mm_seed_quotes`	no	`pm-engine`	5
`market_maker_combos`	no	`pm-engine`	5
`engine_tuning`, `auction_indicative_interval_sec`	no	`pm-engine`	6
`indices`	no	`pm-index`	7
`alf_gateway`	no	`pm-alf-gwy`	8
`balf_gateway`	no	`pm-balf-gwy`	8
`market_data_gateway`	no	`pm-md-gwy`	8
`post_trade_gateway`	no	`pm-ralf-gwy`	8
`dc_gateway`	no	`pm-dc-gwy`	8
`api_gateways`	no	`pm-api-gwy`	8
`log_server`	no	`pm-log-srv`	9
`monitor`	no	`pm-monitor` and every process that sends a health beacon	9

Three facts about the file explain most of the surprises operators meet:

- **No process reads the YAML you edit.** `pm-config-deploy` validates it and compiles it into `<EDUMATCHER_DATA_DIR>/ref_data/` and that compiled artifact is the only thing the running processes load. Edit the YAML, then redeploy, then restart. See The Configuration Workflow.
- **Each process reads only its own blocks.** `pm-engine` never looks at `alf_gateway`; `pm-alf-gwy` never looks at `risk_controls`. A mistake in a gateway block cannot be seen by the engine — `pm-cverifier` is what looks at the whole file.
- **A misspelled key is an error.** The loaders themselves ignore a key they do not know, so `hartbeat_interval_sec` would silently leave the default in force. `pm-cverifier` closes that gap: `S121` reports any key that no loader reads, at any level, and names the closest accepted spelling. The verifier runs inside `pm-config-deploy`, so an unrecognised key stops a deploy.

5.1.1 Conventions used by every block

Convention	Rule
Prices	Written as display money (`209.50`), never as ticks. A price the symbol's tick grid cannot represent (`209.505` at `tick_decimals: 2`) is a config error, not a silent rounding.
IDs and names	Symbols, participant IDs, level names and index IDs are upper-cased by the loader. Write them upper-case anyway.
Durations	Suffix tells the unit: `_sec`, `_ms`, `_days`, `_hours` are what they say. Circuit-breaker fields end in `_ns` — **nanoseconds**: `300000000000` is five minutes.
Fractions	Percentages are fractions: `0.20` means 20 %.
Wall-clock times	`"HH:MM"` strings, 24-hour, in `locale.session_timezone`. Quote them; an unquoted `9:30` is read by YAML as a number of minutes.

Convention	Rule
`null`	An empty value (`halt_duration_ns:`) is `null`. Where a field allows it, `null` means "no limit".
Bind addresses	`bind_address` / `host` default to `0.0.0.0` (all interfaces). The environment variable `EDUMATCHER_GATEWAY_BIND_HOST`, when set, overrides every one of them.
Ports	Every listener in the file must use a different port — see Ports at a glance. `pm-cverifier` reports a clash as `M018`.

5.2 Splitting the file with `include`

A class with hundreds of symbols and participants makes one file hard to read and hard to compare between runs. YAML has no include of its own, so EduMatcher defines a small one: the top-level key `include` lists other files whose contents are merged into the configuration.

```
include:
  - symbols.yaml
  - participants.yaml

sessions_enabled: true
locale: {country: SE, session_timezone: Europe/Stockholm}
```

```
# symbols.yaml
symbols:
  AAPL: {tick_decimals: 2, last_buy_price: 150.00}
  MSFT: {tick_decimals: 2, last_buy_price: 400.00}
```

The rules are deliberately few:

Rule	Detail
Where	Paths are relative to the file that lists them. No absolute paths and no `..`; a subdirectory (`parts/symbols.yaml`) is fine.
Names	Each file may be listed once. Beside the main file, `engine_config.yaml`, `engine_config.json` and `market_sim.yaml` are reserved, because `pm-config-deploy` installs its own files under those names; put such a file in a subdirectory or rename it. The same applies to a `credentials_file`.
Depth	One level. An included file may not have an `include` of its own; list every file in the main file.
Content	An included file is a YAML mapping using the same top-level keys as the main file.
Lists	A list present in several files (`participants`) is concatenated: the main file's entries first, then each include in the order listed.
Mappings	A mapping present in several files (`symbols`) is merged. The same symbol defined twice is an error.
Anything else	A scalar or a block defined twice, or a list in one file and a mapping in another, is an error that names both files. Nothing is silently overridden.

Everything that reads the file sees the merged result, so `pm-cverifier`, `pm-config-show` and `pm-config-deploy` behave exactly as if you had written one file. Three things differ in practice:

- **Findings name the file.** A verifier finding about `symbols.AAPL` that comes from an included file is followed by `in symbols.yaml`, and `pm-config-deploy` shows it as `[symbols.yaml: symbols.AAPL.tick_decimals]`. A broken or missing include is `Y005`.
- **Deploy treats the files as one unit.** The staleness check covers every file, so editing only `symbols.yaml` without redeploying is reported, just as for the main file. `pm-config-deploy` copies the main file and its includes beside the artifact, and removes copies of includes you have since dropped.
- **`pm-new-symbol` edits the include.** A symbol is added to the file that supplies the last listed symbol, and the main file is left untouched.

`pm-config-gen --split --output engine_config.yaml` writes a configuration in this layout from the start: `symbols.yaml` and `participants.yaml` beside the main file.

5.3 Step 1: the smallest working file

Two blocks are required: **`symbols`** (what can be traded) and **`participants`** (who may trade). Add `sessions_enabled: false` and you have a file that starts an exchange which is open for continuous trading immediately.

```
sessions_enabled: false

symbols:
  AAPL:

participants:
  - id: TRADER01
  - id: TRADER02
  - id: OPS01
    role: ADMIN
```

This is a complete, valid `engine_config.yaml`. It lists one instrument, `AAPL`, and three gateway IDs: two ordinary traders and one `ADMIN` for the operator. Everything else is default.

Always decide sessions_enabled explicitly

When a config file exists and `sessions_enabled` is left out, the engine behaves as if it were `true`: it starts in the `CLOSED` state and waits for `pm-scheduler` to open the market. If you did not mean to run a scheduler, your students will log in to a closed exchange. (Only when there is *no* config file at all does the engine default to continuous trading.)
Run the verifier on it:

```
pm-cverifier engine_config.yaml
```

```
pm-cverifier engine_config.yaml

✓ YAML syntax: OK
✓ Schema: 0 errors
⚠ Semantic: 1 warning
⚠ Completeness: 5 advisories

[M032] WARN  No read-only API credential (gateway_id: null) is defined under api_gateways.
[C001] WARN  No symbol has a reference price (last_buy_price / last_sell_price).
[C003] INFO  enforce_collars is true but no symbol has an effective collar ...
[C004] INFO  enforce_circuit_breakers is true but no circuit breaker is configured ...
[C007] INFO  snapshot_interval_sec is at the default of 0.5 seconds.
[C009] INFO  sessions_enabled is false and no schedule is configured. ...

Verdict:  ⚠ 2 WARNINGS, 4 ADVISORIES — engine can start but review warnings
```

(The full report also prints each suggestion and a *Risk Summary*; they are cut here for space.) Zero errors means the engine can start. The warnings and advisories are the to-do list for the steps that follow — by Step 8 they are all gone.

5.3.1 `symbols`

A mapping from symbol name to its settings. A symbol with no settings may be written as a bare key, as above.

Field	Type	Default	Meaning
`tick_decimals`	integer `0`–`8`	`2`	Number of decimal places in the symbol's prices. The tick size is `10^-tick_decimals`: at `2`, prices move in steps of `0.01`.
`last_buy_price`, `last_sell_price`	number	none	Opening reference prices. Added in Step 2.
`outstanding_shares`	positive integer	none	Issued share count. Added in Step 2.
`level`	name from `risk_controls.levels`	`risk_controls.default_level`	Collar profile. Added in Step 4.
`collar`	mapping	none	Per-symbol collar override. Step 4.
`circuit_breaker`	mapping	none	Per-symbol circuit-breaker override. Step 4.
`order_limits`	mapping	none	Per-order size and value caps. Step 4.
`market_maker_quotes`	list	none	Startup quotes. Required once a market maker exists. Step 5.

The symbol universe is fixed when the engine starts. To list a new symbol, edit the file, redeploy and restart — see Adding or Removing Symbols and Listing a New Symbol.

5.3.2 `participants`

A list of the gateway IDs that may log in. Only an ID in this list can submit an order; anything else is refused.

Field	Type	Default	Meaning
`id`	string, **required**	—	The gateway ID a client presents at login. Unique. No ID may be a prefix of another (`TRADER01` and `TRADER010` cannot coexist) — `pm-alf-gwy` and `pm-balf-gwy` refuse to start.
`description`	string	empty	Free-text label shown in tools.
`role`	`TRADER`, `MARKET_MAKER`, `ADMIN`	`TRADER`	What the participant may do (see below).
`disconnect_behaviour`	`CANCEL_QUOTES_ONLY`, `CANCEL_ALL`, `LEAVE_ALL`	`participant_defaults`, else `CANCEL_QUOTES_ONLY`	What the engine does with the participant's resting interest when it disconnects.

The three roles:

Role	Orders	Quotes	Circuit-breaker halt / resume	Typical user
`TRADER`	yes	no	no	Students, manual participants, AI traders
`MARKET_MAKER`	yes	yes	no	Liquidity providers
`ADMIN`	yes	no	yes	The operator or instructor console

The three disconnect behaviours:

Value	On disconnect the engine…
`CANCEL_QUOTES_ONLY`	cancels active quotes and any resting remainder of an inactivated partial-fill quote; ordinary resting orders stay in the book.
`CANCEL_ALL`	cancels active quotes, inactivated partial-fill quote remainders, and all other resting orders. A student who closes the laptop leaves nothing behind.
`LEAVE_ALL`	cancels nothing. Use it for the operator console, whose connection should not disturb the book.

The market-maker-only fields of a participant (`quote_refresh_policy`, `smp_action`, `enforce_mm_obligation`, `mm_max_spread_ticks`, `mm_min_qty`, `mm_obligations`) arrive in Step 5.

5.3.3 `sessions_enabled`

Value	Effect
`false`	The engine starts in `CONTINUOUS` and ignores scheduler transitions.
`true` (the default when a config file exists)	The engine starts `CLOSED`; `pm-scheduler` drives it through the trading day.

5.4 Step 2: reference data

A bare `AAPL:` gives the engine no idea what the stock costs, so collars, circuit breakers and index calculations have nothing to anchor to. Step 2 lists each symbol the way an **initial listing** would: its price grid, its opening reference prices, and its issued shares. It also fills in the descriptions and disconnect behaviour of the participants.

```
symbols:
  AAPL:
    tick_decimals: 2
    last_buy_price: 209.50
    last_sell_price: 210.50
    outstanding_shares: 15400000000
  MSFT:
    tick_decimals: 2
    last_buy_price: 415.00
    last_sell_price: 415.50
    outstanding_shares: 7430000000
  TSLA:
    tick_decimals: 2
    last_buy_price: 248.00
    last_sell_price: 249.00
    outstanding_shares: 3200000000

participants:
  - id: TRADER01
    description: Student workstation 1
    role: TRADER
    disconnect_behaviour: CANCEL_ALL
  - id: TRADER02
    description: Student workstation 2
    role: TRADER
    disconnect_behaviour: CANCEL_ALL
  - id: OPS01
    description: Instructor console
    role: ADMIN
    disconnect_behaviour: LEAVE_ALL
```

Field	Meaning
`tick_decimals: 2`	Prices have two decimals; the tick is `0.01`. Use `0` for whole-unit prices, `4` for a currency quoted to four places.
`last_buy_price`, `last_sell_price`	The opening bid-side and ask-side reference prices, in display money. They seed the book's last prices and the *reference* for collars and circuit breakers on the first order of day one.
`outstanding_shares`	Issued share count. Required for any symbol that is an index constituent (Step 7); used for market capitalisation.

Persisted prices win

After a first trading day the engine writes `book_stats.json` to the data directory and, on the next start, restores the last prices from it. A restored price **replaces** `last_buy_price` / `last_sell_price` from the file. The values in the YAML only matter on the first start, or after you delete `book_stats.json`. See Persistence.
The collar reference is chosen in this order: persisted `last_buy_price`, persisted `last_sell_price`, then the YAML `last_buy_price`, then the YAML `last_sell_price`. A symbol with none of the four has no collar active at all.

5.5 Step 3: the trading day

Until now the exchange was always open. Step 3 turns on **sessions**: the engine starts `CLOSED`, and `pm-scheduler` moves it through the phases of the day at the times you give.

```
sessions_enabled: true

schedule:
  weekdays:
    pre_open: "09:00"
    opening_auction_start: "09:25"
    continuous_start: "09:30"
    closing_auction_start: "16:00"
    closing_auction_end: "16:05"

locale:
  country: Sweden
  session_timezone: Europe/Stockholm
```

The phases in order are:

```
CLOSED → PRE_OPEN → OPENING_AUCTION → CONTINUOUS → CLOSING_AUCTION → CLOSED
```

5.5.1 `schedule`

Each block you write must contain **all five** times; there is no per-key fallback.

Key	Meaning
`pre_open`	`PRE_OPEN`: orders are accepted and rest on the book; nothing executes.
`opening_auction_start`	`OPENING_AUCTION`: the opening call auction collects orders; nothing executes.
`continuous_start`	The opening auction uncrosses at one equilibrium price; `CONTINUOUS` matching begins.
`closing_auction_start`	Continuous matching ends; `CLOSING_AUCTION` collects orders.
`closing_auction_end`	The closing auction uncrosses; the exchange goes `CLOSED` and resting `DAY` orders expire.

Times are `"HH:MM"` in the timezone given by `locale.session_timezone` (below) — **not** the clock of the machine the scheduler happens to run on.

The block names say which days a block applies to:

Block	Applies to
`weekdays`	Monday–Friday, for any day not given its own block.
`mon … fri`	One weekday; overrides `weekdays` for that day.
`weekend`	Saturday and Sunday. Cannot be combined with `sat` / `sun`.
`sat`, `sun`	One weekend day.
`holidays`	Bank holidays of `locale.country`.

A day with no block is `CLOSED` all day. See Session Scheduling and Auctions for worked examples, including how a closed day is skipped.

5.5.2 `locale`

Field	Type	Default	Meaning
`country`	country name (`"Sweden"`) or ISO 3166-1 alpha-2 code (`"SE"`)	`Sweden`	Which country's bank-holiday calendar `pm-scheduler` uses to decide when `holidays` applies. An unrecognised value falls back to `Sweden` with a logged warning.
`session_timezone`	`UTC` or an IANA timezone name	`Europe/Stockholm`	Which calendar date a timestamp belongs to. `pm-stats` and `pm-clearing` use it to cut events into trading days. An unknown name is rejected.

Do not change session_timezone on a running exchange

`pm-stats` and `pm-clearing` remember the timezone they first ran with and refuse to start if the configuration later disagrees, because re-cutting existing history would silently change figures already published.

5.6 Step 4: price protection

Two independent mechanisms guard prices:

- A **collar** rejects an *incoming order* whose limit price is too far from a reference — a fat finger is stopped at the door.
- A **circuit breaker** *halts a symbol* when trades move the price too far, too fast, and reopens it through an auction.

The full behaviour is in Risk Controls. Step 4 adds the global switches, named collar levels, the halt ladder, and per-symbol exceptions.

```
enforce_collars: true

enforce_circuit_breakers: true

risk_controls:
  default_level: L2
  levels:
    L1:
      collar:
        static_band_pct: 0.30
        dynamic_band_pct: 0.05
    L2:
      collar:
        static_band_pct: 0.20
        dynamic_band_pct: 0.02
    L3:
      collar:
        static_band_pct: 0.12
        dynamic_band_pct: 0.01

circuit_breaker_defaults:
  reference_window_ns: 300000000000     # 5 minutes
  levels:
    L1:
      price_shift_pct: 0.07
      halt_duration_ns: 300000000000    # 5 minutes
    L2:
      price_shift_pct: 0.13
      halt_duration_ns: 900000000000    # 15 minutes
    L3:
      price_shift_pct: 0.20
      halt_duration_ns:                 # null = halted for the rest of the day
  reopening:
    enabled: true
    initial_band_pct: 0.10
    random_end_max_ns: 30000000000      # up to 30 s of random extension
    random_seed:                        # null = seed from OS entropy
    expansions:
      - widen_pct: 0.10
        min_duration_ns: 120000000000
      - widen_pct: 0.20
        min_duration_ns: 300000000000

symbols:
  AAPL:
    tick_decimals: 2
    last_buy_price: 209.50
    last_sell_price: 210.50
    outstanding_shares: 15400000000
    order_limits:
      max_order_qty: 10000
      max_order_value: 2000000
    collar:
      dynamic_band_pct: 0.015
  MSFT:
    tick_decimals: 2
    level: L1
    last_buy_price: 415.00
    last_sell_price: 415.50
    outstanding_shares: 7430000000
  TSLA:
    tick_decimals: 2
    level: L3
    last_buy_price: 248.00
    last_sell_price: 249.00
    outstanding_shares: 3200000000
```

Both mechanisms are opt-in per symbol

`enforce_collars: true` does nothing for a symbol that has no collar, and `enforce_circuit_breakers: true` does nothing for a symbol that has no circuit breaker. A symbol has a collar only if it gets one from its own `collar`, its `level`, or `risk_controls.default_level`. It has a circuit breaker only if `circuit_breaker_defaults` or its own `circuit_breaker` block exists. In Step 1's file neither exists, so no symbol was protected.

5.6.1 The global switches

Field	Default	Meaning
`enforce_collars`	`true`	Master switch for collar checks. Set `false` only for tests.
`enforce_circuit_breakers`	`true`	Master switch for halts. Set `false` only for tests.

5.6.2 `risk_controls`

Named, reusable collar profiles.

Field	Type	Default	Meaning
`default_level`	a key of `levels`	none	The level every symbol gets unless it names its own. Must exist in `levels`.
`levels.<NAME>.collar.static_band_pct`	number in `(0, 1)`	`0.20`	Allowed distance from the **reference price** (see Step 2). `0.20` allows prices within ±20 %.
`levels.<NAME>.collar.dynamic_band_pct`	number in `(0, 1)`	`0.02`	Allowed distance from the **latest trade price**. `0.02` allows ±2 % of the last fill.

A level carries only a `collar`. Putting `order_limits` or `circuit_breaker` inside a level is rejected with an error that names the right place.

5.6.3 `circuit_breaker_defaults`

The exchange-wide halt ladder. A symbol's own `circuit_breaker` block is merged over it, field by field.

Field	Type	Default	Meaning
`reference_window_ns`	integer ns	`300000000000` (5 min)	How far back the breaker looks for the price it measures the move against.
`levels.<NAME>.price_shift_pct`	number in `(0, 1)`, **required**	—	The move, as a fraction of the reference, that triggers this rung. Rungs are sorted by it, so name them as you like. Level names are limited to 32 characters.

Field	Type	Default	Meaning
`levels.<NAME>.halt_duration_ns`	positive integer ns, or `null`	`null`	The *minimum* length of the halt's reopening call phase (the random tail below is added on top). `null` means rest-of-day: the symbol never reopens on a timer — it stays halted until an `ADMIN` resumes it or the exchange closes for the day, when halts are reset.

If a circuit-breaker block is present but has no `levels`, the built-in ladder is used: L1 7 % / 5 min, L2 13 % / 15 min, L3 20 % / until resumed. If *no* circuit-breaker block exists anywhere, the symbol has no breaker.

A halt is a **reopening auction**: orders keep arriving and resting, nothing matches, and the halt ends with an uncross. The `reopening` block controls how that auction ends:

Field	Type	Default	Meaning
`reopening.enabled`	boolean	`true`	`false` makes the reopening uncross at whatever price the auction finds, with no corridor.
`reopening.initial_band_pct`	number in `(0, 1)`	`0.10`	The corridor around the reference price within which the symbol may reopen.
`reopening.expansions`	non-empty list	two rungs, shown below	When the indicative price is outside the corridor, the call is extended and the corridor widened by one rung. The last rung repeats. *Exchange-wide only* — not allowed per symbol.
`reopening.expansions[].widen_pct`	number in `(0, 1)`	required	Added to the corridor half-width, as a fraction of the reference price.
`reopening.expansions[].min_duration_ns`	positive integer ns	required	Minimum length of that extended call.
`reopening.random_end_max_ns`	integer ns `>= 0`	`30000000000` (30 s)	Each call ends at a uniformly random moment within this tail after its minimum, so the uncross instant cannot be targeted.
`reopening.random_seed`	integer or `null`	`null`	Seed for that randomness. `null` draws from OS entropy; set an integer for reproducible demos. *Exchange-wide only.*

The built-in expansions are `10 % / 2 min` then `20 % / 5 min`. A symbol may override `reopening.enabled`, `initial_band_pct` and `random_end_max_ns`, but not `expansions` or `random_seed`: those are venue policy, not instrument traits.

5.6.4 Per-symbol exceptions

The step's `symbols` block shows each kind of exception:

In the example	What it does
`MSFT: level: L1`	Uses the wide L1 collar instead of the default L2.
`TSLA: level: L3`	Uses the tight L3 collar.
`AAPL: collar: dynamic_band_pct: 0.015`	Overrides one collar field; `static_band_pct` still comes from the level.
`TSLA: circuit_breaker:`	Widens the reference window to 10 minutes, lengthens the L1 and L2 halts, and starts the reopening corridor at 15 %. L3 and everything else are inherited.
`AAPL: order_limits:`	Caps a single order.

Precedence for a collar field, most specific first: the symbol's `collar`, the symbol's `level`, `risk_controls.default_level`, then the built-in `0.20` / `0.02`.

`order_limits` (symbol level only — there is no level or global default):

Field	Type	Default	Meaning
`max_order_qty`	positive integer	none	Largest quantity one order may carry. Catches the extra-zero fat finger the collar cannot see.
`max_order_value`	positive number	none	Largest notional (`quantity × price`) in display money. Skipped for `MARKET` and `IOC` orders, which carry no price.

An absent cap is not enforced.

5.7 Step 5: market makers and seeded liquidity

An empty order book teaches little. A **market maker** posts a two-sided quote and keeps the book alive. Step 5 adds two market-maker participants, seeds every book with their opening quotes, sets the quality bar they must meet, and adds one startup combo order.

```
participant_defaults:
  disconnect_behaviour: CANCEL_QUOTES_ONLY
  smp_action: NONE

participants:
  - id: TRADER01
    description: Student workstation 1
    role: TRADER
    disconnect_behaviour: CANCEL_ALL
  - id: TRADER02
    description: Student workstation 2
    role: TRADER
    disconnect_behaviour: CANCEL_ALL
  - id: MM01
    description: Primary market maker
    role: MARKET_MAKER
    disconnect_behaviour: CANCEL_QUOTES_ONLY
    quote_refresh_policy: INACTIVATE_ON_ANY_FILL
    smp_action: NONE
    enforce_mm_obligation: true
    mm_max_spread_ticks: 20
    mm_min_qty: 100
    mm_obligations:
      AAPL:
        enforce_mm_obligation: true
        max_spread_ticks: 10
        min_qty: 300
  - id: MM02
    description: Backup market maker
    role: MARKET_MAKER
    quote_refresh_policy: INACTIVATE_ON_FULL_FILL
  - id: OPS01
    description: Instructor console
    role: ADMIN
    disconnect_behaviour: LEAVE_ALL

mm_obligation_defaults:
  enforce_mm_obligation: true
  mm_max_spread_ticks: 20
  mm_min_qty: 100
  symbols:
    AAPL:
      mm_max_spread_ticks: 10
      mm_min_qty: 200

require_mm_seed_quotes: true

symbols:
  AAPL:
    # ... fields from Steps 2 and 4 unchanged ...
    market_maker_quotes:
      - gateway_id: MM01
        quote_id: SEED-MM01-AAPL
        bid_price: 209.45
        ask_price: 209.55
        bid_qty: 2000
        ask_qty: 2000
        tif: DAY
        seed_once: true
      - gateway_id: MM02
        quote_id: SEED-MM02-AAPL
        bid_price: 209.40
        ask_price: 209.50
        bid_qty: 1000
        ask_qty: 1000
  MSFT:
```

5.7.1 The rule you will trip over first

As soon as one participant has `role: MARKET_MAKER`, **every symbol must have at least one entry in `market_maker_quotes`**, or the config is rejected:

```
Symbol 'MSFT': at least one market_maker_quotes entry is required when
MARKET_MAKER gateways are configured (set require_mm_seed_quotes: false to
allow an empty book)
```

Set `require_mm_seed_quotes: false` to allow market makers with empty books at startup. The default is `true`.

5.7.2 `participant_defaults`

Values a participant inherits when it omits them. Only these two keys are accepted.

Field	Default	Meaning
`disconnect_behaviour`	`CANCEL_QUOTES_ONLY`	Inherited by participants that do not set their own.
`smp_action`	`NONE`	Inherited self-match-prevention action (below).

5.7.3 More `participants` fields

Field	Type	Default	Meaning
`quote_refresh_policy`	`INACTIVATE_ON_ANY_FILL`, `INACTIVATE_ON_FULL_FILL`, `NEVER_INACTIVATE`	`INACTIVATE_ON_ANY_FILL`	What a fill does to a two-sided quote. `ANY_FILL`: the other leg is cancelled after any fill and the quote is inactive until re-sent. `FULL_FILL`: only a completely filled leg does that. `NEVER_INACTIVATE`: the quote stays; a fill just reduces its quantity.
`smp_action`	`NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	`participant_defaults`, else `NONE`	Self-match prevention for this participant's own orders: what to cancel when an order would trade against the same participant's resting order. It is a *fallback*: an explicit `SMP=` on an order always wins; quote legs always use this value.
`enforce_mm_obligation`	boolean	`mm_obligation_defaults`, else `false`	Whether this participant's quotes are checked against the quality bar.
`mm_max_spread_ticks`	positive integer	`mm_obligation_defaults`, else `20`	Widest allowed bid–ask spread, in ticks.
`mm_min_qty`	positive integer	`mm_obligation_defaults`, else `100`	Smallest allowed bid and ask quantity.

Field	Type	Default	Meaning
`mm_obligations.<SYMBOL>`	mapping	none	Per-symbol exception, with fields `enforce_mm_obligation`, `max_spread_ticks`, `min_qty`. Note the shorter names here: **no `mm_` prefix**. Unset fields fall back to the participant's flat fields.

5.7.4 `mm_obligation_defaults`

The quote-quality policy every market maker inherits.

Field	Type	Default	Meaning
`enforce_mm_obligation`	boolean	`false`	Master switch for the checks.
`mm_max_spread_ticks`	positive integer	`20`	Widest allowed spread, in ticks.
`mm_min_qty`	positive integer	`100`	Smallest allowed quantity on each side.
`symbols.<SYMBOL>`	mapping	none	Per-symbol override with the same three fields. The symbol must exist.

For one participant and one symbol, the effective policy is taken from the most specific source that sets it:

1. `participants[].mm_obligations.<SYMBOL>`
2. `mm_obligation_defaults.symbols.<SYMBOL>`
3. the participant's flat fields
4. `mm_obligation_defaults` flat fields
5. the built-in defaults

In the example, AAPL is held to 10 ticks by `mm_obligation_defaults.symbols` and MM01 is further held to a 300-share minimum there; every other symbol uses 20 ticks and 100 shares.

Make your seeds meet your own bar

A seed quote is a real quote. Keep each one inside its effective `mm_max_spread_ticks` and `mm_min_qty`; the spread in ticks is `(ask − bid) ÷ tick size`. `pm-cverifier` warns about a wide seed (`M003`), comparing it with the limit that applies to that gateway on that symbol.

5.7.5 `symbols.<SYMBOL>.market_maker_quotes`

Field	Type	Default	Meaning
`gateway_id`	string, **required**	—	The `MARKET_MAKER` participant that owns the quote.
`quote_id`	string	generated (`SEED-<ID>-<SYMBOL>-<n>`)	An explicit label.
`bid_price`, `ask_price`	number, **required**	—	Display prices on the symbol's tick grid; `bid_price` must be below `ask_price`.

Field	Type	Default	Meaning
`bid_qty`, `ask_qty`	positive integer, **required**	—	Quantity on each side.
`tif`	`DAY`, `GTC`	`DAY`	Time in force.
`seed_once`	boolean	`true`	`true`: skip this seed if the gateway already has a live quote on the symbol restored from the previous run, so a restart does not replace it. A quote that was fully traded away or expired is not restored, so the seed fires again. `false`: always inject, replacing any restored quote.

Live quote legs are saved with the resting orders and restored on restart; that is why `seed_once` exists. See Persistence.

5.7.6 `market_maker_combos`

Startup **combo** orders — several legs across symbols that fill all-or-nothing (`AON`). Useful to show students that multi-leg interest exists.

Field	Type	Default	Meaning
`combo_id`	string, **required**	—	Unique label.
`combo_type`	`AON`	`AON`	The only type.
`tif`	`DAY`, `GTC`, `ATO`, `ATC`	`DAY`	Time in force. Prefer `DAY`: a `GTC` seed can duplicate itself against the restored `gtc_combos.json`.
`legs`	list of 2–10	**required**	The legs; each symbol at most once, and each must exist in `symbols`.
`legs[].symbol`	string	required	
`legs[].side`	`BUY`, `SELL`	required	
`legs[].order_type`	`MARKET`, `LIMIT`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	required	
`legs[].quantity`	integer	required	
`legs[].price`, `legs[].stop_price`	number	needed by priced / stop order types	**Display money**, validated on that leg's symbol's tick grid.
`legs[].smp_action`	as above	the seeding gateway's, else `NONE`	

5.8 Step 6: engine tuning

The defaults suit almost every deployment; change them when you have measured a reason to.

```
auction_indicative_interval_sec: 1.0

engine_tuning:
  snapshot_interval_sec: 0.5
  quote_history_maxlen: 30
  drop_copy_buffer_size: 10000
  recent_trades_maxlen: 20
  depth_snapshot_tolerance_ticks: 100
```

Field	Type	Default	Meaning
`auction_indicative_interval_sec`	number > 0	`1.0`	Minimum time between republications of the indicative uncross price during a call phase (`PRE_OPEN`, `PRE_CLOSE`, or a halt).
`engine_tuning.snapshot_interval_sec`	number > 0	`0.5`	Minimum time between order-book snapshots for a symbol whose book changed. Lower: fresher viewers, more CPU and traffic.
`engine_tuning.quote_history_maxlen`	integer > 0	`30`	Inactivated quotes remembered per gateway for `QLEGS RECENT` / `ALL`. Memory-only.
`engine_tuning.drop_copy_buffer_size`	integer > 0	`10000`	Drop-copy events kept for replay when a subscriber reconnects.
`engine_tuning.recent_trades_maxlen`	integer > 0	`20`	Trades kept per symbol for snapshots and diagnostics.
`engine_tuning.depth_snapshot_tolerance_ticks`	integer > 0	`100`	How far from the last trade, in ticks, the published depth reaches.

Note that `auction_indicative_interval_sec` is a top-level key, not part of `engine_tuning`.

5.9 Step 7: a market index

An **index** is a weighted average of constituent prices that `pm-index` calculates as trades print.

```
indices:
  - id: EDU100
    description: EduMatcher broad index
    base_value: 1000.0
    publish_interval_sec: 1.0
    constituents:
      - AAPL
      - MSFT
      - TSLA
```

Field	Type	Default	Meaning
`id`	alphanumeric string, **required**	—	Unique index ID, up to five indices per file.

Field	Type	Default	Meaning
`description`	string, **required**	—	Human-readable label.
`base_value`	number `> 0`	`1000.0`	The index level at inception.
`publish_interval_sec`	number `> 0`	`1.0`	Minimum time between published updates.
`history_file`	path	`data/indexes/<ID>_history.json`	Event history. A relative path is placed under the data directory.
`state_file`	path	`data/indexes/<ID>_state.json`	Checkpoint of the divisor and constituent shares.
`constituents`	non-empty list of symbols, **required**	—	Each must exist in `symbols` **and** have `outstanding_shares`.

`pm-index` creates the history and state directories on start. Leave both paths out unless you have a reason to move them. (`pm-cverifier` warns, `C013`, only when a path you give is below a directory `pm-index` could not create.)

5.10 Step 8: the access points

Everything so far configured the engine. The remaining blocks configure the **processes that stand between the engine and the outside world**. Each is read only by its own process, and each is optional — an absent block means every default shown.

```
alf_gateway:
  enabled: true
  name: alf-gwy01
  bind_address: 0.0.0.0
  port: 5565
  heartbeat_interval_sec: 5
  handshake_timeout_sec: 10
  idle_timeout_sec: 30
  max_connections: 1024
  max_client_queue: 10000
  max_commands_per_second: 100
  max_errors_before_disconnect: 50
  error_window_sec: 60

balf_gateway:
  enabled: true
  name: balf-gwy01
  bind_address: 0.0.0.0
  port: 5560
  heartbeat_interval_sec: 1
  heartbeat_timeout_sec: 5
  idle_timeout_sec: 30
  auth_timeout_sec: 10
  max_connections: 64
  max_client_queue: 10000
  max_messages_per_second: 100
  max_errors_before_disconnect: 10
  error_window_sec: 60
  duplicate_session_policy: REJECT_NEW

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

post_trade_gateway:
  name: ralf-gwy01
  bind_address: 0.0.0.0
  port: 5580
  replay_retention_sec: 86400
  heartbeat_interval_sec: 2
  idle_timeout_sec: 20
  max_connections: 64
  max_client_queue: 10000
  allowed_roles:
    - CLEARING
    - DROP_COPY
    - AUDIT

dc_gateway:
  name: dc-gwy01
  bind_address: 0.0.0.0
  port: 5590
  heartbeat_interval_sec: 2
  idle_timeout_sec: 20
  max_connections: 64
  max_client_queue: 10000
```

Fields that appear in several blocks mean the same thing wherever they appear:

Field	Meaning
`enabled`	`false` makes the process log a warning and refuse to start. (`post_trade_gateway` and `dc_gateway` have no such switch — simply do not run the process.)
`name`	The gateway name reported to clients in the welcome / logon reply.
`bind_address`	The interface the listener binds. Use `127.0.0.1` for loopback only.
`port`	TCP port, `1`–`65535`.
`heartbeat_interval_sec`	How often the gateway sends a heartbeat on an otherwise quiet connection.
`idle_timeout_sec`	How long a client may stay silent before it is disconnected.
`max_connections`	Simultaneous connection cap.
`max_client_queue`	How many outbound messages may queue for one client that is not reading, before it is disconnected as a slow client.

5.10.1 `alf_gateway` — text order entry (`pm-alf-gwy`)

Who may log in, and as what role, comes from `participants`, not from this block.

Field	Default	Meaning
`enabled`, `name`, `bind_address`, `port`	`true`, `alf-gwy01`, `0.0.0.0`, `5565`	As above.
`heartbeat_interval_sec`	`5`	Advertised to clients as `HBINT=`.
`handshake_timeout_sec`	`10`	Time a new connection has to complete `HELLO`.
`idle_timeout_sec`	`30`	Advertised to clients as `IDLE=`.
`max_connections`	`1024`	
`max_client_queue`	`10000`	
`max_commands_per_second`	`100`	Per-client command rate limit.
`max_errors_before_disconnect`	`50`	Protocol errors tolerated within `error_window_sec`.
`error_window_sec`	`60`	Length of the sliding error window.

All values are integers greater than zero. See ALF Gateway.

5.10.2 `balf_gateway` — binary order entry (`pm-balf-gwy`)

Field	Default	Meaning
`enabled`, `name`, `bind_address`, `port`	`true`, `balf-gwy01`, `0.0.0.0`, `5560`	As above.
`heartbeat_interval_sec`	`1`	Interval between server heartbeats on a quiet connection.
`heartbeat_timeout_sec`	`5`	Disconnect when nothing arrives within this time. This, not `idle_timeout_sec`, drives idle disconnects.

Field	Default	Meaning
`idle_timeout_sec`	`30`	Accepted but not used by the runtime today; reserved.
`auth_timeout_sec`	`10`	Time to complete `LOGON` before the connection is closed.
`max_connections`	`64`	
`max_client_queue`	`10000`	
`max_messages_per_second`	`100`	Per-client inbound rate limit.
`max_errors_before_disconnect`	`10`	Errors tolerated within `error_window_sec`.
`error_window_sec`	`60`	
`duplicate_session_policy`	`REJECT_NEW`	A second `LOGON` for a gateway ID already connected: `REJECT_NEW` refuses the newcomer, `EVICT_OLD` drops the existing session.

See BALF Gateway.

5.10.3 `market_data_gateway` — market data (`pm-md-gwy`, CALF)

Field	Default	Meaning
`enabled`, `name`, `bind_address`, `port`	`true`, `md-gwy01`, `0.0.0.0`, `5570`	As above.
`heartbeat_interval_sec`	`1`	
`idle_timeout_sec`	`5`	
`replay_window_sec`	`30`	How much history is held for a client that reconnects and asks to resume.
`max_connections`	`64`	
`max_messages_per_second`	`200`	Per-client inbound rate limit.
`max_symbols_per_client`	`200`	Subscription cap per client.
`max_client_queue`	`10000`	
`depth_levels`	`10`	Aggregated price levels per side sent on the `DEPTH` channel.

All values are integers greater than zero. See Market Data Feed.

5.10.4 `post_trade_gateway` — post-trade feed (`pm-ralf-gwy`, RALF)

Field	Default	Meaning
`name`, `bind_address`, `port`	`ralf-gwy01`, `0.0.0.0`, `5580`	As above.
`replay_retention_sec`	`86400`	How long executed trades are held in memory for replay (24 h).
`heartbeat_interval_sec`	`2`	
`idle_timeout_sec`	`20`	
`max_connections`	`64`	
`max_client_queue`	`10000`	

Field	Default	Meaning
`allowed_roles`	`CLEARING`, `DROP_COPY`, `AUDIT`	Which `HELLO ROLE=` values a subscriber may present.

See Post-Trade Dissemination.

5.10.5 `dc_gateway` — drop copy (`pm-dc-gwy`)

Field	Default	Meaning
`name`, `bind_address`, `port`	`dc-gwy01`, `0.0.0.0`, `5590`	As above.
`heartbeat_interval_sec`	`2`	
`idle_timeout_sec`	`20`	
`max_connections`	`64`	
`max_client_queue`	`10000`	

See Drop-Copy Gateway.

5.10.6 `api_gateways` — REST and WebSocket (`pm-api-gwy`)

Unlike the others, `api_gateways` is a **mapping of named instances**. Each instance is one `pm-api-gwy` process, started with `pm-api-gwy --instance <name>`; the example defines one named `desk`. Several instances let you give different audiences different keys on different ports.

Field	Default	Meaning
`enabled`	`true`	
`host`	`0.0.0.0`	Listen interface.
`port`	`8080`	HTTP port. Give each instance its own.
`swagger_enabled`	`true`	Serve `/docs` and `/openapi.json`.
`log_level`	`info`	Log verbosity for the process.
`stats_db`	`data/stats.db`	`pm-stats` database, for history endpoints.
`audit_db`	`data/audit_index.db`	`pm-audit` index, used only by the order-audit endpoint (`503` if the file is absent).
`clearing_db`	`data/clearing.db`	`pm-clearing` ledger, used only for realised P&L on `GET /positions` (reported as `null` if absent).
`order_retention_sec`	`3600`	How long a finished order stays in the in-memory cache; `0` never evicts.
`market_data_cache_sec`	`60`	Lifetime of the per-symbol cache behind WebSocket snapshots and resume.
`credentials`	empty	The accepted API keys (below).
`credentials_file`	none	A file holding the keys instead (below). Not allowed together with `credentials`.
`rate_limit.writes_per_second`	`10`	Sustained write rate per key.

Field	Default	Meaning
`rate_limit.burst`	`20`	Burst allowance per key. Reads are not limited.
`timeouts.engine_auth_sec`	`3.0`	Wait for the engine to accept the gateway's login.
`timeouts.engine_reply_sec`	`3.0`	Wait for an engine reply to a request.
`timeouts.wait_ack_sec`	`3.0`	Wait used by `?wait=ack`.

Each entry of `credentials`:

Field	Meaning
`api_key`	The bearer token a client sends. **Required.** Treat it as a password.
`gateway_id`	The `participants` ID the key trades as, or `null` for a **read-only** key. A given non-null ID may appear in only one instance.
`description`	Free-text label.

Relative database paths are placed under the data directory.

Keys in their own file. With one key per participant the `credentials` list grows as long as the participant list, and the keys are secrets that do not belong in a file you keep under version control. Name a file instead:

```
api_gateways:
  desk:
    port: 8080
    credentials_file: desk-keys.yaml
```

```
# desk-keys.yaml -- a YAML list, one entry per key
- api_key: key-trader01-4ye7zh
  gateway_id: TRADER01
  description: Student desk 1
- api_key: key-dashboard-9fk2qa
  gateway_id: null
  description: Read-only dashboard
```

The path follows the same rules as `include:` (relative to the file that names it, no absolute path, no `..`), and an instance may use `credentials` or `credentials_file`, not both. Everything else treats the file as if its list had been written inline: the verifier checks every key, a finding names the file (`in desk-keys.yaml`), and `pm-config-deploy` compiles the keys into the artifact, installs a copy of the file beside it and reports an edit you have not deployed. Several instances may share one file. The file is not protected by EduMatcher: keep it readable by the exchange's user only (`pm-config-gen` creates it with mode `0600`).

Replace the example keys

The `REPLACE-WITH-…` values in the listing are placeholders that anyone who has read this book knows. Use long random strings, and keep the deployed config readable only by the exchange's own user. `pm-config-gen` can generate keys for you. Add at least one read-only key (`gateway_id: null`): without it `pm-cverifier` warns (`M032`) and `pm-dash`, the terminal GUI and the book GUI have nothing to read with.
See API Gateway.

5.11 Step 9: operations — logging and health

The last two blocks are about **watching** the exchange.

```
log_server:
  enabled: true
  name: log-srv01
  bind_address: 0.0.0.0
  port: 5600
  db_path: data/log.db
  retention_days: 30
  max_message_bytes: 65536
  max_client_queue: 10000
  write_batch_size: 50
  write_batch_interval_ms: 100
  heartbeat_interval_sec: 5
  pubsub_enabled: true
  pub_port: 5601
  pull_port: 5602
  lease_sec: 30
  max_lease_sec: 300
  max_subscribers: 32
  notify_interval_ms: 250
  backfill_chunk_rows: 500
  max_backfill_minutes: 1440
  max_backfill_rows: 100000
  max_pending_rows: 20000
  pub_sndhwm: 10000

monitor:
  enabled: true
  bind_address: 127.0.0.1
  host: 127.0.0.1
  pull_port: 5610
  pub_port: 5611
  beacon_interval_sec: 2
  hung_after_sec: 5
  silent_after_beacons: 3
  book_metrics_interval_sec: 10
  retention:
    raw_hours: 6
    minute_days: 7
  thresholds:
    host_cpu_pct:         {warn: 80,   crit: 95}
    host_mem_pct:         {warn: 80,   crit: 92}
    disk_free_pct:        {warn: 15,   crit: 5}
    engine_handle_p99_us: {warn: 5000, crit: 20000}
    reject_pct_5m:        {warn: 10,   crit: 30}
    rss_growth_pct_1h:    {warn: 50,   crit: 100}
    heap_bloat_ratio:     {warn: 1.5,  crit: 3}
```

5.11.1 `log_server` — central log collector (`pm-log-srv`)

Every `pm-*` process sends its log lines to `pm-log-srv`, which stores them in SQLite. The first group configures how logs come **in**.

Field	Default	Meaning
`enabled`, `name`, `bind_address`, `port`	`true`, `log-srv01`, `0.0.0.0`, `5600`	As above.
`db_path`	`data/log.db`	The SQLite database. A relative path is placed under the data directory.
`retention_days`	`30`	Rows older than this are pruned hourly. `null` keeps everything.

Field	Default	Meaning
`max_message_bytes`	`65536`	Longer messages are truncated, never dropped.
`max_client_queue`	`10000`	Per-connection backlog before backpressure.
`write_batch_size`	`50`	Most rows written per SQLite transaction.
`write_batch_interval_ms`	`100`	Longest wait before a partial batch is written.
`heartbeat_interval_sec`	`5`	How often a client must send something; the server disconnects after twice this much silence.

The second group configures **LALF-PS**, the ZeroMQ interface that pushes live rows to log viewers such as `pm-log-ui`:

Field	Default	Meaning
`pubsub_enabled`	`true`	`false` binds no ZeroMQ socket; `pm-log-srv` is then a pure TCP collector.
`pub_port`	`5601`	ZeroMQ `PUB` port carrying live rows.
`pull_port`	`5602`	ZeroMQ `PULL` port receiving subscriber requests.
`lease_sec`	`30`	Lifetime of a subscription unless renewed.
`max_lease_sec`	`300`	Ceiling on a requested lease; larger requests are clamped. Must be `>= lease_sec`.
`max_subscribers`	`32`	Concurrent subscriptions.
`notify_interval_ms`	`250`	Coalescing window for notify-mode ticks.
`backfill_chunk_rows`	`500`	Rows per backfill chunk.
`max_backfill_minutes`	`1440`	Largest "last n minutes" window a viewer may ask for.
`max_backfill_rows`	`100000`	Hard cap on rows in one backfill.
`max_pending_rows`	`20000`	Per-subscriber stream buffer; a slow viewer loses its oldest rows.
`pub_sndhwm`	`10000`	ZeroMQ send high-water mark.

`port`, `pub_port` and `pull_port` must all differ. See Centralized Log Server.

5.11.2 `monitor` — health beacons (`pm-monitor`)

Every process sends a small **beacon** to `pm-monitor`, which judges health and publishes it to `pm-dash` and `pm-monitor-cli`. An absent block means these defaults, with monitoring on.

Field	Default	Meaning
`enabled`	`true`	`false`: no process sends beacons and `pm-monitor` idles.
`bind_address`	`127.0.0.1`	Interface `pm-monitor` binds.
`host`	`127.0.0.1`	Where processes send beacons and where `pm-monitor-cli` and the dashboard connect. Must be able to reach `bind_address`.
`pull_port`	`5610`	Beacons in.
`pub_port`	`5611`	Health snapshots and events out.
`beacon_interval_sec`	`2`	How often each process reports.
`hung_after_sec`	`5`	A process whose main loop is older than this is `HUNG`. Must exceed `beacon_interval_sec`.
`silent_after_beacons`	`3`	Missed beacons before a process is `SILENT` (alive) or `DEAD`.
`book_metrics_interval_sec`	`10`	How often the engine recomputes per-book figures.
`retention.raw_hours`	`6`	Hours of 10-second samples kept.
`retention.minute_days`	`7`	Days of one-minute buckets kept. Daily totals are kept indefinitely.
`thresholds.<rule>.warn` / `.crit`	see below	The levels at which a reading turns yellow / red.

The seven threshold rules and their defaults:

Rule	`warn`	`crit`	Reading
`host_cpu_pct`	80	95	Host CPU use, %
`host_mem_pct`	80	92	Host memory use, %
`disk_free_pct`	15	5	Free disk, % — **lower is worse**, so `warn` is above `crit`
`engine_handle_p99_us`	5000	20000	Engine message-handling 99th percentile, µs
`reject_pct_5m`	10	30	Order rejects over five minutes, %
`rss_growth_pct_1h`	50	100	Process memory growth over an hour, %
`heap_bloat_ratio`	1.5	3	Book heap size over the size compaction tolerates (1 = at the limit)

A rule you leave out keeps its default. See Health Monitoring.

5.12 The complete file

This is Step 9's cumulative file in one piece — everything above, ready to copy, edit and deploy. It has no errors, no warnings and no completeness advisories beyond one informational note that `snapshot_interval_sec` is at its default.

```
sessions_enabled: true

symbols:
  AAPL:
    tick_decimals: 2
    last_buy_price: 209.50
    last_sell_price: 210.50
    outstanding_shares: 15400000000
    order_limits:
      max_order_qty: 10000
      max_order_value: 2000000
    collar:
      dynamic_band_pct: 0.015
    market_maker_quotes:
      - gateway_id: MM01
        quote_id: SEED-MM01-AAPL
        bid_price: 209.45
        ask_price: 209.55
        bid_qty: 2000
        ask_qty: 2000
        tif: DAY
        seed_once: true
      - gateway_id: MM02
        quote_id: SEED-MM02-AAPL
        bid_price: 209.40
        ask_price: 209.50
        bid_qty: 1000
        ask_qty: 1000
  MSFT:
    tick_decimals: 2
    level: L1
    last_buy_price: 415.00
    last_sell_price: 415.50
    outstanding_shares: 7430000000
    market_maker_quotes:
      - gateway_id: MM01
        bid_price: 414.90
        ask_price: 415.10
        bid_qty: 1000
        ask_qty: 1000
  TSLA:
    tick_decimals: 2
    level: L3
    last_buy_price: 248.00
    last_sell_price: 249.00
    outstanding_shares: 3200000000
    circuit_breaker:
      reference_window_ns: 600000000000
      levels:
        L1:
          halt_duration_ns: 600000000000
        L2:
          halt_duration_ns: 1800000000000
      reopening:
        initial_band_pct: 0.15
    market_maker_quotes:
      - gateway_id: MM01
        bid_price: 247.90
        ask_price: 248.10
        bid_qty: 500
        ask_qty: 500
        seed_once: false

participants:
  - id: TRADER01
    description: Student workstation 1
```

Verify, deploy and inspect it:

```
pm-cverifier engine_config.yaml
pm-config-deploy engine_config.yaml
pm-config-show
```

```
pm-cverifier engine_config.yaml

✓ YAML syntax: OK
✓ Schema: 0 errors
✓ Semantic: 0 warnings
i Completeness: 1 advisory

[C007] INFO  snapshot_interval_sec is at the default of 0.5 seconds.

Symbols          3  (AAPL, MSFT, TSLA)
Gateways         5  (TRADER01: TRADER, TRADER02: TRADER, MM01: MARKET_MAKER, MM02: MARKET_MAKER … +1 more)
Sessions         enabled — 5 of 7 days scheduled
Collars          enabled — L1: static=30%, dynamic=5%; L2: static=20%, dynamic=2%; L3: static=12%, dynamic=1%
Order limits     AAPL: max qty 10000, max value 2000000
Circuit breakers enabled — L1=7% (5 min), L2=13% (15 min), L3=20% (rest-of-day)
MM obligations   enforced
Admin gateway    OPS01
Indices          1  (EDU100)

Verdict:  ✓ OK — no issues found
```

(Advisory suggestions and rule lines are shortened here.) Starting `pm-engine` on the deployed file logs the seeds going in:

```
Session handling: enabled (startup state: CLOSED)
Risk enforcement: collars=on, circuit_breakers=on
MM quote SEED-MM01-AAPL AAPL bid=209.45x2000 ask=209.55x2000 gw=MM01
MM quote SEED-MM02-AAPL AAPL bid=209.4x1000 ask=209.5x1000 gw=MM02
MM quote SEED-MM01-MSFT-1 MSFT bid=414.9x1000 ask=415.1x1000 gw=MM01
MM quote SEED-MM01-TSLA-1 TSLA bid=247.9x500 ask=248.1x500 gw=MM01
COMBO SEED-PAIR-AAPL-MSFT accepted (2 legs) from MM
Injected 4 market-maker quote(s) and 1 combo(s).
```

5.13 Ports at a glance

Every listener has its own port. The configurable ones come from the blocks above; the others are fixed or set through environment variables and appear nowhere in `engine_config.yaml`.

	Port	Protocol	Process	Purpose	Set by
	5553	ZMQ PUB	`pm-market-sim`	True values and heartbeat	`EDUMATCHER_SIM_PUB_PORT`
	5554	ZMQ PULL	`pm-market-sim`	Instructor commands	`EDUMATCHER_SIM_PULL_PORT`
	5555	ZMQ PULL	`pm-engine`	Order intake	fixed
	5556	ZMQ PUB	`pm-engine`	Event and book feed	fixed
	5557	ZMQ PUB	`pm-engine`	Drop-copy feed	fixed
	5558	ZMQ PUB	`pm-index`	Index values	`EDUMATCHER_INDEX_PUB_PORT`
	5559	ZMQ PULL	`pm-index`	Index commands	`EDUMATCHER_INDEX_PULL_PORT`
	5560	TCP	`pm-balf-gwy`	Binary order entry	`balf_gateway.port`
	5561	ZMQ REP	`pm-stats`	Health ping	`EDUMATCHER_STATS_HEALTH_PO`
	5562	ZMQ REP	`pm-audit`	Health ping	`EDUMATCHER_AUDIT_HEALTH_PO`
	5563	ZMQ REP	`pm-clearing`	Health ping	`EDUMATCHER_CLEARING_HEALTH`

Port	Protocol	Process	Purpose	Set by
5564	ZMQ REP	`pm-index`	Health ping	`EDUMATCHER_INDEX_HEALTH_POR`
5565	TCP	`pm-alf-gwy`	Text order entry	`alf_gateway.port`
5570	TCP	`pm-md-gwy`	Market data	`market_data_gateway.port`
5580	TCP	`pm-ralf-gwy`	Post-trade feed	`post_trade_gateway.port`
5590	TCP	`pm-dc-gwy`	Drop copy	`dc_gateway.port`
5600	TCP	`pm-log-srv`	Log ingest	`log_server.port`
5601	ZMQ PUB	`pm-log-srv`	Live log feed	`log_server.pub_port`
5602	ZMQ PULL	`pm-log-srv`	Log subscriber control	`log_server.pull_port`
5610	ZMQ PULL	`pm-monitor`	Health beacons in	`monitor.pull_port`
5611	ZMQ PUB	`pm-monitor`	Health feed out	`monitor.pub_port`
8080	HTTP	`pm-api-gwy`	REST / WebSocket	`api_gateways.<name>.port`

The engine's own sockets (5555–5557) bind `127.0.0.1` unless `EDUMATCHER_ENGINE_BIND_HOST` says otherwise, and are not configurable in the file.

5.14 Where to go next

- The Configuration Workflow — generate, verify, deploy and inspect the file.
- Config Verifier — every check code `pm-cverifier` can raise.
- Configuration Schema and Process Blocks and the Engine Configuration Specification — the exhaustive reference and grammar.
- Example Engine Configs — ready-made files for classes of different sizes.

CHAPTER 6

The Configuration Workflow

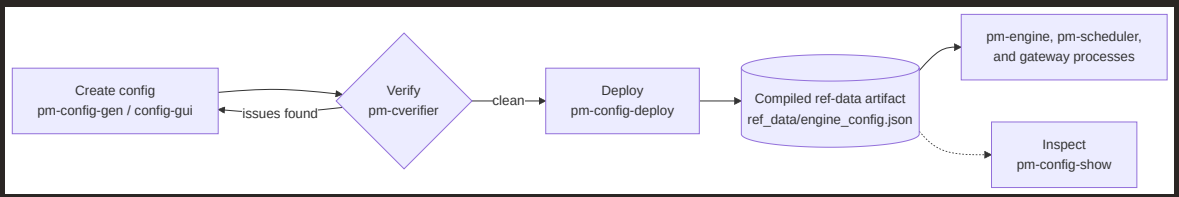
Learning objectives

After reading this page you will understand:

- How a configuration moves through the create → verify → deploy → inspect pipeline
- How to generate a starter config with `pm-config-gen`, and what each of its options does
- How to compile and install a config with `pm-config-deploy`
- How to inspect a deployed config at a glance with `pm-config-show`
- Where to find the structure and every parameter of the file itself
- Which checks to perform before using a config in a class, demo, or test

6.1 Configuration Workflow

`engine_config.yaml` is compiled, not read directly. The file you author is never the file any process runs — it always passes through the same create → verify → deploy pipeline before it becomes the ref-data artifact the exchange actually loads:



The rest of this page documents each stage: generating a starter file, verifying it, deploying it as the compiled artifact every process reads, and inspecting what was deployed.

6.2 Configuring the Exchange

What goes in the file is the next chapter's subject

This chapter is about the **workflow** around `engine_config.yaml`. The file's structure, a minimal version, and every parameter of every block are described in The Configuration File, Block by Block. Read that first if you have not written a config before.

The matching engine and session scheduler both read the configuration. The engine uses it to define the symbol universe, authenticated gateway IDs, session mode, risk controls, market-maker policy, per-symbol outstanding shares, and startup seeds. The scheduler uses the `schedule` and `locale` sections. The optional `alf_gateway`, `balf_gateway`, `post_trade_gateway`, `market_data_gateway`, `dc_gateway`, `api_gateways`, `log_server` and `monitor` sections are read by `pm-alf-gwy`, `pm-balf-gwy`, `pm-ralf-gwy`, `pm-md-gwy`, `pm-dc-gwy`, `pm-api-gwy`, `pm-log-srv` and `pm-monitor` respectively, and `indices` by `pm-index`.

If no configuration has been deployed, `pm-engine` starts in unrestricted mode (it logs a warning saying so): any symbol and gateway can be used, and no startup seeds are loaded. `pm-scheduler` instead refuses to start. If a configuration is deployed, the parser requires two sections:

- `symbols` - a mapping of accepted symbols; generated examples include a positive integer `outstanding_shares` field for each symbol
- `participants` - a list with at least one accepted ALF gateway

Prefer a form over the CLI? Use the Config Builder GUI

If you would rather build or edit `engine_config.yaml` visually — with live validation, per-field help, and import of an existing file — see the Configuration GUI (`config-gui`) chapter. It targets the same file format as `pm-config-gen` described below.

participants lists identities, not gateway processes

The top-level `participants:` list holds the *participant identities* that may log in. The gateway **processes** themselves are configured via separate **top-level** keys (`alf_gateway`, `balf_gateway`, `market_data_gateway`, `post_trade_gateway`, `dc_gateway`, `log_server`, and `api_gateways`) and are read by their own processes, not by `pm-engine`.
Each protocol's configuration lives in a different part of `engine_config.yaml`:

- **ALF** — two parts. *Who may connect* is configured under `participants`; `pm-engine` uses it to authenticate order-entry connections from `pm-alf-console` and `pm-alf-gwy`, as well as `pm-balf-gwy` (the gateway id used in the BALF configurations must exist under `participants`). *How the TCP gateway behaves* (port, timeouts, rate limits) is configured under the top-level `alf_gateway` key and read by `pm-alf-gwy`; see Configuring `pm-alf-gwy`. Uses a pipe-delimited text format (`FIELD=VALUE|FIELD=VALUE`).
- **BALF** — configured under the top-level `balf_gateway` key; used by `pm-balf-gwy`. Uses fixed-width binary frames with sequence numbers and integer-scaled prices, targeting programmatic clients where text-parsing overhead is undesirable. See BALF Gateway for more usage and BALF Protocol for the full specification.
- **CALF** — configured under the top-level `market_data_gateway` key; used by `pm-md-gwy`. Provides a subscribe/unsubscribe market-data feed delivering order-book snapshots, trade prints, and session-state changes over a persistent TCP connection with sequence-based gap detection. See Market Data Feed for usage and CALF Protocol for the full protocol specification.
- **RALF** — configured under the top-level `post_trade_gateway` key; used by `pm-ralf-gwy`. Provides a replayable audit feed of all executed trades, including the original order details, over a persistent TCP connection with sequence-based gap detection. See Post Trade for usage and RALF Protocol for the full protocol specification.
- **DC1** — configured under the top-level `dc_gateway` key; used by `pm-dc-gwy`. Relays the engine's internal drop-copy feed to plain TCP clients that cannot speak ZeroMQ, using the lightweight DC1 text protocol. See Drop-Copy Gateway for full usage and protocol details.
- **LALF** — configured under the top-level `log_server` key; used by `pm-log-srv`. Collects operational `logging`-module output from every other `pm-*` process over a persistent TCP connection into a queryable SQLite database. The same key also configures **LALF-PS**, the ZeroMQ `PUB/PULL` interface that distributes those rows back out to live log viewers. See Centralized Log Server for usage and LALF Protocol Reference for the full protocol specification.
- A Full overview of all protocol and their intended usage can be found in Protocols Overview.

6.3 File Location

EduMatcher separates the configuration you *author* from the one the exchange *runs*.

The authored `engine_config.yaml` lives wherever suits you — normally under version control alongside the rest of your course material. Edit it, review it, diff it.

What the exchange runs is a *compiled artifact* at `<EDUMATCHER_DATA_DIR>/ref_data/engine_config.json`. That is the only file any running process reads. No process accepts a config path, so it is not possible to start two of them against different files.

`<EDUMATCHER_DATA_DIR>` is resolved centrally by the runtime configuration module. `EDUMATCHER_DATA_DIR` takes precedence; without it, a source checkout uses `<repo>/src/data/`, while an installed production package uses `~/.local/share/edumatcher`. Source mode is determined from the installed location of `edumatcher/config.py` (its package parent is named `src`), not from the current working directory. See Reference Manual — how the default is selected for the exact precedence rules.

The same resolver places configured relative runtime paths, such as `data/stats.db` and `data/log.db`, under this shared data directory. This keeps `pm-engine`, `pm-stats`, `pm-log-srv`, and `pm-api-gwy` on the same files even when they are launched from different directories.

See Compile Configs with `pm-config-deploy` for how the authored file becomes that artifact.

6.4 Generate Configs with `pm-config-gen`

`pm-config-gen` creates a parser-compatible `engine_config.yaml` from concise CLI inputs. It is designed for operators and instructors who want to bootstrap new sessions without manually writing large YAML blocks.

Use it when:

- you are creating a new class/demo config from scratch
- you want consistent defaults and validation hints
- you need repeatable config generation in scripts

6.5 Generate Configs with `config-gui`

A more user friendly way to create a configuration file (also known as reference data) is to spin up the Web-App. The easiest way to run it is by running the container image. You can use either `docker` or `podman` but in the examples we use `podman`

1. Download latest `edumatcher-config-gui-<VERSION>.tar.gz` and unzip
2. `podman load --input dist/edumatcher-config-gui-<VERSION>tar.gz`
3. Then run: `podman run -p 8080:8080 edumatcher-config-gui:<VERSION>`

You can then access the UI at `http://localhost:8080/`

More details of the Web app and how to use it can be found in Configuration GUI

6.6 Verify Configs with `pm-cverifier`

Before starting the engine with a hand-written or generated config, run `pm-cverifier` to get a deep, actionable report on every problem the config contains — not just the first one the engine would encounter.

```
# human-readable report
pm-cverifier engine_config.yaml

# CI-friendly: fail on any warning, output JSON
pm-cverifier --strict --format json engine_config.yaml
```

`pm-cverifier` is **read-only** and safe to run at any time. It reports:

- every YAML syntax or schema error that would prevent the engine from starting
- semantic inconsistencies (e.g. sessions enabled but no schedule, MM gateway without seed quotes, index constituent not in `symbols`)
- completeness advisories (e.g. no reference prices, MM obligations not enforced)
- a plain-English **Risk Summary** showing what collars, circuit breakers, and gateways are actually active

For a full description of all check codes and CLI options, see Config Verifier (pm-cverifier).

6.6.1 Quick start

Installed mode:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --sessions-enabled \
  --output engine_config.yaml
```

Poetry/source mode:

```
poetry run pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --sessions-enabled \
  --output engine_config.yaml
```

Print to stdout only (no file write):

```
pm-config-gen \
  --symbols AAPL \
  --participants TRADER01 \
  --outstanding-shares AAPL:15400000000 \
  --dry-run
```

6.6.2 Important behavior

- If `--output` is omitted, YAML is printed to stdout.
- If `--output` exists, generation fails unless `--force` is set.
- If any gateway is `MARKET_MAKER` and you do not pass `--seed-mm-mid-range`, MM quote stubs are emitted with `bid_price: null` and `ask_price: null`. Fill these values before starting `pm-engine`.
- If you pass `--seed-mm-mid-range`, MM quotes are emitted with concrete prices on the configured tick grid.
- Loader validation is skipped only in the MM-stub case above. It runs automatically for non-MM configs, MM configs with seeded midpoints, and `--no-mm-seed-quotes` configs (see below).
- `--no-mm-seed-quotes` sets `require_mm_seed_quotes: false`. A `MARKET_MAKER` gateway may then exist with no `market_maker_quote` entries at all — a genuinely empty book at startup — instead of the null-priced stub. See Market-Maker Quotes for why this differs from real-market practice and when to use it.

MM quote generation decision matrix:

Gateway/flags state	Generated `market_maker_quotes`	Generated `last_buy_price` / `last_sell_price`
No `MARKET_MAKER` gateway configured	No MM quote section emitted	Only emitted if `--seed-last-prices` is set (as `null` placeholders)
`MARKET_MAKER` present, no `--seed-mm-mid-range`	Stub quotes with `bid_price: null`, `ask_price: null`	`null` placeholders only if `--seed-last-prices` is set
`MARKET_MAKER` present, with `--seed-mm-mid-range MIN:MAX`	Concrete bid/ask quote prices generated on tick grid	If `--seed-last-prices-from-mm` is set, both are set to the same midpoint used for seeded quotes
`MARKET_MAKER` present, with `--no-mm-seed-quotes`, no `--seed-mm-mid-range`	No MM quote section emitted (`require_mm_seed_quotes: false` recorded instead)	Only emitted if `--seed-last-prices` is set (as `null` placeholders)
`MARKET_MAKER` present, with `--no-mm-seed-quotes` and `--seed-mm-mid-range MIN:MAX`	No MM quote section emitted — `--no-mm-seed-quotes` always wins; the mid range is used only for last prices	If `--seed-last-prices-from-mm` is set, both are set to the generated midpoint (this is how the `*-nomm` examples get reference prices with an empty book)

In this guide, "MM stub" means a quote row exists but prices are `null` and must be filled manually. "Full MM setup" means concrete bid/ask prices are generated for each MM quote seed at generation time.

6.6.3 Option reference

Required inputs:

Option	Type	Description
`--symbols SYM [SYM ...]`	Repeatable tokens	Symbol universe (uppercased on parse)
`--participants GW_SPEC [GW_SPEC ...]`	Repeatable tokens	Gateway specs as `ID[:ROLE[:DISCONNECT]]`; see the `--participants` format below
`--participant-desc GW_ID:TEXT`	Repeatable	Sets `participants[<GW_ID>].description`; `GW_ID` must be one of `--participants`. Quote it when `TEXT` has spaces
`--participant-smp GW_ID:SMP_ACTION`	Repeatable	Sets `participants[<GW_ID>].smp_action` (`NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`); `GW_ID` must be one of `--participants`; omitted gateways inherit `--participant-default-smp`, else `NONE`. `GW_ID:NONE` is written when it overrides a non-`NONE` default. See Risk Controls — Self-Match Prevention
`--participant-default-smp SMP_ACTION`	Choice	Writes `participant_defaults.smp_action`, inherited by every gateway without its own `smp_action`. See Participant Defaults
`--participant-default-disconnect DISCONNECT`	Choice	Writes `participant_defaults.disconnect_behaviour` (`CANCEL_ALL`, `CANCEL_QUOTES_ONLY`, `LEAVE_ALL`). Gateways whose `--participants` spec does not name a disconnect behaviour inherit it instead of the per-role default; this includes `ADMIN` and `MARKET_MAKER` gateways. See Participant Defaults

Session and schedule options:

Option	Type	Default	Description
`--sessions-enabled` / `--no-sessions-enabled`	Flag pair	`false`	Enable/disable scheduler-driven sessions
`--schedule` / `--no-schedule`	Flag pair	auto	Force include/exclude `schedule`; auto emits when sessions are enabled
`--pre-open HH:MM`	String	`09:00`	Schedule pre-open time
`--opening-auction HH:MM`	String	`09:25`	Opening auction start
`--continuous HH:MM`	String	`09:30`	Continuous start
`--closing-auction HH:MM`	String	`16:00`	Closing auction start
`--closing-end HH:MM`	String	`16:05`	Closing auction end
`--weekend`	Flag	off	Also emit a `weekend:` block (Saturday and Sunday). Without it weekends are `CLOSED`
`--weekend-pre-open HH:MM`, `--weekend-opening-auction HH:MM`, `--weekend-continuous HH:MM`, `--weekend-closing-auction HH:MM`, `--weekend-closing-end HH:MM`	String	`10:00`, `10:25`, `10:30`, `14:00`, `14:05`	The five weekend times; only used with `--weekend`
`--holidays` / `--no-holidays`	Flag pair	off	Also emit a `holidays:` block. Without it bank holidays are `CLOSED`

Option	Type	Default	Description
`--holidays-pre-open HH:MM`, `--holidays-opening-auction HH:MM`, `--holidays-continuous HH:MM`, `--holidays-closing-auction HH:MM`, `--holidays-closing-end HH:MM`	String	`10:00`, `10:25`, `10:30`, `14:00`, `14:05`	The five bank-holiday times; only used with `--holidays`
`--country COUNTRY`	String	unset	Writes `locale.country`: a country name (`Sweden`) or ISO code (`SE`). Unset: no value written, so the engine default `Sweden` applies
`--session-timezone TZ`	String	unset	Writes `locale.session_timezone`, an IANA name such as `America/New_York`. Unset: no value written, so the engine default `Europe/Stockholm` applies. Schedule times are interpreted in this timezone

Core engine and risk options:

Option	Type	Default	Description
`--snapshot-interval SECS`	float (`> 0`)	`0.5`	`engine_tuning.snapshot_interval_sec`
`--quote-history-maxlen N`	int (`> 0`)	`30`	`engine_tuning.quote_history_maxlen`
`--drop-copy-buffer-size N`	int (`> 0`)	`10000`	`engine_tuning.drop_copy_buffer_size`
`--recent-trades-maxlen N`	int (`> 0`)	`20`	`engine_tuning.recent_trades_maxlen`
`--depth-snapshot-tolerance-ticks N`	int (`> 0`)	`100`	`engine_tuning.depth_snapshot_tolerance_ticks`
`--no-collars`	Flag	off	Emit `enforce_collars: false`
`--no-circuit-breakers`	Flag	off	Emit `enforce_circuit_breakers: false`
`--static-band PCT`	float in `(0,1)`	unset	Default risk-control static band (`DEFAULT` level)
`--dynamic-band PCT`	float in `(0,1)`	unset	Default risk-control dynamic band (`DEFAULT` level)
`--symbol-static-band SYM:PCT`	Repeatable	none	Per-symbol `collar.static_band_pct` override
`--symbol-dynamic-band SYM:PCT`	Repeatable	none	Per-symbol `collar.dynamic_band_pct` override
`--symbol-max-order-qty SYM:N`	Repeatable	none	Per-symbol `order_limits.max_order_qty` (shares, `> 0`)
`--symbol-max-order-value SYM:AMOUNT`	Repeatable	none	Per-symbol `order_limits.max_order_value` (notional, `> 0`)
`--symbol-risk-level SYM:LEVEL`	Repeatable	none	Per-symbol `symbols.<SYM>.level` override
`--risk-level NAME:STATIC[:DYNAMIC]`	Repeatable	none	Add named risk levels under `risk_controls.levels`
`--cb-levels NAME:SHIFT[:HALT_MINS] ...`	List	built-in ladder	Circuit-breaker level specs
`--cb-window-ns NS`	int (`> 0`)	`300000000000`	Circuit-breaker reference window
`--no-ace`	Flag	off	Write `circuit_breaker_defaults.reopening.enabled: false` (no reopening corridor)
`--ace-initial-band PCT`	float in `(0,1)`	`0.1`	`reopening.initial_band_pct` — starting corridor half-width, as a fraction of the reference price
`--ace-expansions WIDEN:MINUTES ...`	List	`0.10:2 0.20:5`	`reopening.expansions` — corridor widening rungs, in order; the last rung repeats

Option	Type	Default	Description
`--ace-random-end-ns NS`	int (`>= 0`)	`30000000000`	`reopening.random_end_max_ns` — random tail on every call phase; `0` makes reopen times predictable
`--ace-random-seed N`	int	OS entropy	`reopening.random_seed` — reproducible reopen times for demos

The `--no-ace` and `--ace-*` options shape the `reopening` block of `circuit_breaker_defaults`, which `pm-config-gen` writes when `--cb-levels` is given. Without `--cb-levels` they have no effect. The per-symbol forms are the `ace_*` keys of `--symbol-opts` below.

Market-maker and symbol defaults:

Option	Type	Default	Description
`--mm-spread-ticks N`	int (`> 0`)	`20`	Global MM spread threshold
`--mm-min-qty N`	int (`> 0`)	`100`	Global MM min quantity
`--enforce-mm-obligations` / `--no-enforce-mm-obligations`	Flag pair	`false`	Global MM obligation toggle
`--tick-decimals N`	int `0..8`	`2`	Default `tick_decimals` for symbols
`--outstanding-shares SYM:N`	Repeatable	none	Per-symbol outstanding shares in the generated config
`--seed-last-prices`	Flag	off	Emit `last_buy_price/last_sell_price` placeholders
`--seed N`	int	random source default	Deterministic RNG seed for generated training values
`--seed-mm-mid-range MIN:MAX`	string	none	Seed MM quotes from a random midpoint in the inclusive price range
`--mm-seed-spread-ticks N`	int (`> 0`)	`10`	Half-spread, in ticks, for seeded MM stub quotes (bid/ask sit this many ticks either side of the seeded midpoint)
`--no-mm-seed-quotes`	flag	off	Set `require_mm_seed_quotes: false` — allow a `MARKET_MAKER` gateway with no seeded quotes
`--seed-last-prices-from-mm`	Flag	off	Set `last_buy_price/last_sell_price` to the same midpoint used for seeded MM quotes

Output and safety options:

Option	Type	Default	Description
`--output FILE`	Path	none	Write YAML to file

Option	Type	Default	Description
`--split`	Flag	off	Write `symbols` and `participants` to `symbols.yaml` and `participants.yaml` beside `--output`, joined to the main file by an `include:` list (see Splitting the file). Needs `--output` or `--dry-run`
`--force`	Flag	off	Overwrite existing output file (with `--split`, all three files)
`--dry-run`	Flag	off	Print YAML only; do not write file
`--comment-default-config-fields`	Flag	off	Add a commented-out header showing every optional section with its default values, ready to uncomment — the engine settings and the process sections (the gateways, `api_gateways`, `log_server`, `monitor`, `indices`, `schedule`), each field explained. A section's own option (`--log-server`, `--monitor`, …) writes that section for real instead

Before writing or printing anything, `pm-config-gen` loads the generated YAML with the same loader `pm-engine` uses at startup. If that load fails, it prints `[ERROR] Generated config failed validation: …`, writes nothing (to the file or to stdout, `--dry-run` included) and exits with code `2`, just as it does for invalid flags. A file `pm-config-gen` writes therefore always loads.

The one deliberate exception is a `MARKET_MAKER` participant whose quotes are not seeded (no `--seed-mm-mid-range` and no `--no-mm-seed-quotes`). The file is then a template: `market_maker_quotes` holds `null` bid and ask prices for you to fill in, so validation is skipped and `pm-config-gen` prints a hint instead. Run `pm-cverifier` on the file once you have filled them in.

Post-trade gateway options:

Option	Type	Default	Description
`--post-trade-gateway`	Flag	off	Emit top-level `post_trade_gateway` block for `pm-ralf-gwy`
`--post-trade-name`	string	`ralf-gwy01`	`post_trade_gateway.name`
`--post-trade-bind-address`	string	`0.0.0.0`	`post_trade_gateway.bind_address`
`--post-trade-port`	int (> 0)	`5580`	`post_trade_gateway.port`
`--post-trade-replay-retention-sec`	int (> 0)	`86400`	`post_trade_gateway.replay_retention_sec`
`--post-trade-heartbeat-interval-sec`	int (> 0)	`1`	`post_trade_gateway.heartbeat_interval_sec`
`--post-trade-idle-timeout-sec`	int (> 0)	`5`	`post_trade_gateway.idle_timeout_sec`
`--post-trade-max-connections`	int (> 0)	`64`	`post_trade_gateway.max_connections`
`--post-trade-max-client-queue`	int (> 0)	`10000`	`post_trade_gateway.max_client_queue`

Option	Type	Default	Description
`--post-trade-allowed-roles ROLE [ROLE ...]`	list	`CLEARING DROP_COPY AUDIT`	`post_trade_gateway.allowed_roles`

Market-data gateway options:

Option	Type	Default	Description
`--market-data-gateway`	Flag	off	Emit top-level `market_data_gateway` block for `pm-md-gwy`
`--market-data-enabled` / `--market-data-disabled`	Flag pair	unset (`true` when emitted)	Set `market_data_gateway.enabled`
`--market-data-name`	string	`md-gwy01`	`market_data_gateway.name`
`--market-data-bind-address`	string	`0.0.0.0`	`market_data_gateway.bind_address`
`--market-data-port`	int (> 0)	`5570`	`market_data_gateway.port`
`--market-data-heartbeat-interval-sec`	int (> 0)	`1`	`market_data_gateway.heartbeat_interval_sec`
`--market-data-idle-timeout-sec`	int (> 0)	`5`	`market_data_gateway.idle_timeout_sec`
`--market-data-replay-window-sec`	int (> 0)	`30`	`market_data_gateway.replay_window_sec`
`--market-data-max-connections`	int (> 0)	`64`	`market_data_gateway.max_connections`
`--market-data-max-symbols-per-client`	int (> 0)	`200`	`market_data_gateway.max_symbols_per_client`
`--market-data-max-client-queue`	int (> 0)	`10000`	`market_data_gateway.max_client_queue`
`--market-data-depth-levels`	int (> 0)	`10`	`market_data_gateway.depth_levels`

ALF gateway options:

Option	Type	Default	Description
`--alf-gateway`	Flag	off	Emit top-level `alf_gateway` block for `pm-alf-gwy`; any other `--alf-*` option also emits it
`--alf-enabled` / `--alf-disabled`	Flag pair	enabled	`alf_gateway.enabled`
`--alf-name`	string	`alf-gwy01`	`alf_gateway.name`
`--alf-bind-address`	string	`0.0.0.0`	`alf_gateway.bind_address`
`--alf-port`	int (`1..65535`)	`5565`	`alf_gateway.port`
`--alf-heartbeat-interval-sec`	int (> 0)	`5`	`alf_gateway.heartbeat_interval_se`
`--alf-handshake-timeout-sec`	int (> 0)	`10`	`alf_gateway.handshake_timeout_sec`
`--alf-idle-timeout-sec`	int (> 0)	`30`	`alf_gateway.idle_timeout_sec`
`--alf-max-connections`	int (> 0)	`1024`	`alf_gateway.max_connections`
`--alf-max-client-queue`	int (> 0)	`10000`	`alf_gateway.max_client_queue`
`--alf-max-commands-per-second`	int (> 0)	`100`	`alf_gateway.max_commands_per_seco`
`--alf-max-errors-before-disconnect`	int (> 0)	`50`	`alf_gateway.max_errors_before_dis`
`--alf-error-window-sec`	int (> 0)	`60`	`alf_gateway.error_window_sec`

BALF gateway options:

Option	Type	Default	Description
`--balf-gateway`	Flag	off	Emit top-level `balf_gateway` block for `pm-balf-gwy`
`--balf-name`	string	`balf-gwy01`	`balf_gateway.name`
`--balf-bind-address`	string	`0.0.0.0`	`balf_gateway.bind_address`
`--balf-port`	int (> 0)	`5560`	`balf_gateway.port`
`--balf-heartbeat-interval-sec`	int (> 0)	`1`	`balf_gateway.heartbeat_interval_`
`--balf-heartbeat-timeout-sec`	int (> 0)	`5`	`balf_gateway.heartbeat_timeout_s`
`--balf-idle-timeout-sec`	int (> 0)	`30`	`balf_gateway.idle_timeout_sec`
`--balf-auth-timeout-sec`	int (> 0)	`10`	`balf_gateway.auth_timeout_sec`
`--balf-max-connections`	int (> 0)	`64`	`balf_gateway.max_connections`
`--balf-max-client-queue`	int (> 0)	`10000`	`balf_gateway.max_client_queue`
`--balf-max-messages-per-second`	int (> 0)	`100`	`balf_gateway.max_messages_per_se`
`--balf-max-errors-before-disconnect`	int (> 0)	`10`	`balf_gateway.max_errors_before_d`
`--balf-error-window-sec`	int (> 0)	`60`	`balf_gateway.error_window_sec`
`--balf-duplicate-session-policy`	enum	`REJECT_NEW`	`balf_gateway.duplicate_session_p` `REJECT_NEW` or `EVICT_OLD`

Drop-copy gateway options:

Option	Type	Default	Description
`--dc-gateway`	Flag	off	Emit top-level `dc_gateway` block for `pm-dc-gwy`
`--dc-name`	string	`dc-gwy01`	`dc_gateway.name`
`--dc-bind-address`	string	`0.0.0.0`	`dc_gateway.bind_address`
`--dc-port`	int (> 0)	`5590`	`dc_gateway.port`
`--dc-heartbeat-interval-sec`	int (> 0)	`5`	`dc_gateway.heartbeat_interval_se`
`--dc-idle-timeout-sec`	int (> 0)	`30`	`dc_gateway.idle_timeout_sec`
`--dc-max-connections`	int (> 0)	`64`	`dc_gateway.max_connections`
`--dc-max-client-queue`	int (> 0)	`10000`	`dc_gateway.max_client_queue`

Log server options:

Option	Type	Default	Description
`--log-server`	Flag	off	Emit top-level `log_server` block for `pm-log-srv`
`--log-server-enabled` / `--log-server-disabled`	Flag pair	unset (`true` when emitted)	Set `log_server.enabled`
`--log-server-name`	string	`log-srv01`	`log_server.name`
`--log-server-bind-address`	string	`0.0.0.0`	`log_server.bind_address`
`--log-server-port`	int (> 0)	`5600`	`log_server.port`
`--log-server-db-path`	path	`data/log.db`	`log_server.db_path`

Option	Type	Default	Description
`--log-server-retention-days`	int (`>= 0`)	`30`	`log_server.retention_days`; `0` means unbounded retention
`--log-server-max-message-bytes`	int (`> 0`)	`65536`	`log_server.max_message_bytes`
`--log-server-max-client-queue`	int (`> 0`)	`10000`	`log_server.max_client_queue`
`--log-server-write-batch-size`	int (`> 0`)	`50`	`log_server.write_batch_size`
`--log-server-write-batch-interval-ms`	int (`>= 0`)	`100`	`log_server.write_batch_interval_m`
`--log-server-heartbeat-interval-sec`	int (`> 0`)	`5`	`log_server.heartbeat_interval_sec`

Log server LALF-PS options — the ZeroMQ log-distribution interface, see LALF-PS:

Option	Type	Default	Description
`--log-server-pubsub-enabled` / `--log-server-pubsub-disabled`	Flag pair	unset (`true` when emitted)	Set `log_server.pubsub_enabled` — the master switch for LALF-PS
`--log-server-pub-port`	int (`> 0`)	`5601`	`log_server.pub_port` — ZeroMQ `PUB` socket carrying rows, ticks, backfill chunks and acks
`--log-server-pull-port`	int (`> 0`)	`5602`	`log_server.pull_port` — ZeroMQ `PULL` socket receiving subscriber control requests
`--log-server-lease-sec`	int (`> 0`)	`30`	`log_server.lease_sec` — subscription lease TTL
`--log-server-max-lease-sec`	int (`>= lease_sec`)	`300`	`log_server.max_lease_sec` — ceiling on a subscriber's requested lease
`--log-server-max-subscribers`	int (`> 0`)	`32`	`log_server.max_subscribers`
`--log-server-notify-interval-ms`	int (`> 0`)	`250`	`log_server.notify_interval_ms` — `NOTIFY`-mode coalescing window
`--log-server-backfill-chunk-rows`	int (`> 0`)	`500`	`log_server.backfill_chunk_rows`
`--log-server-max-backfill-minutes`	int (`> 0`)	`1440`	`log_server.max_backfill_minutes`
`--log-server-max-backfill-rows`	int (`> 0`)	`100000`	`log_server.max_backfill_rows`
`--log-server-max-pending-rows`	int (`> 0`)	`20000`	`log_server.max_pending_rows`
`--log-server-pub-sndhwm`	int (`> 0`)	`10000`	`log_server.pub_sndhwm`

Passing any of these implies the `log_server` block, so `--log-server` is not additionally required. `pm-config-gen` refuses to write a file that `pm-log-srv` would then refuse to start on: `port`, `pub_port` and `pull_port` must resolve to three different numbers (compared against each other's *defaults*, not only against explicitly-set values), and `max-lease-sec` must be at least `lease-sec`.

```
# Two log servers on one host — move the whole three-port block
pm-config-gen --symbols AAPL --participants GW01 \
  --log-server-port 5700 --log-server-pub-port 5701 --log-server-pull-port 5702

# Collect logs but publish nothing: no ZeroMQ socket is bound
pm-config-gen --symbols AAPL --participants GW01 --log-server-pubsub-disabled

# Reap dead viewers within 10 s, and cap concurrent viewers at 8
pm-config-gen --symbols AAPL --participants GW01 \
  --log-server-lease-sec 10 --log-server-max-subscribers 8
```

Monitor options — health beacons and `pm-monitor`, see Health Monitoring:

Option	Type	Default	Description
`--monitor`	Flag	off	Emit the top-level `monitor` block with every field. Without it the block is omitted and every process runs on the defaults
`--monitor-enabled` / `--monitor-disabled`	Flag pair	unset (`true` when emitted)	Set `monitor.enabled`; disabled means no beacons and an idle `pm-monitor`
`--monitor-bind-address`	string	`127.0.0.1`	`monitor.bind_address` — the interface `pm-monitor` binds
`--monitor-host`	string	`127.0.0.1`	`monitor.host` — where every process sends its beacons
`--monitor-pull-port`	int	`5610`	`monitor.pull_port` — beacons in
`--monitor-pub-port`	int	`5611`	`monitor.pub_port` — snapshots and events out
`--monitor-beacon-interval-sec`	float (`> 0`)	`2`	`monitor.beacon_interval_sec`
`--monitor-hung-after-sec`	float (`> interval`)	`5`	`monitor.hung_after_sec`
`--monitor-silent-after-beacons`	int (`> 0`)	`3`	`monitor.silent_after_beacons`
`--monitor-book-metrics-interval-sec`	float (`> 0`)	`10`	`monitor.book_metrics_interval_se`
`--monitor-raw-hours`	int (`> 0`)	`6`	`monitor.retention.raw_hours`
`--monitor-minute-days`	int (`> 0`)	`7`	`monitor.retention.minute_days`
`--monitor-threshold RULE:WARN:CRIT`	repeatable	the §9.6 bands	One threshold band, e.g. `host_cpu_pct:70:90`; rules `host_cpu_pct`, `host_mem_pct`, `disk_free_pct`, `engine_handle_p99_us`, `reject_pct_5m`, `rss_growth_pct_1h`, `heap_bloat_ratio`

Passing any of these implies the `monitor` block. The emitted block lists every field, defaults included, so the file shows what `pm-monitor` will run with. `pm-config-gen` refuses (exit 2) what `pm-config-deploy` would refuse: a value out of range, an unknown rule, a band in the wrong order (warn below crit, except `disk_free_pct` where a lower value is worse — `M029`), `hung-after-sec` not above the beacon interval (`M030`), or equal pull and pub ports. A `host` that cannot reach `bind_address` is written with a warning (`M031`).

```
# Monitor a busier exchange: quieter CPU alarm, longer raw history
pm-config-gen --symbols AAPL --participants GW01 \
  --monitor-threshold host_cpu_pct:90:98 --monitor-raw-hours 12

# Beacons from other hosts: pm-monitor listens everywhere, clients send to its name
pm-config-gen --symbols AAPL --participants GW01 \
  --monitor-bind-address 0.0.0.0 --monitor-host monitor.example.lan
```

API gateway options:

Option	Type	Default	Description
`--api-gateway`	Flag	off	Emit top-level `api_gateways` block for `pm-api-gwy`
`--api-gateway-name NAME`	string	`default`	Name of the generated `api_gateways.<NAME>` entry for single-process generation
`--api-gateway-instance NAME:GATEWAY[,GATEWAY...][:PORT]`	Repeatable	none	Emit one named API gateway process per option; use `NAME::PORT` for an identity-free read-only process
`--api-gateway-enabled` / `--api-gateway-disabled`	Flag pair	unset (`true` when emitted)	Set each generated API gateway `enabled` field
`--api-gateway-host ADDR`	string	`0.0.0.0`	HTTP bind address
`--api-gateway-port N`	int (`> 0`)	`8080`	HTTP listen port
`--api-gateway-swagger-enabled` / `--api-gateway-swagger-disabled`	Flag pair	unset (`true` when emitted)	Enable or disable `/docs` and `/openapi.json`
`--api-gateway-log-level LEVEL`	enum	`info`	`debug`, `info`, `warning`, or `error`
`--api-gateway-stats-db PATH`	path	`data/stats.db`	SQLite database used by `/history/*` endpoints
`--api-key KEY:GATEWAY_ID[:DESCRIPTION]`	Repeatable	none	Add an explicit bearer-token credential; use `GATEWAY_ID=null` for read-only access
`--api-gateway-generate-keys` / `--no-api-gateway-generate-keys`	Flag pair	generated when emitted	Generate one key for each `participants` entry
`--api-keys-for GW_ID [GW_ID ...]`	Tokens	none	Write one key for each listed participant to `api_keys.yaml` beside `--output` and point the API gateway at it with `credentials_file`; needs `--output` or `--dry-run`, and a single API gateway. See Keys for many participants
`--api-gateway-readonly-key`	Flag	off	Generate an additional read-only key with `gateway_id: null`
`--api-gateway-rate-limit-writes-per-second N`	int (`> 0`)	`10`	Per-key write rate limit

Option	Type	Default	Description
`--api-gateway-rate-limit-burst N`	int (> 0)	`20`	Per-key write burst capacity
`--api-gateway-engine-auth-secs`	float (> 0)	`3.0`	Engine auth timeout field
`--api-gateway-engine-reply-secs`	float (> 0)	`3.0`	Engine request/reply timeout
`--api-gateway-wait-ack-secs`	float (> 0)	`3.0`	`?wait=ack` timeout
`--api-gateway-order-retention-secs`	int (>= 0)	`3600`	Seconds a terminal order stays in the gateway's in-memory cache. `0` disables eviction

Combo seed options:

Option	Type	Default	Description
`--combo COMBO_SPEC`	Repeatable	none	Seed a `market_maker_combos` entry; format described in `--combo` format

Index options:

Option	Type	Default	Description
`--index ID[:DESCRIPTION]`	Repeatable	none	Define an index; `ID` is letters and digits, `DESCRIPTION` is optional text after the first colon
`--index-constituents ID:SYM[,SYM,...]`	Repeatable	none	Set constituent symbols for the named index
`--index-base-value ID:VALUE`	Repeatable	`1000.0`	Override `base_value` for the named index
`--index-interval ID:SECS`	Repeatable	`1.0`	Override `publish_interval_sec` for the named index
`--index-history-file ID:PATH`	Repeatable	derived from ID	Override `history_file` path; default is `data/indexes/<ID>_history.jsonl`
`--index-state-file ID:PATH`	Repeatable	derived from ID	Override `state_file` path; default is `data/indexes/<ID>_state.json`

Up to 5 indices may be defined. Each `ID` may contain only letters and digits (no hyphens or underscores), and is upper-cased. The other `--index-*` flags are repeatable across indices but take one value per index: give each at most once per `ID`, listing all of an index's constituents in a single `--index-constituents`, and only for an `ID` defined with `--index`. Every index must have at least one constituent, with no symbol listed twice. When `history_file` and `state_file` are omitted, `pm-config-gen` derives them from the index ID under `data/indexes/`.

Every constituent must also be something `pm-index` can price, so `pm-config-gen` refuses a constituent that has:

- no `--outstanding-shares SYM:N`, which `pm-index` needs to weight constituents by market capitalisation;
- no reference price. `pm-index` sets each index's initial divisor from the constituents' last prices, and `pm-config-gen` only writes real prices when it seeds them from a market-maker midpoint: add a `MARKET_MAKER` participant with `--seed-mm-mid-range MIN:MAX --seed-last-prices-from-mm`. Add `--no-mm-seed-quotes` as well if the book should still start empty. `--seed-last-prices` only writes `null` placeholders, which is not enough.

When `--api-gateway` is enabled, `pm-config-gen` emits sensible local defaults and automatically generates one bearer token for each configured ALF gateway. Pass `--seed N` to make those generated keys reproducible. Use `--no-api-gateway-generate-keys` when you only want manually supplied `--api-key` entries.

6.6.4 Keys for many participants

A class of a hundred students needs a hundred keys. `--api-keys-for` writes them to their own file instead of into the main one:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 TRADER03 \
  --api-keys-for TRADER01 TRADER02 TRADER03 \
  --api-gateway-readonly-key \
  --seed 20260814 \
  --output engine_config.yaml
```

This writes `engine_config.yaml` with `credentials_file: api_keys.yaml` and, beside it, `api_keys.yaml` (mode `0600`) with one key per listed participant and the read-only key. It combines with `--split`, which moves the symbols and participants out as well. Without `--force` it refuses to replace an existing `api_keys.yaml`.

The keys are derived from the seed, so the same command line always gives the same keys: handy for regenerating a lost file, and a reason to keep the seed private. Without `--seed` a fixed default seed is used and the command prints a note, because anyone who knows the default and a participant ID can then work out that participant's key. That is acceptable for a throwaway classroom exchange and nowhere else.

Use repeated `--api-gateway-instance` options when you want separate API gateway processes for logical separation. A non-null `gateway_id` can belong to only one generated API gateway entry. An instance with no gateway list uses the `NAME::PORT` form and is suitable for a dashboard or other read-only service. For example, this gives the desk process credentials for every participant and the dashboard process one read-only credential:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 MM01:MARKET_MAKER OPS01:ADMIN \
  --api-gateway-instance desk:TRADER01,TRADER02,MM01,OPS01:8080 \
  --api-gateway-instance dashboards::8081 \
  --api-gateway-readonly-key \
  --seed 20260814 \
  --output engine_config.yaml
```

With named instances, `--api-gateway-readonly-key` is generated only for identity-free instances such as `dashboards::8081`; it is not added to the identity-bound `desk` instance. Read-only `gateway_id: null` credentials may be repeated across named instances.

Typical CLI example for a local lab with RALF enabled:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --sessions-enabled \
  --post-trade-gateway \
  --post-trade-bind-address 127.0.0.1 \
  --post-trade-port 5580 \
  --post-trade-replay-retention-sec 3600 \
  --post-trade-heartbeat-interval-sec 1 \
  --post-trade-idle-timeout-sec 10 \
  --post-trade-max-client-queue 2000 \
  --post-trade-allowed-roles CLEARING AUDIT \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --output engine_config.yaml
```

This generates a standard engine config plus a top-level `post_trade_gateway` block for `pm-ralf-gwy`. Use `127.0.0.1` for a single-host lab; switch to a controlled network bind such as `0.0.0.0` only when external clients must connect from other machines.

6.6.5 `--participants` format

Each gateway token is:

```
ID[:ROLE[:DISCONNECT]]
```

Examples:

- `TRADER01`
- `MM01:MARKET_MAKER`
- `OPS01:ADMIN:LEAVE_ALL`

`ID` must start with an uppercase letter and contain only `A-Z`, `0-9` and `_`, at most 32 characters. Anything else is an error, including lowercase: `pm-config-gen` does not uppercase IDs for you.

A description is not part of the spec. Set it with the separate, repeatable `--participant-desc GW_ID:TEXT`, and quote it when the text has spaces:

```
--participant-desc "MM01:Primary market maker"
```

The split is deliberate. A description inside the spec would have to be quoted too, and when the quotes are forgotten the shell turns `Student desk 1` into three words, which used to be accepted silently as participants `DESK` and `1`. Now the stray words are rejected, either as invalid IDs or as unrecognised arguments.

Role defaults for disconnect behavior:

Role	Default disconnect behavior
`TRADER`	`CANCEL_ALL`
`MARKET_MAKER`	`CANCEL_QUOTES_ONLY`
`ADMIN`	`LEAVE_ALL`

6.6.6 `--symbol-opts` format

Use `--symbol-opts` for per-symbol overrides:

```
SYMBOL:KEY=VALUE[,KEY=VALUE,...]
```

Example:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 MM01:MARKET_MAKER \
  --symbol-opts AAPL:tick_decimals=2,level=L1,mm_spread_ticks=8 \
  --symbol-opts MSFT:dynamic_band=0.03,cb_halt_l1=10,ace_initial_band=0.05 \
  --symbol-opts AAPL:enforce_mm_obligation=true
```

Supported `KEY` values:

Key	Value type	Effect
`tick_decimals`	int `0..8`	Override symbol tick precision
`static_band`	float `(0,1)`	Symbol collar static band
`dynamic_band`	float `(0,1)`	Symbol collar dynamic band
`max_order_qty`	int `> 0`	Symbol `order_limits.max_order_qty` cap
`max_order_value`	float `> 0`	Symbol `order_limits.max_order_value` cap (display money)
`cb_shift_l1` / `cb_shift_l2` / `cb_shift_l3`	float `(0,1)`	Override CB level shift pct
`cb_halt_l1` / `cb_halt_l2` / `cb_halt_l3`	int `>= 0` minutes	Override CB halt duration (`0` means rest-of-day)
`ace_enabled`	`true` / `false`	Enable or disable Automated Corridor Expansion for that symbol
`ace_initial_band`	float `(0,1)`	Symbol reopening corridor half-width

Key	Value type	Effect
`ace_random_end_ns`	int `>= 0` ns	Symbol random-end bound (`0` = predictable reopen times)
`level`	string	Symbol risk level key
`mm_spread_ticks`	int `> 0`	Symbol MM spread threshold
`mm_min_qty`	int `> 0`	Symbol MM minimum quantity
`enforce_mm_obligation`	`true` or `false`	Override per-symbol `enforce_mm_obligation` in `mm_obligation_defaults.symbols`

Four of these have explicit flags as well, which avoids the `KEY=VALUE` syntax when you are only setting one thing — the two collar bands and the two order-limit caps:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 \
  --symbol-static-band AAPL:0.18 \
  --symbol-dynamic-band AAPL:0.03 \
  --symbol-max-order-qty AAPL:50000 \
  --symbol-max-order-value AAPL:2500000
```

The caps are equivalent to `--symbol-opts "AAPL:max_order_qty=50000,max_order_value=2500000"`, and like every order limit they apply to that symbol alone — there is no risk-level or global form of them (see Risk Controls).

Per-symbol risk-level assignment can also use an explicit flag:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 \
  --risk-level CORE:0.18:0.02 \
  --risk-level HIGH_BETA:0.12:0.04 \
  --symbol-risk-level AAPL:CORE \
  --symbol-risk-level TSLA:HIGH_BETA
```

`--symbol-risk-level` is a convenience alias for `--symbol-opts SYM:level=...`. It writes `symbols.<SYM>.level` and uses the same runtime validation rules.

Unknown symbols/keys or invalid values in `--symbol-opts` are reported as warnings and ignored.

The generated `symbols:` section also includes an `outstanding_shares` field for every symbol. Use that as the slow-changing input for statistics and future index-style consumers; market capitalization can then be derived from it and the latest price instead of being stored as a separate static field.

6.6.7 `--combo` format

`--combo` seeds one `market_maker_combos` entry per flag:

```
ID:TYPE:TIF:LEG[,LEG,...]
```

Where each `LEG` is:

```
SYM/SIDE/ORDER_TYPE/QTY[/PRICE[/STOP_PRICE[/SMP_ACTION]]]
```

Part	Required	Accepted values	Default
`ID`	Yes	Non-empty string; becomes `combo_id`	—
`TYPE`	Yes	`AON`	—
`TIF`	Yes	`DAY`, `GTC`, `ATO`, `ATC`	—
`SYM`	Yes	Symbol in `--symbols`; unique within the combo	—
`SIDE`	Yes	`BUY`, `SELL`	—

Part	Required	Accepted values	Default
`ORDER_TYPE`	Yes	`LIMIT`, `MARKET`, `STOP`, `STOP_LIMIT`, `FOK`, `ICEBERG`, `IOC`, `TRAILING_STOP`	—
`QTY`	Yes	Positive integer	—
`PRICE`	No	Display price, e.g. `209.50` (see note below); omit or use `null` for market orders	`null`
`STOP_PRICE`	No	Display price for stop orders; omit or use `null` otherwise	`null`
`SMP_ACTION`	No	`NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	`NONE`

Constraints: at least 2 and at most 10 legs; each `SYM` must appear in `--symbols`; duplicate leg symbols within one combo are rejected.

PRICE and STOP_PRICE are display prices — write the decimal point

Every price in `engine_config.yaml`, including `market_maker_combos[].legs[].price` and `stop_price`, is **display money**: `209.50` means $209.50. `pm-config-gen` copies the number you give it into the file unchanged. A bare integer such as `20950` is therefore *not* read as a tick count — it is written as `20950.0`, a price of $20,950.00, which is valid on a two-decimal grid and so passes every check. Always write the price as the quote shows it, with its decimals: `AAPL/BUY/LIMIT/100/209.50`. Minimal two-leg example:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 \
  --combo "SEED-PAIR:AON:DAY:AAPL/BUY/LIMIT/100/209.50,MSFT/SELL/LIMIT/50/415.50" \
  --output engine_config.yaml
```

Generated `market_maker_combos` section:

```
market_maker_combos:
- combo_id: SEED-PAIR
  combo_type: AON
  tif: DAY
  legs:
  - symbol: AAPL
    side: BUY
    order_type: LIMIT
    quantity: 100
    price: 209.5
    stop_price: null
    smp_action: NONE
  - symbol: MSFT
    side: SELL
    order_type: LIMIT
    quantity: 50
    price: 415.5
    stop_price: null
    smp_action: NONE
```

Multiple combos use repeated `--combo` flags:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 \
  --combo "PAIR-AM:AON:DAY:AAPL/BUY/LIMIT/100/209.50,MSFT/SELL/LIMIT/50/415.50" \
  --combo "PAIR-AT:AON:DAY:AAPL/BUY/LIMIT/100/209.50,TSLA/SELL/LIMIT/20/248.00" \
  --output engine_config.yaml
```

6.6.8 Practical recipes

Minimal classroom config:

```
pm-config-gen \
  --symbols AAPL \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --no-sessions-enabled \
  --output engine_config.yaml
```

Session-driven day with risk levels and CB ladder:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --outstanding-shares TSLA:3200000000 \
  --sessions-enabled \
  --risk-level L1:0.30:0.05 \
  --risk-level L2:0.20:0.02 \
  --cb-levels L1:0.07:5 L2:0.13:15 L3:0.20 \
  --output engine_config.yaml
```

Market-maker session with seeded startup quotes:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 MM01:MARKET_MAKER OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --sessions-enabled \
  --enforce-mm-obligations \
  --seed 20260621 \
  --seed-mm-mid-range 20:300 \
  --seed-last-prices-from-mm \
  --output engine_config.yaml
```

After generation, validate manually:

```
poetry run python -c 'from pathlib import Path; from edumatcher.engine.config_loader import load_engine_config; print(load_engine_conf
```

If MM gateways are present and you do not use `--seed-mm-mid-range`, fill all `market_maker_quotes` prices first, then run the validation command.

Post-trade gateway config with explicit RALF listener settings:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --post-trade-gateway \
  --post-trade-bind-address 127.0.0.1 \
  --post-trade-port 5580 \
  --post-trade-allowed-roles CLEARING AUDIT \
  --output engine_config.yaml
```

Expected emitted section:

```
post_trade_gateway:
  name: ralf-gwy01
  bind_address: 127.0.0.1
  port: 5580
  replay_retention_sec: 86400
  heartbeat_interval_sec: 2
  idle_timeout_sec: 20
  max_connections: 64
  max_client_queue: 10000
  allowed_roles:
  - CLEARING
  - AUDIT
```

This is the quickest path when you want one command that prepares both:

- the engine symbol and ALF gateway config used by `pm-engine`
- the optional RALF listener settings used by `pm-ralf-gwy`

REST/WebSocket API gateway config with generated keys:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --api-gateway \
  --api-gateway-readonly-key \
  --api-gateway-host 0.0.0.0 \
  --api-gateway-port 8080 \
  --seed 20260624 \
  --output engine_config.yaml
```

Expected emitted section shape:

```
api_gateways:
  default:
    enabled: true
    host: 0.0.0.0
    port: 8080
    swagger_enabled: true
    log_level: info
    stats_db: data/stats.db
    order_retention_sec: 3600
    credentials:
    - api_key: key-trader01-...
      gateway_id: TRADER01
      description: Generated key for TRADER01
    - api_key: key-trader02-...
      gateway_id: TRADER02
      description: Generated key for TRADER02
    - api_key: key-ops01-...
      gateway_id: OPS01
      description: Generated key for OPS01
    - api_key: key-readonly-...
      gateway_id: null
      description: Generated read-only market-data key
    rate_limit:
      writes_per_second: 10
      burst: 20
    timeouts:
      engine_auth_sec: 3.0
      engine_reply_sec: 3.0
      wait_ack_sec: 3.0
```

Explicit API-key config:

```
pm-config-gen \
  --symbols AAPL \
  --participants TRADER01 \
  --api-key trader-secret:TRADER01:"Desk app" \
  --api-key dashboard-secret:null:"Read-only dashboard" \
  --no-api-gateway-generate-keys \
  --output engine_config.yaml
```

`gateway_id` values in API credentials must either be `null` for read-only market-data access or match an ID from `participants`. Generated keys are plain YAML bearer tokens for local labs and teaching setups; production deployments should manage secrets with the surrounding platform and terminate TLS in front of `pm-api-gwy`.

For multiple generated processes, start a specific named entry with `pm-api-gwy --instance NAME`.

BALF gateway config with explicit settings:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --balf-gateway \
  --balf-bind-address 127.0.0.1 \
  --balf-port 5560 \
  --balf-duplicate-session-policy EVICT_OLD \
  --output engine_config.yaml
```

Expected emitted section:

```
balf_gateway:
  name: balf-gwy01
  bind_address: 127.0.0.1
  port: 5560
  heartbeat_interval_sec: 1
  heartbeat_timeout_sec: 5
  idle_timeout_sec: 30
  auth_timeout_sec: 10
  max_connections: 64
  max_client_queue: 10000
  max_messages_per_second: 100
  max_errors_before_disconnect: 10
  error_window_sec: 60
  duplicate_session_policy: EVICT_OLD
```

Index calculation config with `pm-index`:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 OPS01:ADMIN MM01:MARKET_MAKER \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --outstanding-shares TSLA:3200000000 \
  --seed-mm-mid-range 100:400 \
  --seed-last-prices-from-mm \
  --no-mm-seed-quotes \
  --sessions-enabled \
  --index EDU100:"EduMatcher broad index" \
  --index-constituents EDU100:AAPL,MSFT,TSLA \
  --output engine_config.yaml
```

Expected emitted section shape:

```
indices:
- id: EDU100
  description: EduMatcher broad index
  base_value: 1000.0
  publish_interval_sec: 1.0
  history_file: data/indexes/EDU100_history.jsonl
  state_file: data/indexes/EDU100_state.json
  constituents:
  - AAPL
  - MSFT
  - TSLA
```

With multiple indices and custom settings:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 OPS01:ADMIN MM01:MARKET_MAKER \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --outstanding-shares TSLA:3200000000 \
  --seed-mm-mid-range 100:400 \
  --seed-last-prices-from-mm \
  --no-mm-seed-quotes \
  --index TECH2:"Technology pair" \
  --index-constituents TECH2:AAPL,MSFT \
  --index-base-value TECH2:500.0 \
  --index-interval TECH2:2.0 \
  --index VOLAT1:"High-beta watch" \
  --index-constituents VOLAT1:TSLA \
  --output engine_config.yaml
```

Startup combo seeds with two pairs:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --outstanding-shares AAPL:15400000000 \
  --outstanding-shares MSFT:7430000000 \
  --outstanding-shares TSLA:3200000000 \
  --sessions-enabled \
  --combo "SEED-AM:AON:DAY:AAPL/BUY/LIMIT/100/209.50,MSFT/SELL/LIMIT/50/415.50" \
  --combo "SEED-AT:AON:DAY:AAPL/BUY/LIMIT/100/209.50,TSLA/SELL/LIMIT/20/248.00" \
  --output engine_config.yaml
```

Custom circuit-breaker ladder in place of the built-in defaults:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 OPS01:ADMIN \
  --sessions-enabled \
  --cb-levels L1:0.07:5 L2:0.13:15 L3:0.20 \
  --output engine_config.yaml
```

Per-symbol reopening-auction band override while using global defaults for the circuit-breaker levels:

```
pm-config-gen \
  --symbols AAPL TSLA \
  --participants TRADER01 OPS01:ADMIN \
  --cb-levels L1:0.07:5 L2:0.13:15 L3:0.20 \
  --symbol-opts TSLA:ace_initial_band=0.05 \
  --output engine_config.yaml
```

Gateway description labels and per-symbol MM obligation override:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01:TRADER:CANCEL_ALL TRADER02:TRADER:CANCEL_ALL \
    MM01:MARKET_MAKER:CANCEL_QUOTES_ONLY OPS01:ADMIN:LEAVE_ALL \
  --participant-desc "TRADER01:Student desk 1" \
  --participant-desc "TRADER02:Student desk 2" \
  --participant-desc "MM01:Primary market maker" \
  --participant-desc "OPS01:Instructor console" \
  --enforce-mm-obligations \
  --symbol-opts AAPL:enforce_mm_obligation=true,mm_spread_ticks=8 \
  --symbol-opts MSFT:enforce_mm_obligation=false \
  --seed-mm-mid-range 20:300 \
  --mm-seed-spread-ticks 4 \
  --seed 20260706 \
  --sessions-enabled \
  --output engine_config.yaml
```

This uses `enforce_mm_obligation=false` on MSFT to disable the check for that symbol only, while leaving it enabled globally. `--mm-seed-spread-ticks 4` makes the seeded quotes 8 ticks wide, so they satisfy AAPL's `mm_spread_ticks=8`; with the default half-spread of 10 ticks the seeds would be 20 ticks wide, and `pm-config-gen` would refuse to write the file. Gateway descriptions appear in the generated YAML as the `description` field on each `participants` entry.

6.7 Compile Configs with `pm-config-deploy`

`pm-config-deploy` is the bridge between the file you author and the compiled artifact the exchange runs (see File Location). It:

1. **validates** the authored file with all four `pm-cverifier` layers, so a configuration nobody checked can no longer reach a running exchange;
2. **resolves every default exactly once**, rather than in the eight loaders that used to hold their own copies and could drift apart;
3. **installs** the result atomically, alongside a copy of the source it was built from.

```
pm-config-deploy my_config.yaml         # validate, compile and install
pm-config-deploy --check my_config.yaml # validate only, install nothing
pm-config-deploy --show                 # where do the deployed files live?

pm-engine --verbose
pm-scheduler
```

The artifact is not a reformatted copy of your YAML. A source naming two keys compiles to ten fully-resolved sections (`engine`, the six gateway and service sections, `log_client`, `monitor` and `api_gateways`) plus a `meta` block: a `market_data_gateway` block you never wrote still arrives with all of its fields, which is what lets each process deserialise rather than decide.

Deployment replaces the running configuration but does not disturb live processes; restart them to pick it up.

6.7.1 Add a symbol with `pm-new-symbol`

To list a new symbol (an IPO) in the running configuration, stop the exchange and use `pm-new-symbol` (alias `pm-ipo`). It edits the deployed configuration's authored source, validates it, and redeploys it:

```
pm-new-symbol --symbol NEWCO --ipo-price 20.00 --outstanding-shares 50000000
```

See Listing a New Symbol for the full procedure.

6.7.2 Deploying example configurations

By using the option `--example` it is possible to deploy one of the example configurations supplied as examples in an easy way.

The available examples are all located in the directory `docs/examples/ref_data/<SPEC>/engine_config.yaml` and are as follows

Directory	`--example` shorthand	Profile	Number of symbols	Session enabled	MM seed quotes
`s1-basic-setup`	`s1-basic`	basic	1	no	yes
`s1-nominal-setup`	`s1-nominal`	nominal	1	yes	yes
`s1-complex-setup`	`s1-complex`	complex	1	yes	yes
`s3-basic-setup`	`s3-basic`	basic	3	no	yes
`s3-nominal-setup`	`s3-nominal`	nominal	3	yes	yes
`s3-complex-setup`	`s3-complex`	complex	3	yes	yes
`s10-basic-setup`	`s10-basic`	basic	10	no	yes
`s10-nominal-setup`	`s10-nominal`	nominal	10	yes	yes
`s10-complex-setup`	`s10-complex`	complex	10	yes	yes
`s30-basic-setup`	`s30-basic`	basic	30	no	yes
`s30-nominal-setup`	`s30-nominal`	nominal	30	yes	yes
`s30-complex-setup`	`s30-complex`	complex	30	yes	yes
`s150-basic-setup`	`s150-basic`	basic	150	no	yes
`s150-nominal-setup`	`s150-nominal`	nominal	150	yes	yes
`s150-complex-setup`	`s150-complex`	complex	150	yes	yes
`s1-basic-nomm-setup`	`s1-basic-nomm`	basic	1	no	no
`s1-nominal-nomm-setup`	`s1-nominal-nomm`	nominal	1	yes	no
`s1-complex-nomm-setup`	`s1-complex-nomm`	complex	1	yes	no
`s3-basic-nomm-setup`	`s3-basic-nomm`	basic	3	no	no
`s3-nominal-nomm-setup`	`s3-nominal-nomm`	nominal	3	yes	no
`s3-complex-nomm-setup`	`s3-complex-nomm`	complex	3	yes	no
`s10-basic-nomm-setup`	`s10-basic-nomm`	basic	10	no	no
`s10-nominal-nomm-setup`	`s10-nominal-nomm`	nominal	10	yes	no
`s10-complex-nomm-setup`	`s10-complex-nomm`	complex	10	yes	no
`s30-basic-nomm-setup`	`s30-basic-nomm`	basic	30	no	no
`s30-nominal-nomm-setup`	`s30-nominal-nomm`	nominal	30	yes	no
`s30-complex-nomm-setup`	`s30-complex-nomm`	complex	30	yes	no
`s150-basic-nomm-setup`	`s150-basic-nomm`	basic	150	no	no
`s150-nominal-nomm-setup`	`s150-nominal-nomm`	nominal	150	yes	no
`s150-complex-nomm-setup`	`s150-complex-nomm`	complex	150	yes	no
`s300-load-setup`	`s300-load`	load	300	yes	yes
`s300-sim-setup`	`s300-sim`	sim	300	yes	yes

The shorthand is always `s<count>-<profile>`, where `<count>` is one of `1`, `3`, `10`, `30`, or `150` and `<profile>` is one of `basic`, `nominal`, or `complex`. The exceptions are `s300-load`, the AI-trader load test bed (300 symbols, participants `AI001`–`AI500`), and `s300-sim`, the long-running simulated market (300 symbols, `AI001`–`AI100`, with a `market_sim.yaml`); neither has a profile or a `-nomm` variant. Any other shorthand also accepts an optional trailing `-nomm` (`s<count>-<profile>-nomm`, e.g. `s3-basic-nomm`) to deploy the no-market-maker-quotes variant of that same example — the `MARKET_MAKER` gateway is still present so a market maker can connect and quote, but the book starts with no seeded `market_maker_quotes` at all, so the very first order rests on an empty book instead of matching immediately. See docs/concepts/03-concepts-mm-quotes.md for why real markets always need an opening quote (even on IPO day) while this teaching exchange lets you opt out of one. The three profiles differ as follows:

Profile	Gateways	Sessions and schedule	Auxiliary blocks	Risk controls
basic	4 (2 `TRADER`, 1 `MARKET_MAKER`, 1 `ADMIN`)	disabled — starts in `CONTINUOUS`	none	engine defaults only
nominal	4 (2 `TRADER`, 1 `MARKET_MAKER`, 1 `ADMIN`)	enabled, with a full `schedule`	`post_trade_gateway`, `market_data_gateway`, `api_gateways` (`desk` + `dashboards`)	engine defaults only
complex	8 (5 `TRADER`, 2 `MARKET_MAKER`, 1 `ADMIN`)	enabled, with a full `schedule`	same as nominal, plus `market_maker_combos`	named `risk_controls` levels and a `circuit_breaker_defaults` ladder

Pick `basic` for a quick matching demo with no scheduler, `nominal` for a realistic single-desk session with the market-data, post-trade, and REST gateways available, and `complex` when you need multiple desks, two market makers, startup combo seeds, and explicit collar/circuit-breaker policy.

For example

```
pm-config-deploy --example s3-basic
```

will validate, compile, and install `docs/examples/ref_data/s3-basic-setup/engine_config.yaml` as the deployed `ref_data/engine_confi` artifact, exactly as if you had passed that path as `SOURCE`. The same shorthand works with `pm-setup --config`. Appending `-nomm`, as in `pm-config-deploy --example s3-basic-nomm` or `pm-setup --config s3-basic-nomm`, deploys `docs/examples/ref_data/s3-bas` instead.

6.7.3 What the artifact records about itself

```
"meta": {
  "schema_version": 5,
  "compiler_version": "0.66.3",
  "compiled_at": "2026-10-31T23:19:37.000Z",
  "source_path": "/Users/you/course/engine_config.yaml",
  "source_sha256": "e3bc2f14cf5c10da…",
  "content_sha256": "87be4dc0b9b645cb…"
}
```

The two digests answer different questions, and both are checked:

- `source_sha256` — *has the authored file changed since this was built?* Each process warns at startup when it has, so an edit you forgot to deploy is visible rather than silently ignored.
- `content_sha256` — *has this file changed since it was built?* Recomputed on every load. Editing the deployed artifact by hand is refused, naming `pm-config-deploy` as the way to make the change properly.

The payload digest detects modification, not malice: it travels inside the file it protects, so anyone who edits the payload can recompute it. Proving provenance rather than integrity would need a signature.

`schema_version` guards against a build reading an artifact shaped for another; an unknown version is refused with a message telling you to recompile.

If you are running from a Poetry checkout, prefix commands with `poetry run`.

6.7.4 Missing-file Behavior

Neither process accepts a config path on the command line — both always read the deployed artifact — and they handle a missing artifact differently:

Process	No deployed artifact
`pm-engine`	Starts unrestricted
`pm-scheduler`	Fatal error — refuses to start

Unrestricted engine mode means there is no symbol allowlist, no gateway allowlist, no configured risk levels, no seeded last prices, no seeded market-maker quotes, no configured startup combos, and no outstanding share metadata.

`pm-scheduler` treats a missing artifact as fatal because running a timetable the engine has never seen is worse than not starting at all. This is different from a *deployed* config with session scheduling disabled (`sessions_enabled: false` or no `schedule` section): in that case the scheduler starts normally and falls back to its built-in default schedule.

6.8 Inspect Configs with `pm-config-show`

`pm-config-show` prints the effective configuration as a terminal dashboard and exits. Where `pm-cverifier` answers *is this file correct*, `pm-config-show` answers *what does this file actually say* — a question that is surprisingly hard to answer by reading the YAML, because a deployed config is 800–1500 lines of which perhaps 150 are data.

```
pm-config-show                      # the deployed config, essentials only
pm-config-show -m                   # denser: risk, breakers, gateway tuning
pm-config-show -a                   # everything, API keys unmasked
pm-config-show -f my_config.yaml    # an authored file, before deploying it
pm-config-show --format pdf -o exchange.pdf
```

With no arguments it reads `<DATA_DIR>/ref_data/engine_config.yaml` — the same file File Location describes — so what you see is what the exchange was configured from. `pm-config-show` is **read-only**: it never writes to the configuration or the data directory, and `--output` is the only path it ever creates.

Three questions dominate day-to-day use, and the layout is built around them.

Which ports are in use, and what binds each one? This is the one thing that cannot be read off the file at all. Three engine sockets and two index sockets are compiled into `config.py` and appear nowhere in the YAML; and a gateway section present *without* a `port:` key still binds, on its runtime default. The ports panel shows all of them together with the process, the function, and where the value came from — `fixed`, `env`, `set` or `default` — and flags any port claimed twice in red, the same condition `pm-cverifier` reports as `M018`.

What are the API keys? They exist to be copied, so a key is never wrapped or truncated at any width, and no styling is applied inside the token, which lets a terminal double-click select the whole thing. Keys are masked by default; `--all` reveals them. Masked and revealed keys are the same length, so revealing never moves the layout.

What instruments are configured? The symbol list reflows into as many side-by-side sub-tables as the width allows, reading alphabetically *down* each column like a printed index.

```
┌─ ENGINE CONFIGURATION ──────────────────────────────────────────────────────┐
│ docs/examples/ref_data/s10-nominal-setup/engine_config.yaml                 │
│ 42.7 kB  ·  2026-10-11 20:27  ·  via --file                                 │
│ ● on sessions   ○ off collars   ● on breakers   ● on mm-oblig               │
│ 10 symbols   4 participants   2 API gateways   5 keys   17 listeners        │
└─────────────────────────────────────────────────────────────────────────────┘

┌─ PORTS & LISTENERS ─────────────────────────────────────────────────────────┐
│  PORT   PROTO      PROCESS       FUNCTION                        BIND       │
│ ─────────────────────────────────────────────────────────────────────────── │
│  5553   ZMQ PUB    pm-market-…   True values and heartbeat       127.0.0.1  env     │
│  5554   ZMQ PULL   pm-market-…   Instructor commands             127.0.0.1  env     │
│  5555   ZMQ PULL   pm-engine     Order intake                    127.0.0.1  fixed   │
│  5556   ZMQ PUB    pm-engine     Event + book feed               127.0.0.1  fixed   │
│  5557   ZMQ PUB    pm-engine     Drop-copy feed                  127.0.0.1  fixed   │
│  5558   ZMQ PUB    pm-index      Index value publish             127.0.0.1  env     │
│  5559   ZMQ PULL   pm-index      Index command intake            127.0.0.1  env     │
│  5561   ZMQ REP    pm-stats      Health ping                     127.0.0.1  env     │
│  5562   ZMQ REP    pm-audit      Health ping                     127.0.0.1  env     │
│  5563   ZMQ REP    pm-clearing   Health ping                     127.0.0.1  env     │
│  5564   ZMQ REP    pm-index      Health ping                     127.0.0.1  env     │
│  5570   TCP        pm-md-gwy     Market data (CALF)              0.0.0.0    set     │
│  5580   TCP        pm-ralf-gwy   Post-trade (RALF)               0.0.0.0    set     │
│  5610   ZMQ PULL   pm-monitor    Health beacons (in)             127.0.0.1  default │
│  5611   ZMQ PUB    pm-monitor    Health snapshot feed            127.0.0.1  default │
│  8080   HTTP       pm-api-gwy    REST API — desk                 0.0.0.0    set     │
│  8081   HTTP       pm-api-gwy    REST API — dashboards           0.0.0.0    set     │
└─────────────────────────────────────────────────────────────────────────────┘

┌─ API KEYS ──────────────────────────────────────────────────────────────────┐
│ GATEWAY ID      API GW        ROLE             API KEY                      │
│ ─────────────────────────────────────────────────────────────────────────── │
│ TRADER01        desk          TRADER           key-trader01-••••••••••••••••••••••••••••g6u1  │
│ TRADER02        desk          TRADER           key-trader02-••••••••••••••••••••••••••••y09z  │
│ OPS01           desk          ADMIN            key-ops01-••••••••••••••••••••••••••••oes2     │
│ MM01            desk          MARKET_MAKER     key-mm01-••••••••••••••••••••••••••••1o3s      │
│ —               dashboards    READ-ONLY        key-readonly-••••••••••••••••••••••••••nrjd    │
│ masked — run with -a/--all to reveal                                        │
└─────────────────────────────────────────────────────────────────────────────┘
```

6.8.1 Adapting to the terminal

The layout is computed from the terminal you actually have; there is no fixed column count. Panels are packed side by side where they fit, short panels are stacked beside tall ones so no gutter is left empty, and individual tables shed optional columns as they narrow. In round terms:

Terminal	What you get
below 72 columns or 18 rows	A plain summary: filename, counts, flags, and the port map. No boxes.
72–99 columns	Single column; wide tables drop their optional columns.
100–169 columns	Two columns, sometimes three where panels are narrow.
170 columns and up	Three columns; symbols reflow to four or five sub-tables.

At the default density only, the output is also trimmed to fit the window height: optional panels are dropped first, then the symbol list is shortened. Both trims say so, and name the flag that brings the content back. Passing `-m` or `-a` disables height trimming entirely — asking for more information is taken as accepting that you will scroll.

6.8.2 Options

Option	Meaning
`-f, --file YAML`	Config file to read. Default: `<DATA_DIR>/ref_data/engine_config.yaml`.
`-m, --density [1|2]`	Pack more in. Bare `-m` means `1`. Adds collars, circuit breakers, gateway tuning and combo seeds at `1`; engine tuning, indices, the reopening ladder and per-symbol override markers at `2`.
`-a, --all`	Everything: implies `-m 2`, unmasks API keys, and lists unrecognised top-level keys.
`--format {terminal,pdf}`	Output format. Default `terminal`.
`-o, --output FILE`	Destination for `--format pdf`. Defaults to `engine-config-<stem>.pdf`.
`--no-color`	Suppress ANSI colour. Also implied when stdout is not a TTY, or when `NO_COLOR` is set.
`--ascii`	ASCII box drawing. Auto-enabled on non-UTF-8 terminals.
`--width N, --height N`	Force render dimensions, for piping or scripted capture.

Density is a *layout* control and `--all` is a *disclosure* control, which is why they are separate flags: `-m 2` shows every setting but keeps keys masked, so it stays safe to run on a projector.

Exit codes are `0` on success, `2` when the file is missing or unreadable, and `3` when the YAML does not parse. A parse failure prints the parser's message and points at `pm-cverifier` rather than attempting partial recovery.

6.8.3 PDF output

`--format pdf` renders the same content to A4 landscape across several pages: an overview with the full port table and the session schedule, an access page with participants and credentials, a risk and market-making page, then as many symbol pages as the universe needs, and an appendix of tuning and indices. Every page repeats the four global enforcement flags in its header and carries `page N of M`, so a page printed on its own still says whether collars were on.

```
pm-config-show --format pdf -o handout.pdf             # keys masked
pm-config-show --format pdf --all -o operations.pdf    # keys in full
```

A PDF made with –all contains live credentials

Masking is per-run, not per-file. Prefer the masked form for anything you hand out or print for a class.

6.9 Choosing a Starting Point

Three real configurations used to be listed here as "minimal", "medium" and "fully complex". They are now built up step by step, with every parameter explained and every listing verified, in The Configuration File, Block by Block. Pick the step that matches the exchange you want to run:

You want…	Start from	Then add
A matching demo with no scheduler and no liquidity	Step 1 — `symbols`, `participants`, `sessions_enabled: false`	reference prices (Step 2)
A classroom day with auctions	Step 3	collars and breakers (Step 4)
Books that are never empty	Step 5	an index (Step 7)
External clients, dashboards, logs	Step 8 and Step 9	—
A ready-made file for a class of a given size	Deploying example configurations	edit and redeploy

The complete file at the end of that chapter is the reference "full" configuration.

6.10 Configuration Checklist

Use this checklist when creating a new engine configuration.

1. Decide session mode. Use `sessions_enabled: false` for simple demos and tests. Use `sessions_enabled: true` when `pm-scheduler` should drive phases.
2. Define the symbol universe. Add every tradable symbol under `symbols`, set `tick_decimals`, and add `last_buy_price` / `last_sell_price` so collars and circuit breakers have a reference from the first order. Add `outstanding_shares` for index constituents.
3. Define the participants. Add every expected `pm-alf-console --id ...` (and every BALF and API identity) under `participants`. Choose `TRADER`, `MARKET_MAKER`, or `ADMIN`, then choose disconnect behavior. Include at least one `ADMIN`.
4. Decide whether market makers exist. If no gateway has `role: MARKET_MAKER`, `market_maker_quotes` are optional. If any gateway has `role: MARKET_MAKER`, every symbol needs at least one quote seed. Quote seed `gateway_id` values must reference configured `MARKET_MAKER` gateways.
5. Add risk controls only as needed. Use `risk_controls.levels` for reusable collar profiles, `circuit_breaker_defaults` for the global breaker ladder, and symbol-level overrides only for exceptions.
6. Add market-maker obligation policy if quote quality matters. Start with `mm_obligation_defaults`, override by symbol under `mm_obligation_defaults.symbols`, and use `participants[*].mm_obligations` only for gateway-specific exceptions.
7. Add startup combos only after symbols are stable. Keep combo leg symbols unique within one combo and use 2 to 10 legs. Leg prices are display money (`209.50`), like every other price in the file.
8. Add index calculations if needed. Define each `pm-index` process in the `indices` block. Every constituent must appear in `symbols:` with a positive `outstanding_shares`. Run `pm-index` for each configured index; `pm-config-gen --index` generates the block automatically.
9. Add a schedule if sessions are enabled. Every schedule block must contain all five keys. Times are `HH:MM` in `locale.session_timezo` (default `Europe/Stockholm`), not the host's clock; set `locale` to match where your class is.
10. Check persistence before first run. Remove stale state when changing seed behavior or symbol universe, especially `book_stats.json`, `gtc_orders.json`, and `gtc_combos.json` in the data directory (`src/data/` in a source checkout, `~/.local/share/e` when installed, or `$EDUMATCHER_DATA_DIR` if set).
11. Validate before class or demo. Run `pm-cverifier`, deploy with `pm-config-deploy`, start `pm-engine --verbose`, connect each gateway ID you expect to use, and run `SYMBOLS` from a gateway.

6.11 Startup and Persistence Order

The effective engine startup sequence is:

```
Engine startup
    |
    +-- 1. Load the deployed (compiled) configuration, if there is one
    +-- 2. Bind the main PULL (5555) and PUB (5556) sockets
    +-- 3. Restore persisted GTC orders and combos from
    |      <DATA_DIR>/gtc_orders.json and <DATA_DIR>/gtc_combos.json
    +-- 4. Load persisted book stats from <DATA_DIR>/book_stats.json, seed the
    |      books' last prices, and build collars and circuit breakers from them
    +-- 5. Inject market_maker_quotes (a seed_once quote is skipped when its
    |      gateway already has a restored live quote on the symbol)
    +-- 6. Inject market_maker_combos
    +-- 7. Publish the starting collar bands and book checkpoints
    +-- 8. Bind the drop-copy PUB socket (5557)
    +-- 9. Start serving
```

`<DATA_DIR>` resolves to `src/data/` in a source checkout, `~/.local/share/edumatcher` when installed, or `$EDUMATCHER_DATA_DIR` if that environment variable is set.

This ordering means persisted GTC state comes back before config seed liquidity, and persisted book stats override `last_buy_price` / `last_sell_price` seeds. When changing seed behavior or symbol definitions, consider removing stale data:

```
# from a source checkout:
rm -f src/data/gtc_orders.json src/data/book_stats.json src/data/gtc_combos.json
# or, if EDUMATCHER_DATA_DIR is set:
rm -f "$EDUMATCHER_DATA_DIR"/gtc_orders.json "$EDUMATCHER_DATA_DIR"/book_stats.json "$EDUMATCHER_DATA_DIR"/gtc_combos.json
```

6.12 Adding or Removing Symbols

Adding a symbol is the configuration equivalent of an **IPO (initial listing)**: you define the instrument together with its opening reference price (`last_buy_price` / `last_sell_price`), its issued share count (`outstanding_shares`), and — when a market maker is configured — its opening quote. Those opening values seed the book's last prices and both the collar and circuit-breaker references, so the symbol is priced and protected from its very first order (see Risk Controls - Day one (IPO) behaviour).

The symbol universe is fixed at startup

`pm-engine` reads `symbols` once, at startup. There is **no** command to list a new symbol intra-day. Introducing a symbol always means editing `engine_config.yaml` and restarting the engine — plan the full instrument set before the session, or restart during a maintenance window to add a new listing.

Edit `engine_config.yaml` and restart the engine.

- adding a symbol makes it tradable on next startup
- removing a symbol causes future orders for it to be rejected
- persisted GTC orders for removed symbols are skipped during restore
- startup combo seeds referencing removed symbols make config loading fail
- `mm_obligation_defaults.symbols.<SYMBOL>` entries must reference configured symbols

6.13 Validation Commands

Before a configuration reaches a running exchange, check it in this order:

```
pm-cverifier engine_config.yaml          # every layer, with a Risk Summary
pm-config-deploy --check engine_config.yaml   # validate and compile, install nothing
```

`pm-cverifier` reports every problem it can find; `pm-config-deploy --check` additionally compiles each section with the loader its own process uses. Neither changes anything.

For a quick parser check from a source checkout, load the file with the engine's own loader:

```
poetry run python -c 'from pathlib import Path; from edumatcher.engine.config_loader import load_engine_config; print(load_engine_cor
```

If the file is valid, this prints the parsed `EngineConfig` object. On error you get a traceback ending with a descriptive message:

```
ValueError: Engine config must have a 'symbols' mapping
```

```
ValueError: participants[0].disconnect_behaviour is invalid
```

`pm-engine` itself does not take a config path: it always loads the **deployed** artifact, so an authored file is only checked by `pm-engine` after `pm-config-deploy` has installed it.

For the focused config parser test suite:

```
poetry run pytest tests/test_config_loader.py tests/test_config_extensions.py
```

These commands answer *does it load*. To see what a file that loads actually says — ports, keys, participants, symbols — use `pm-config-show`.

6.14 Verifying the Deployed Artifact

The compiled artifact carries two independent integrity checks. They answer different questions, they fail differently, and confusing them wastes an afternoon — so they are described separately here.

Check	Compares	Severity	Runs
Content digest	the artifact against *itself*	**error** — the process refuses to start	on every load, in every process
Source staleness	the artifact against the authored YAML	**warning** — the process starts anyway	when a process reports its deployment

6.14.1 The content digest (an error)

Every artifact stores `meta.content_sha256`: a SHA-256 over all of its own sections *except* `meta`. On **every** load, `load_compiled_config()` recomputes that digest and refuses the file if it no longer matches.

The YAML is not involved. Nothing is recompiled, and the authored file is not read. The check is a *round trip of the artifact through the running build's own type definitions*:

1. parse `engine_config.json`;
2. build the `CompiledConfig` dataclass tree from it;
3. serialise that tree straight back to primitives;
4. canonicalise (sorted keys, fixed indent) and SHA-256 the result;
5. compare against the `meta.content_sha256` stored inside the same file.

Step 2 is the one that surprises people. A key **absent** from the JSON is filled from the dataclass default, and step 3 writes out **every** declared field. So what gets hashed is not the file's bytes — it is what those bytes mean to *this* build. Adding a configuration field to the code is therefore enough to change the digest of an artifact nobody has touched: the key decodes to its default and is re-emitted as `"new_field": null`, which the recorded digest never covered.

That is a completely different situation from a hand-edit, and it needs a different fix, so the loader distinguishes them **exactly** rather than by guesswork: before reporting anything it also hashes the payload *as it sits on disk*. If that still matches what was recorded, the file is provably intact and only the schema moved.

Cause 1 — the artifact predates a schema change. Nothing is wrong with the file; it was compiled by a build that declared a different set of configuration fields. Upgrading (or downgrading) EduMatcher across such a change invalidates every artifact compiled before it. This is the common cause and it is not a fault:

```
ArtifactError: compiled config is intact but was compiled against a
different configuration schema: it was written by version 0.29.2, and this
build declares a different set of configuration fields. Nothing has been
edited — run pm-config-deploy to recompile it
```

Cause 2 — the deployed `.json` really was edited after deployment. The payload no longer hashes to what the file itself records:

```
ArtifactError: compiled config has been modified since it was compiled
(payload digest 8e77158733b5… does not match the recorded 4a63ba384b6f…)
— edit the source and run pm-config-deploy rather than editing the deployed
artifact
```

An edit is the more serious finding, so an edit *on top of* a schema change reports as an edit.

The check is symmetric, and deliberately so: an artifact compiled by a *newer* build fails on an older one too, because the older build ignores the field it does not know about and re-emits a payload missing it. Neither direction can silently run a configuration it has misread.

This detects modification, not malice

The digest travels inside the file it protects, so anyone who edits the payload can recompute it. It proves the artifact is internally consistent, not that it came from a trusted party — that would need a signature over a key the artifact does not carry.

A coarser gate runs *before* the digest: `meta.schema_version` must equal the `SCHEMA_VERSION` the build reads, and a mismatch produces its own distinct error. That version is bumped only when an older reader would **misinterpret** a newer artifact; an additively-changed schema is left to the digest check, which already refuses it loudly and now names the reason.

6.14.2 The source staleness check (a warning)

The artifact also records `meta.source_path` and `meta.source_sha256` — where the YAML was read from, and the SHA-256 of its text (of the main file followed by every file it lists under `include:`, so an edit to an included file counts too). This is the check that *does* look at your YAML, and it catches the one failure compiling introduces: editing the authored file and forgetting to deploy, so the exchange keeps running the previous configuration while the file on disk says otherwise.

```
engine_config.yaml has changed since the running configuration was compiled
at 2026-09-04T09:43:28.000Z — this exchange is still running the previous
one. Run pm-config-deploy to pick up the edit.
```

It only warns. It also stays silent when the source is unreachable — a configuration compiled on another machine, or from a file since moved, is not evidence of staleness, and warning about it would train you to ignore the warning.

6.14.3 Checking both by hand

Where the two files live:

```
pm-config-deploy --show
```

To see the artifact's provenance and confirm both checks pass, from a source checkout:

```
poetry run python - <<'PY'
from edumatcher.config_artifact import (
    load_compiled_config, content_digest, staleness)

cfg = load_compiled_config()
print("compiled by :", cfg.meta.compiler_version, "at", cfg.meta.compiled_at)
print("source      :", cfg.meta.source_path)
print("recorded    :", cfg.meta.content_sha256[:16])
print("recomputed  :", content_digest(cfg)[:16])
print("stale?      :", staleness(cfg) or "no - source matches")
PY
```

Reaching the `recomputed` line at all means the digest check already passed — `load_compiled_config()` raises on a mismatch rather than returning a bad config, so the two digests can never print differently. The line is there to show you the value, not to perform the comparison.

6.14.4 Fixing a digest mismatch

Both causes have the same fix — recompile the YAML that is already deployed beside the artifact:

```
pm-config-deploy src/data/ref_data/engine_config.yaml
```

Two things to know before reaching for something else:

- **`pm-config-deploy` has no `--force`.** It takes a positional source path and always overwrites. The `--force` you may be thinking of belongs to `pm-setup`, where it means "recompile and overwrite an already-deployed configuration" — a different command with a different source.
- **Do not "fix" it with `--example`** unless you really are on a stock example. A deployed configuration generated from an example is identical to it *except* for the randomly generated `api_key` values, and redeploying from the example silently replaces every one of them.

Redeploying is always safe here: the artifact is derived data, and the YAML beside it is by construction the exact bytes that were last compiled.

6.15 See Also

In this book:

- Running the Engine - startup order and common runtime workflows
- Risk Controls - collar and circuit-breaker behavior in depth
- Persistence - how GTC orders, book stats, and combos are saved and restored

Participant Guide:

- The Trader Console (`pm-alf-console`) - ALF commands and gateway behavior

Reference Manual:

- Processes - which process reads which config section

CHAPTER 7

Configuration Verifier (`pm-cverifier`)

Learning objectives

After reading this page you will understand:

- Why `pm-cverifier` exists and what problems it solves
- How to run it from the command line and interpret its output
- What each verification layer checks and what the check codes mean
- How to use it in CI pipelines with `--strict` and `--format json`
- How to read the Risk Summary to confirm your intended risk posture

7.1 Why a Config Verifier?

`engine_config.yaml` (see The Configuration Workflow for the full guide and App Config Spec for the normative field reference) covers symbols, gateways, risk controls, collar bands, circuit breakers, market-maker obligations, session schedules, combo seeds, index definitions, and seven optional gateway subsystems (`alf_gateway`, `balf_gateway`, `post_trade_gateway`, `market_data_gateway`, `dc_gateway`, `log_server`, `api_gateways`). Getting all of this right by hand is error-prone.

The current engine loader (`load_engine_config()`) reports only the *first* hard error it encounters before aborting. This means operators spend time in a trial-and-error loop: edit the file, start the engine, read a terse `ValueError`, fix one problem, repeat. There is also no concept of advisory warnings — a valid config that quietly uses defaults the operator did not intend is accepted without comment.

`pm-cverifier` is a standalone, **read-only** tool that inspects a config file deeply and produces a human-friendly report covering:

- every YAML syntax error that would prevent the file from being parsed
- every hard error that `load_engine_config()` would raise, plus additional ones
- completeness gaps — settings that are missing but probably should be present
- advisory suggestions for settings that are valid but likely to surprise
- a plain-English **Risk Summary** that shows what risk controls are actually active
- an overall `OK` / `WARNING` / `ERROR` verdict

7.2 Quick Start

```
# verify a config file and print a human-readable report
pm-cverifier engine_config.yaml

# only show warnings and above (suppress info-level advisories)
pm-cverifier --level warn engine_config.yaml

# use in CI — fail if any warnings exist, output machine-readable JSON
pm-cverifier --strict --format json engine_config.yaml
```

Exit codes:

Code	Meaning
0	All checks passed; zero errors and zero warnings
1	One or more warnings; no errors
2	One or more hard errors

With `--strict`, any warning also produces exit code 2.

7.3 CLI Reference

`pm-cverifier --help`:

```
usage: pm-cverifier [-h] [--version] [--format {text,json}]
                    [--level {info,warn,error}] [--no-color] [--strict]
                    CONFIG_FILE

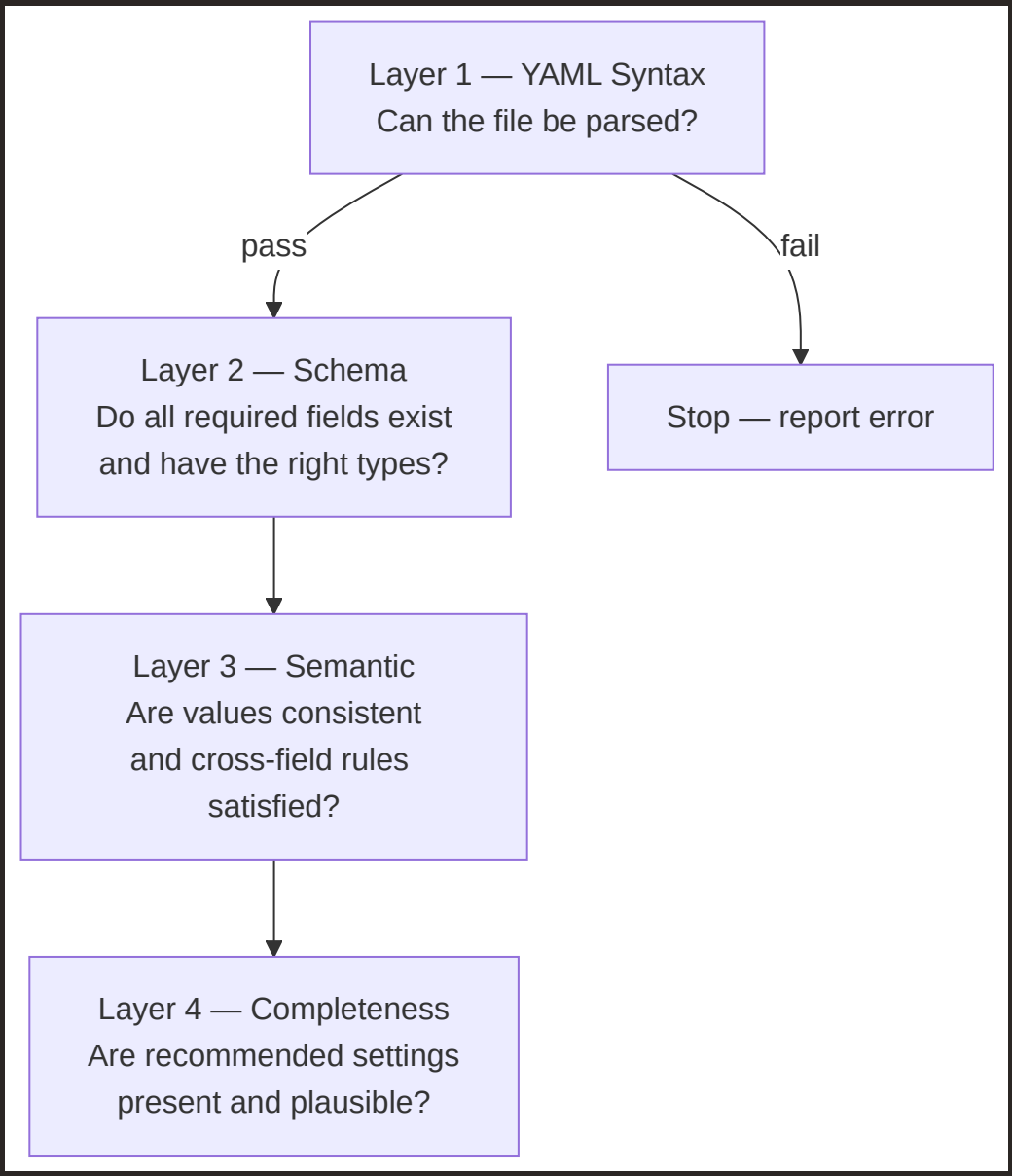
Read-only engine_config.yaml verification tool.

positional arguments:
  CONFIG_FILE           Path to engine_config.yaml

options:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --format {text,json}  Output format (default: text)
  --level {info,warn,error}
                        Minimum severity to show (default: info)
  --no-color            Disable ANSI color in text output
  --strict              Treat warnings as errors for CI exit-code purposes
```

7.4 Verification Layers

The verifier runs checks in four sequential layers. A hard error in Layer 1 (unparseable file) or in Layer 2 (schema errors) stops the later layers, which are reported as `skipped (fix schema errors first)`. Within a layer, every independent problem is reported, so a single pass surfaces as many issues as possible.



Layer	Severity	What it checks
1 — YAML Syntax	ERROR	File readable, valid YAML, top-level is a mapping
2 — Schema	ERROR	Required keys, correct types, in-range values
3 — Semantic	ERROR or WARN	Cross-field consistency, referential integrity
4 — Completeness	WARN or INFO	Missing-but-recommended settings, default-value notices

7.5 Understanding the Output

7.5.1 Text output (default)

```
pm-cverifier engine_config.yaml

✓ YAML syntax: OK
✓ Schema: 0 errors
⚠ Semantic: 1 warning
i Completeness: 2 advisories

──────────────────────────────────────────────────────────
Warnings
──────────────────────────────────────────────────────────

[M013] WARN  No gateway has role: ADMIN.
  → Without an admin gateway, halt, resume, kill-switch, and emergency
    commands cannot be issued at runtime. Add a gateway with role: ADMIN:
    - id: OPS01
      role: ADMIN
      disconnect_behaviour: LEAVE_ALL

──────────────────────────────────────────────────────────
Risk Summary
──────────────────────────────────────────────────────────

Symbols          2  (AAPL, MSFT)
Gateways         2  (TRADER01: TRADER, MM01: MARKET_MAKER)
Sessions         disabled — always CONTINUOUS
Collars          enabled — DEFAULT: static=20%, dynamic=2%
Order limits     none configured
Circuit breakers enabled — enabled but no circuit breaker configured ⚠
MM obligations   not enforced
Admin gateway    none ⚠

──────────────────────────────────────────────────────────
Verdict:  ⚠ 1 WARNING, 2 ADVISORIES — engine can start but review warnings
──────────────────────────────────────────────────────────
```

Every finding has:

1. A **check code** (e.g. `M013`) — use this to look up the exact condition in the check catalogue below.
2. A **severity** — `ERROR`, `WARN`, or `INFO`.
3. A **message** — one sentence naming the problem and the exact field.
4. A **suggestion** — a concrete fix, including a YAML snippet where useful.

7.5.2 JSON output (`--format json`)

```
{
  "file": "engine_config.yaml",
  "verdict": "WARN",
  "summary": { "errors": 0, "warnings": 1, "info": 2 },
  "checks": [
    {
      "code": "M013",
      "severity": "WARN",
      "message": "No gateway has role: ADMIN.",
      "suggestion": "...",
      "path": "participants"
    }
  ],
  "risk_summary": { ... }
}
```

The JSON output is stable and suitable for parsing in CI pipelines or custom reporting scripts.

7.6 Check Catalogue

7.6.1 Layer 1 — YAML Syntax (`Y001`–`Y005`)

Code	Condition
`Y001`	File not found
`Y002`	File is not readable
`Y003`	YAML parse error
`Y004`	Top-level document is not a mapping
`Y005`	An `include:` list is invalid or an included file is missing, unreadable, not valid YAML or clashes with another file; the message names the file (see Splitting the file)

7.6.2 Layer 2 — Schema (`S001`–`S131`)

Top-level structure

Code	Condition
`S001`	`symbols` absent or not a mapping
`S002`	`participants` absent or not a list
`S004`	`symbols` is empty
`S005`	`participants` is empty

Symbol fields

Code	Condition
`S010`	`tick_decimals` not an integer in `0..8`
`S011`	`last_buy_price` or `last_sell_price` not numeric
`S012`	`outstanding_shares` not a positive integer
`S013`	`level` references an undefined risk level
`S014`	`market_maker_quotes[n]` missing required fields

Code	Condition
S015	market_maker_quotes[n].bid_price >= ask_price
S016	market_maker_quotes[n] has a non-numeric price, non-positive/non-integer quantity, or invalid tif
S017	market_maker_quotes is present but not a list
S018	market_maker_quotes[n] is not a mapping
S019	market_maker_quotes[n].gateway_id is blank when provided

Symbol collar / circuit-breaker overrides

symbols.<SYMBOL>.collar and symbols.<SYMBOL>.circuit_breaker.levels are inline per-symbol overrides that the engine loader validates just as strictly as the top-level risk_controls.levels.*.collar and circuit_breaker_defaults.levels sections (S030–S034, S041–S042) — these codes cover the per-symbol equivalents. order_limits has no level-scope equivalent at all: S114–S116 are the only value checks, and S117 refuses the key if it appears on a risk level.

Code	Condition
S036	symbols.<SYMBOL>.collar present but not a mapping
S037	symbols.<SYMBOL>.collar.static_band_pct not in (0, 1)
S038	symbols.<SYMBOL>.collar.dynamic_band_pct not in (0, 1)
S114	symbols.<SYMBOL>.order_limits present but not a mapping
S115	symbols.<SYMBOL>.order_limits.max_order_qty not a positive integer
S116	symbols.<SYMBOL>.order_limits.max_order_value not a positive number
S065	symbols.<SYMBOL>.circuit_breaker or .levels not a mapping — **also reused** for locale.country present but not a non-empty string (unrelated condition, same code)
S122	locale present but not a mapping
S123	locale.session_timezone present but not a non-empty string
S066	symbols.<SYMBOL>.circuit_breaker.levels.<LEVEL> missing price_shift_pct
S067	symbols.<SYMBOL>.circuit_breaker.levels.<LEVEL>.price_shift_pct out of range (0, 1)
S068	symbols.<SYMBOL>.circuit_breaker.levels.<LEVEL>.halt_duration_ns not a positive integer

Gateway fields

Code	Condition
S020	Gateway entry missing id
S021	Duplicate gateway ID
S022	role is not a recognised value
S023	disconnect_behaviour is not a recognised value
S024	quote_refresh_policy is not a recognised value
S086	smp_action is not a recognised value
S025	enforce_mm_obligation is not a boolean
S026	mm_max_spread_ticks or mm_min_qty invalid
S027	mm_obligations is present but not a mapping
S028	mm_obligations.<symbol> entry is invalid
S029	participants[n] is not a mapping
S084	Two participants IDs are prefixes of each other

Code	Condition
S118	participant_defaults is not a mapping, or has a key other than smp_action / disconnect_behaviour
S119	participant_defaults.disconnect_behaviour is not a recognised value
S120	participant_defaults.smp_action is not a recognised value
S121	A key that no loader reads, at any level of the file: a top-level key, a key in any engine section (symbols, participants, mm_obligation_defaults, engine_tuning, locale, risk_controls, circuit_breaker_defaults, market_maker_combos, indices, a schedule day block, and their nested blocks), or a key in a gateway or service block (alf_gateway, balf_gateway, market_data_gateway, post_trade_gateway, dc_gateway, log_server, monitor, or an api_gateways.<name> instance and its nested blocks). The loaders ignore such a key, so the setting silently has no effect; the verifier reports the closest accepted name

Circuit breaker fields

Code	Condition
S030	circuit_breaker_defaults or .levels not a mapping
S031	CB level missing price_shift_pct
S032	price_shift_pct out of range (0, 1)
S033	halt_duration_ns not a positive integer
S035	circuit_breaker present inside a risk level

Risk controls

Code	Condition
S040	risk_controls.default_level references an undefined level
S041	collar.static_band_pct not in (0, 1)
S042	collar.dynamic_band_pct not in (0, 1)
S117	risk_controls.levels.<LEVEL>.order_limits is present — order limits are a per-symbol control and are not supported here

Indices (indices)

Code	Condition
S043	indices is present but not a list
S044	indices[n] is not a mapping
S045	indices[n].id invalid (empty, non-alnum, or duplicate)
S046	indices[n].description is empty or not a string
S047	indices[n].base_value or publish_interval_sec invalid
S048	indices[n].history_file or state_file invalid
S049	indices[n].constituents invalid shape or duplicates

Combo seeds (market_maker_combos)

Code	Condition
S055	market_maker_combos is present but not a list

Code	Condition
S056	market_maker_combos[n] invalid mapping/combo_id
S057	market_maker_combos[n].combo_type or .tif invalid
S058	market_maker_combos[n].legs invalid shape/length
S059	market_maker_combos[n].legs[m] invalid, duplicate, or unknown symbol

Runtime / schedule / top-level flags

Code	Condition
S060	sessions_enabled present but not a boolean
S061	engine_tuning.snapshot_interval_sec present but not a positive number
S062	enforce_collars present but not a boolean
S063	enforce_circuit_breakers present but not a boolean
S064	schedule present but not a mapping
S113	require_mm_seed_quotes present but not a boolean

Reopening / Automated Corridor Expansion (circuit_breaker.reopening)

Applied to both circuit_breaker_defaults.reopening and each symbols.<SYM>.circuit_breaker.reopening.

Code	Condition
S104	reopening is present but not a mapping
S105	reopening.enabled is not a boolean
S106	reopening.initial_band_pct is outside (0, 1)
S107	reopening.expansions is not a non-empty list, or an entry is not a mapping
S108	reopening.expansions[].widen_pct is missing or outside (0, 1)
S109	reopening.expansions[].min_duration_ns is missing or not > 0
S110	reopening.random_seed is set on a symbol — it is engine-wide and belongs in circuit_breaker_defaults
S111	reopening.random_end_max_ns is not an integer >= 0
S112	reopening.expansions is set on a symbol — the ladder is exchange-wide and belongs in circuit_breaker_defaults

S110 and S112 are errors rather than warnings because both keys would otherwise be accepted, stored and then ignored. The random-end generator is one engine-wide instance, and the expansion ladder is read from the exchange defaults, so in each case the symbol would appear configured while behaving exactly as if it were not.

MM obligation defaults (mm_obligation_defaults)

Code	Condition
S070	mm_obligation_defaults is present but not a mapping
S071	mm_obligation_defaults.enforce_mm_obligation is not a boolean
S072	mm_obligation_defaults.mm_max_spread_ticks invalid
S073	mm_obligation_defaults.mm_min_qty invalid
S074	mm_obligation_defaults.symbols is present but not a mapping
S075	mm_obligation_defaults.symbols.<symbol> is not a mapping

Code	Condition
S076	mm_obligation_defaults.symbols.<symbol>.enforce_mm_obligation invalid
S077	mm_obligation_defaults.symbols.<symbol>.mm_max_spread_ticks/min_qty invalid

Price tick grid

Code	Condition
S078	A price is not a whole multiple of its symbol's tick size

S078 covers every price in the file — last_buy_price, last_sell_price, market_maker_quotes[n].bid_price/ask_price, and market_maker_combos[n].legs[m].price/stop_price. Each is display money the engine converts to integer ticks as it loads, and it refuses one it cannot represent rather than rounding it. A combo leg is checked against its own symbol's tick_decimals, not the combo's first leg: the legs of one combo trade different instruments, which need not share a tick size. The check stands down for a symbol whose tick_decimals is itself invalid — S010 reports that, and there is no grid to check against until it is fixed.

API gateway sections

Code	Condition
S080	api_gateways section fails runtime-loader validation

S080 also covers the case where the config uses the unsupported singular key api_gateway instead of api_gateways — see the App Config Spec (CV15), which documents that rejection as a normative rule.

The verifier accepts multiple named API gateway entries, including a separate read-only dashboard entry generated from pm-config-gen with --api-gateway-instance dashboards::8081. In the resulting YAML, dashboards.credentials[].gateway_id: null is valid and is intentionally ignored by the M022 gateway-reference check. Non-null credentials must match an ID under participants, and each non-null ID may belong to only one named api_gateways entry. M018 also checks every named API gateway port for collisions with the other configured listeners.

For example, this generated shape is valid:

```
api_gateways:
  desk:
    port: 8080
    credentials:
      - api_key: key-trader01-example
        gateway_id: TRADER01
  dashboards:
    port: 8081
    credentials:
      - api_key: key-dashboard-example
        gateway_id: null
```

ALF, RALF, and market-data gateway sections

Code	Condition
S081	alf_gateway section fails runtime-loader validation
S082	post_trade_gateway section fails runtime-loader validation
S083	market_data_gateway section fails runtime-loader validation

BALF gateway fields (balf_gateway)

Code	Condition
S050	balf_gateway is present but not a mapping
S051	balf_gateway.port not an integer in 1..65535
S052	BALF capacity fields not positive integers (max_connections, etc.)
S053	BALF timeout/interval fields not positive numbers
S054	balf_gateway.duplicate_session_policy not REJECT_NEW or EVICT_OLD
S085	balf_gateway fails runtime-loader validation for a reason not covered by S050–S054 (drift safety net; only fires when none of those already reported)

Drop-copy gateway fields (dc_gateway)

Code	Condition
S090	dc_gateway is present but not a mapping
S091	dc_gateway.port not an integer in 1..65535
S092	dc_gateway.name or .bind_address blank or not a string
S093	dc_gateway.max_connections or .max_client_queue not a positive integer
S094	dc_gateway.heartbeat_interval_sec or .idle_timeout_sec not a positive number

Log server fields (log_server)

Code	Condition
S095	log_server is present but not a mapping
S096	log_server.enabled or .pubsub_enabled present but not a boolean
S097	log_server.port, .pub_port, or .pull_port not an integer in 1..65535
S098	log_server.name, .bind_address, or .db_path blank or not a string
S099	log_server.max_message_bytes, .max_client_queue, .write_batch_size, .write_batch_interval_ms, .heartbeat_interval_sec, .lease_sec, .max_lease_sec, .max_subscribers, .notify_interval_ms, .backfill_chunk_rows, .max_backfill_minutes, .max_backfill_rows, .max_pending_rows, or .pub_sndhwm not a positive integer
S100	log_server.retention_days present but not a non-negative integer (or null)
S101	log_server fails runtime-loader validation for a reason not covered by S095–S103 (drift safety net; only fires when none of those already reported)
S102	Two of log_server.port, .pub_port, .pull_port resolve to the same port
S103	log_server.max_lease_sec is below log_server.lease_sec

log_server.retention_days follows the same convention documented in Configuring pm-log-srv: 0 and null both mean unbounded retention, so S100 only fires on a negative value or a non-integer type, never on 0.

S102 and S103 cover the two LALF-PS rules that involve more than one field, and both are conditions pm-log-srv refuses to start on:

- **S102** — pm-log-srv binds three listeners: LALF/TCP on port, and the LALF-PS ZeroMQ PUB/PULL sockets on pub_port/pull_p The check compares *effective* values, applying each field's own default when the key is omitted, so writing pub_port: 5600 collides with the default LALF port just as surely as two explicit duplicates would.
- **S103** — max_lease_sec is the ceiling applied to a subscriber's requested lease_sec. A ceiling below the server's own default lease would make that default unreachable, which is incoherent rather than merely unusual.

Monitoring (monitor section)

The optional `monitor` block configures health beacons and `pm-monitor` (operator dashboard). An absent block means defaults with monitoring on.

Code	Condition
S124	`monitor` present but not a mapping
S125	`monitor.enabled` not a boolean (a quoted `"false"` is a string and would otherwise leave monitoring on)
S126	`monitor.pull_port` or `monitor.pub_port` not an integer in 1–65535
S127	`monitor.bind_address` or `monitor.host` not a non-empty string
S128	`monitor.beacon_interval_sec`, `hung_after_sec` or `book_metrics_interval_sec` not a positive number, or `silent_after_beacons` not a positive integer
S129	`monitor.retention` not a mapping, or `raw_hours`/`minute_days` not a positive integer
S130	`monitor.thresholds` not a mapping, a rule not a `{warn, crit}` mapping, or `warn`/`crit` missing or not a non-negative number
S131	`monitor` fails runtime-loader validation for a reason not covered by S124–S130 (drift safety net; only fires when none of those already reported)

The rules that relate two `monitor` fields are semantic checks (M029, M030, M031); a `monitor` port that collides with another listener, or `pull_port` equal to `pub_port`, is M018.

See Health Monitoring for what these fields do.

7.6.3 Layer 3 — Semantic (M001–M033)

M014 is currently emitted during the schema pass because CB threshold ordering is validated while parsing `circuit_breaker_defaults`.

Code	Severity	Condition
M001	ERROR	MM gateway present but a symbol has no seed quotes (skipped when `require_mm_seed_quotes: false`)
M002	ERROR	MM seed references a gateway ID not in `participants`
M003	WARN	MM seed spread exceeds the `mm_max_spread_ticks` that applies to that gateway and symbol (participant `mm_obligations`, then `mm_obligation_defaults.symbols`, then the participant's own value, then `mm_obligation_defaults`, then 20)
M004	INFO	Sessions enabled (the default) but no `schedule`: the built-in weekday timetable is used
M005	WARN	`sessions_enabled: false` but a `schedule` is present
M006	WARN	Schedule times not in chronological order
M007	WARN	`enforce_collars: false` while collars are defined
M008	WARN	`enforce_circuit_breakers: false` while CB levels are defined
M009	ERROR	Index constituent not in `symbols`
M010	ERROR	Index constituent missing `outstanding_shares`
M011	ERROR	More than 5 indices defined
M012	WARN	Combo uses `tif: GTC`
M013	WARN	No ADMIN gateway configured
M014	WARN	CB level thresholds not strictly increasing
M015	ERROR	Combo leg references symbol not in `symbols`

Code	Severity	Condition
M016	WARN	post_trade_gateway configured but no ADMIN gateway
M017	WARN	balf_gateway.heartbeat_timeout_sec <= heartbeat_interval_sec
M018	ERROR	Two configured listeners are bound to the same port. Covers alf_gateway, balf_gateway, post_trade_gateway, market_data_gateway, dc_gateway, every api_gateways.<name>, monitor.pull_port/pub_port (counted even when the monitor block is absent, since pm-monitor then binds 5610/5611; skipped only for enabled: false), and all three of log_server's own ports — port (LALF/TCP) plus pub_port/pull_port (LALF-PS), the latter two skipped when pubsub_enabled: false since a disabled interface binds nothing. Each of these is also checked against the fixed listeners that are not configured in engine_config.yaml — the engine's 5555–5557, pm-market-sim's 5553/5554, pm-index's 5558/5559 and the health ping ports 5561–5564 — using any EDUMATCHER_*_PORT override in the verifier's environment
M019	ERROR	mm_obligation_defaults.symbols references an unknown symbol
M020	ERROR	MM seed gateway_id exists but is not a MARKET_MAKER gateway
M021	ERROR	A schedule time value isn't a quoted "HH:MM" string (e.g. YAML mis-parsed it as a sexagesimal integer)
M022	ERROR	API credential gateway_id is not defined in participants
M023	ERROR	A schedule time value doesn't parse as a valid 24-hour HH:MM time
M024	ERROR	sessions_enabled: true but schedule is missing one or more of the five phase keys
M025	ERROR	schedule phases don't form a legal transition chain starting from CLOSED
M026	WARN	locale.country is not recognised by the holidays package (pm-scheduler falls back to Sweden at runtime)
M027	ERROR	schedule has an unknown top-level key (valid keys: weekdays, mon, tue, wed, thu, fri, sat, sun, weekend, holidays)
M028	ERROR	schedule specifies both weekend and an individual sat/sun block — use one or the other
M029	ERROR	A monitor.thresholds band is in the wrong order: warn must be strictly below crit, except disk_free_pct, where a lower value is worse and warn must be strictly above crit
M030	ERROR	monitor.hung_after_sec is not greater than monitor.beacon_interval_sec
M031	WARN	monitor.bind_address is neither a wildcard nor monitor.host (loopback on both sides counts as a match): every process sends its beacons to host, where pm-monitor may not be listening. A warning because EDUMATCHER_GATEWAY_BIND_HOST overrides the bind address at run time
M032	WARN	No read-only API credential (gateway_id: null) under api_gateways: pm-dash has no session schedule, terminal-gui no history and book-gui no data
M033	ERROR	idle_timeout_sec is not more than twice heartbeat_interval_sec in post_trade_gateway, market_data_gateway or dc_gateway (an omitted field is checked at its default). The gateway's own heartbeats keep a reading client from timing out, so a timeout this close to the interval drops healthy quiet sessions

7.6.4 Layer 4 — Completeness (C001–C013)

C010 is emitted during the semantic pass (Layer 3) but kept in the C code family because it is an operational-completeness warning. C005 is reserved and not currently emitted: an MM gateway with no mm_obligation_defaults at all is deliberately left quiet (confirmed by test_c005_mm_gw_no_obligation_defaults_is_quiet in the test suite) — only C006 (obligations defined but enforce_mm_obligation: false) fires.

Code	Severity	Condition
C001	WARN	No symbol has a reference price
C002	INFO	Symbol has only one of last_buy_price / last_sell_price
C003	INFO	enforce_collars: true but no collar configured
C004	INFO	enforce_circuit_breakers: true but no circuit breaker is configured (no circuit_breaker_defaults, no symbol circuit_breaker): no symbol will be halted
C006	WARN	enforce_mm_obligation: false
C007	INFO	engine_tuning.snapshot_interval_sec is at the default 0.5
C008	WARN	Index constituent has no reference price
C009	INFO	sessions_enabled: false and no schedule (always-on)
C010	WARN	disconnect_behaviour: LEAVE_ALL on a non-ADMIN gateway
C011	INFO	Risk level defined but no symbol uses it
C012	WARN	More than 20 symbols with snapshot_interval_sec < 0.2
C013	WARN	Index history_file or state_file is below a directory pm-index cannot create (relative paths are resolved under the data directory)

7.7 Using in CI

```
# pre-flight check in a deployment script
pm-cverifier --strict --format json engine_config.yaml
echo "Config OK"

# GitHub Actions step example
- name: Verify engine config
  run: pm-cverifier --strict --format json engine_config.yaml
```

With --strict, exit code 2 is returned for any warning, so CI will fail fast rather than silently deploying a misconfigured exchange.

7.8 The Risk Summary

The Risk Summary is always printed at the end of the text report (and included in the risk_summary key of JSON output). It answers: *"What does this config actually do at runtime?"*

Field	Description
Symbols	Total count and names
Gateways	Total count, ID, and role for each
Sessions	Enabled/disabled, schedule summary
Collars	Whether enforced and which levels are configured
Order limits	Per-symbol max order qty/value, or "none configured"
Circuit breakers	Whether enforced and threshold summary; reads "no circuit breaker configured " when neither circuit_breaker_defaults nor any symbol circuit_breaker block exists
MM obligations	Whether enforcement is active
Admin gateway	Which gateway (if any) has role ADMIN
Indices	Index IDs if any are configured

A next to a Risk Summary line marks a potentially risky or incomplete subsystem configuration (for example missing ADMIN gateway or missing collar configuration), even if the overall verdict is OK.

The JSON `risk_summary` object carries a few additional machine-readable fields not shown in the text table above, including `circuit_breakers_configured` (`false` when neither a `circuit_breaker_defaults` block nor a symbol `circuit_breaker` block exists, so no symbol will be halted).

7.9 Relationship to Other Tools

Tool	Purpose
`pm-config-gen` (see The Configuration Workflow)	*Generate* an `engine_config.yaml` from CLI flags
Config GUI (see Config GUI)	*Generate/edit* an `engine_config.yaml` visually, with a built-in "Verify with pm-cverifier" action
`pm-cverifier`	*Verify* an existing config file before use
`pm-engine` (see Running the Engine)	*Load* the config and start the matching engine

`pm-cverifier` is read-only and has no effect on the engine or any runtime state. It is safe to run at any time, including while the engine is running.

CHAPTER 8

Configuration GUI (`config-gui`)

Learning objectives

After reading this page you will understand:

- What the Config Builder GUI is and how it relates to `pm-config-gen`, `pm-cverifier`, and the engine itself
- Every way to run the application — a single container command, a pre-built image artifact, or a local development setup — and what each one needs
- How the interface is organised: the three-pane layout, personas, tabs, and the live diagnostics drawer
- The core exchange concepts each tab configures, with links to the reference chapters that explain them in depth
- How to add symbols with the IPO dialog, configure market makers, and read the effective-value Symbol Overview
- How to import, validate, and export a configuration
- How to deploy the GUI in production and troubleshoot common problems

8.1 Overview

The **Config Builder** (`config-gui`) is a browser-based application for creating, importing, editing, validating, and exporting an `engine_config.yaml` file interactively.

It is a **companion to**, not a replacement for, the `pm-config-gen` CLI. Both produce the same file format, but they suit different situations:

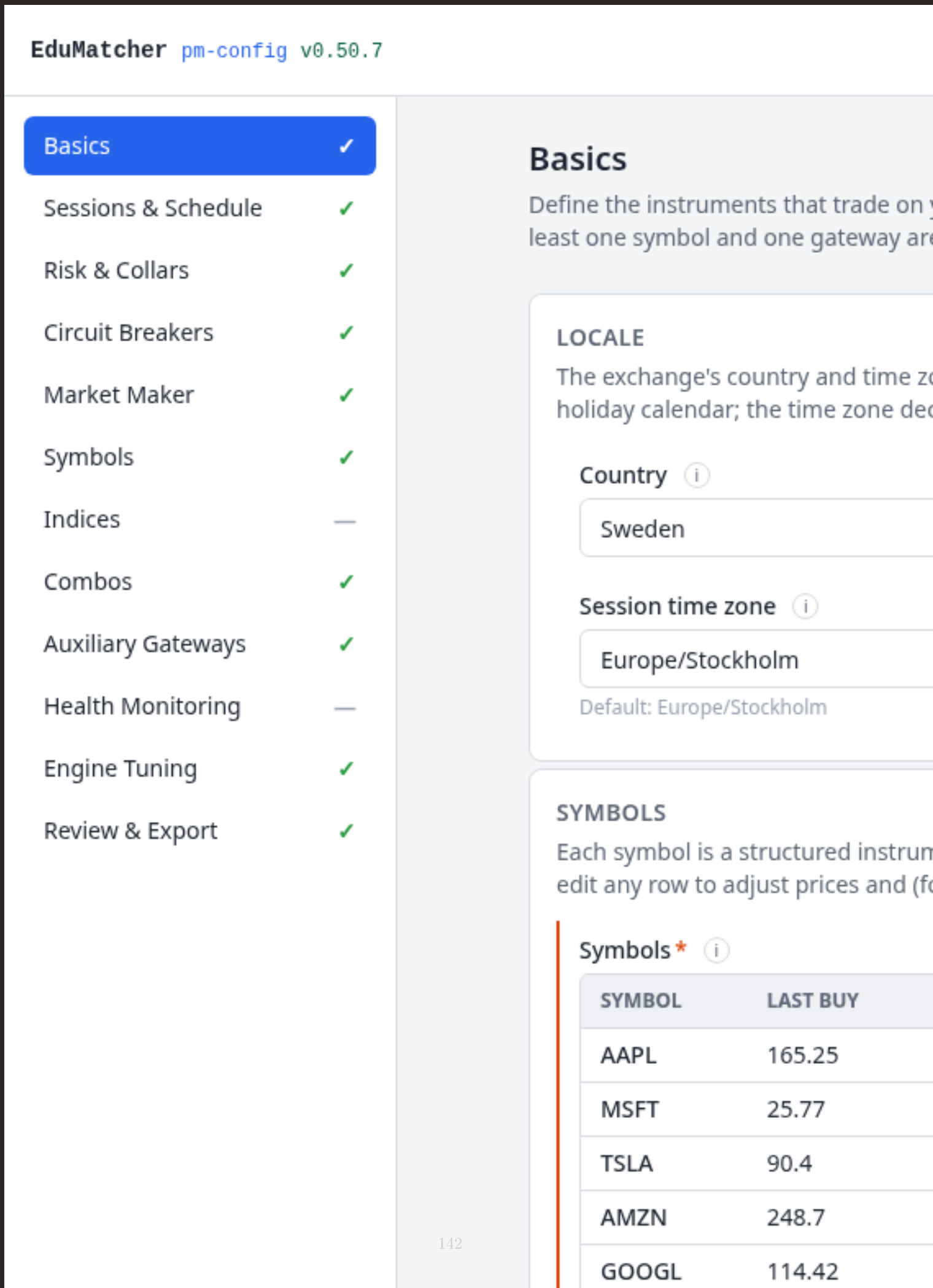
Use the…	when you want to…
GUI (`config-gui`)	build a config by filling in forms, see mistakes highlighted as you type, import and edit an existing file, or learn the schema through inline help.
CLI (`pm-config-gen`)	script config generation in CI, or drive it from a shell with repeatable flags.

The GUI adds three things the CLI cannot offer: **live cross-field validation** (undefined risk levels, port collisions, out-of-order schedules, and more), **progressive disclosure** so beginners see only a handful of fields while experts see everything, and a full **round-trip import** of existing configs.

Where it lives

The application is a self-contained Node.js/TypeScript project in the `web-apps/config-gui/` directory. It is independent of the Python engine and needs no Python to run — with one optional exception, the server-side verification feature.
Its output is held to the same correctness bar as the CLI: a golden-file test pipes generated configs through the engine's own `load_engine_config()` parser, so anything the GUI exports is guaranteed to load (see How the GUI stays valid).

*The `s10-complex` example imported at the Expert persona. The top bar holds the persona switcher, **Comment defaults**, **Import**, **New**, the theme toggle and **Save**; the navigation on the left shows every tab's status glyph; the Basics form fills the centre; the diagnostics drawer on the right reports no issues.*


EduMatcher pm-config v0.50.7
Basics ✓
Sessions & Schedule ✓
Risk & Collars ✓
Circuit Breakers ✓
Market Maker ✓
Symbols ✓
Indices —
Combos ✓
Auxiliary Gateways ✓
Health Monitoring —
Engine Tuning ✓
Review & Export ✓
Basics
Define the instruments that trade on
least one symbol and one gateway ar
LOCALE
The exchange's country and time z
holiday calendar; the time zone de
Country
Sweden
Session time zone
Europe/Stockholm
Default: Europe/Stockholm
SYMBOLS
Each symbol is a structured instrum
edit any row to adjust prices and (f
Symbols *
SYMBOL
LAST BUY
AAPL
165.25
MSFT
25.77
TSLA
90.4
AMZN
248.7
GOOGL
114.42

8.2 Prerequisites

What you need depends on how you intend to run the application.

8.2.1 Running the container (recommended for most users)

Requirement	Notes
Podman 4 or Docker 24	Podman is preferred; Docker works equally well. The `make up` target auto-detects which one is installed.
Compose plugin	`podman compose` (Podman 5+ built-in, or `podman-compose`) or `docker compose` (the V2 plugin — not the legacy `docker-compose` binary). Verify with `docker compose version` / `podman compose version`.
GNU Make	Only needed for the `Makefile` targets. On macOS, install via Homebrew (`brew install make`) or the Xcode Command Line Tools.
macOS only	A Podman machine must be running. `make up` starts it automatically; to do it manually run `podman machine init && podman machine start`.

No Node.js, npm, or Python is required on the host for the container path — every build step happens inside the image.

8.2.2 Local development

Requirement	Notes
Node.js 20	Developed and tested on Node 22 and Node 26. Install via nvm, fnm, or your OS package manager.
npm 10	Bundled with Node.js 20. Verify with `npm --version`.
GNU Make	Optional but convenient for the `Makefile` targets.

8.2.3 Optional: server-side verification

Needed only if you want the **Verify with pm-cverifier** button on the Review tab to work:

Requirement	Notes
Python environment	The repository's Poetry environment installed at the repo root (`poetry install`).
`pm-cverifier` on `PATH`	Provided by the Poetry env, or set `CVERIFIER_COMMAND="poetry run pm-cverifier"` so the backend can locate it. See Config Verifier.

8.2.4 Browser

Any modern browser (Chrome/Edge 110, Firefox 115, Safari 16). The application uses no browser extensions and runs fully offline once loaded.

8.3 Running the application

There are three supported ways to start the GUI, in increasing order of setup effort. Pick the one that matches your situation.

Method	Best for	Needs	URL
Container stack	Most users; production	Podman/Docker + Make	http://localhost:8092
Pre-built image artifact	Offline hosts, no repo clone	Podman/Docker only	http://localhost:8092
Local development	Contributing to the GUI itself	Node.js + npm	http://127.0.0.1:8192

8.3.1 Option 1 — One command with the container stack

If you have a repository checkout and a container runtime, this is the fastest path. From the web-apps/config-gui/ directory:

```
make up
```

make up auto-detects Podman or Docker (preferring Podman), starts the Podman machine on macOS if needed, builds the image, and starts the stack in the background. Open **http://localhost:8092** once the build completes.

```
make down     # stop and remove the stack
make logs     # follow the container log
make ps       # show stack status
make help     # list every available target
```

Prefer to drive the runtime directly? docker compose up --build (or podman compose up --build) does the same thing. Deployment details — the image layout, environment variables, and building behind a corporate proxy — are covered under Deployment and operations.

8.3.2 Option 2 — A pre-built image artifact

For a host without a repository clone — or to ship a ready-to-run image — the distributable artifact is a self-contained OCI image archive built with make cdist:

```
dist/edumatcher-config-gui-<version>.tar.xz
```

The archive contains everything the application needs at runtime (Node.js runtime, compiled frontend, and the Fastify backend) with no external dependencies. Load and run it on the target host:

```
# Podman
podman load  --input edumatcher-config-gui-<version>.tar.xz
podman run -d --name config-gui -p 8092:8092 edumatcher-config-gui:<version>

# Docker
docker load  --input edumatcher-config-gui-<version>.tar.xz
docker run -d --name config-gui -p 8092:8092 edumatcher-config-gui:<version>
```

The load command prints the exact image tag, e.g. edumatcher-config-gui:1.0.0. Open **http://localhost:8092**. Stop and remove with podman stop config-gui && podman rm config-gui (or the docker equivalents). Pass any backend environment variable with -e, for example -e CORS_ORIGIN="https://myhost.example".

No Node.js or npm required

The tar archive is fully self-contained. The host only needs Podman or Docker — no Node.js, no npm, and no repository clone.

8.3.3 Option 3 — Local development

To hack on the GUI, run the two processes (the Fastify **API** and the Vite **web** dev server) from a checkout:

```
cd web-apps/config-gui
npm install
npm run dev          # starts the API (port 5192) and the web UI (port 8192)
```

Open **http://127.0.0.1:8192**. The Vite dev server hot-reloads on save and proxies /api/* to the backend, so you only ever open the 8192 URL.

For separate logs, run the processes in two terminals with npm run dev:server and npm run dev:web (or make dev-server /

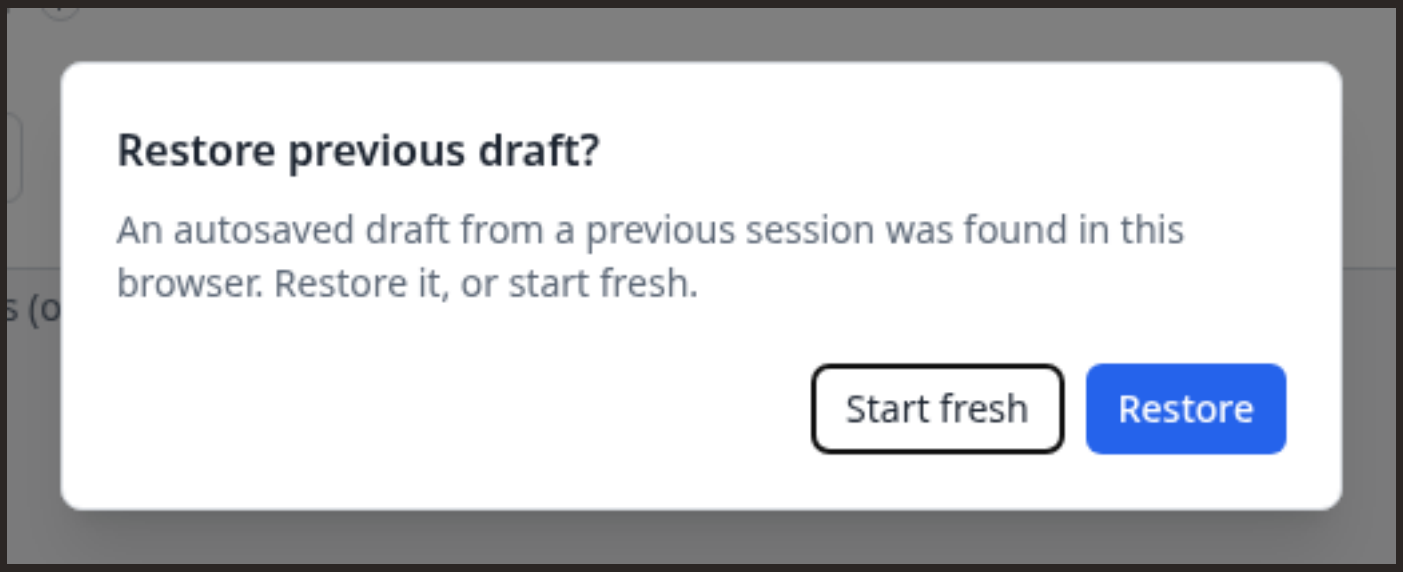


Figure 8.2: The restore-draft dialog.

`make dev-web`). Developer-facing details — project layout, tests, and how to keep the GUI in sync with the engine — live in the project's `config-gui/README.md`.

8.4 A tour of the interface

The window is a persistent **three-pane layout** designed so that you always know what to edit, what you are editing, and what still needs attention.

```
┌──────────────────────────────────────────────────────────┐
│ Top bar: [Persona ▾] [□ Comment defaults] Import New ☾ Save │
├──────────────┬──────────────────────────────┬────────────┤
│ Navigation   │        Active panel          │ Diagnostics │
│ (tab list    │   (fields for the selected   │ drawer      │
│  with status │    tab / master–detail editor) │ (errors,   │
│  glyphs)     │                              │  warnings, info) │
└──────────────┴──────────────────────────────┴────────────┘
```

- **Top bar** — the persona switcher, **Comment defaults** (add the commented reference block of every optional field at its default to the exported file — `pm-config-gen --comment-default-config-fields`; the same setting as the Review tab's toggle), **Import** (load an existing YAML file), **New** (start a blank draft, with a confirmation), the light/dark **theme toggle** (/), and **Save** (shows a confirmation toast; the draft is already autosaved to `localStorage` continuously on every change, so a refresh or crash never loses work — clicking Save doesn't write anything that wasn't already saved).
- **Navigation** — the list of tabs. Each carries a status glyph: ✓ valid, ! has warnings, ✗ has errors, – optional and untouched. Tabs above your current persona are hidden entirely, not disabled.
- **Active panel** — the form for the selected tab. Large tabs (Symbols, Auxiliary Gateways) use a **master–detail** sub-layout: a list on the left, an editor for the selected item on the right.
- **Diagnostics drawer** — always visible; it lists *every* current issue across the *whole* configuration, not just the active tab, each with a **Jump →** action that navigates to and highlights the offending field(s). See Diagnostics and live validation.

Because the draft is autosaved, opening the application again in the same browser offers it back before anything else happens: **Restore** reloads it, **Start fresh** begins a blank draft.

Shown on start-up when this browser holds an autosaved draft, over the blank draft it would otherwise begin with.

Mandatory fields carry a red asterisk and a red left border; optional fields show their default as ghost placeholder text until you set them. Every non-obvious field has an **(i) help popover** giving a one-paragraph explanation and the equivalent `pm-config-gen` flag.

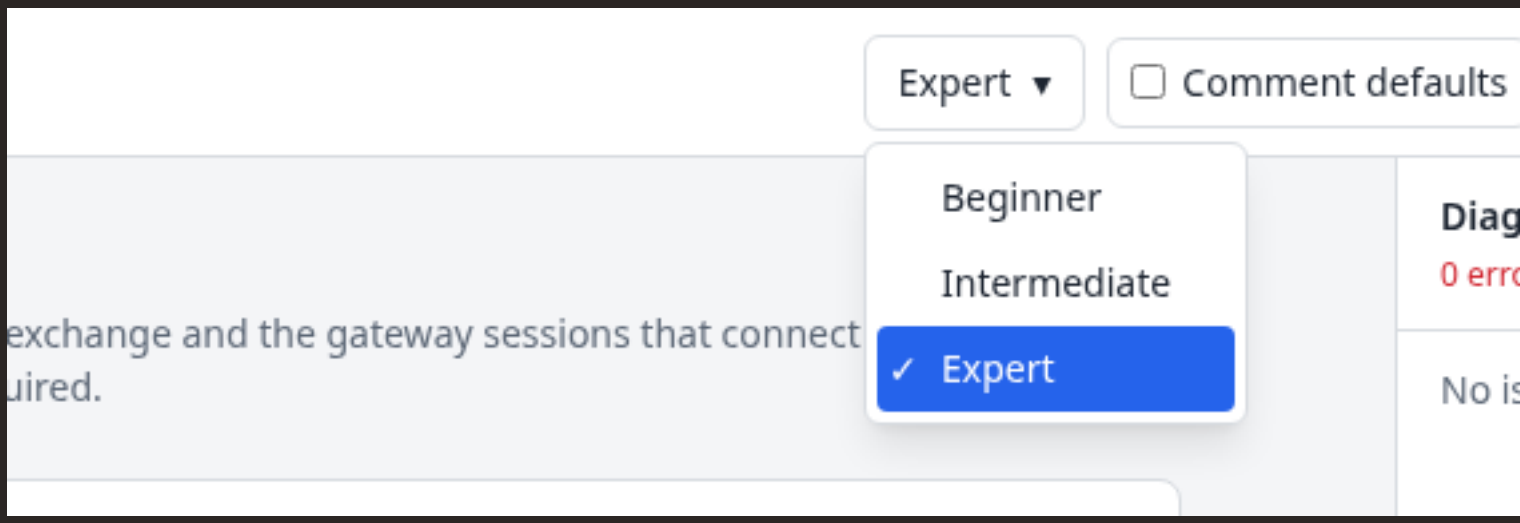


Figure 8.3: The persona switcher, open.

8.5 Personas

A single global **persona** setting controls how much of the field surface is visible. It is a *view filter*, never a data filter: switching personas never discards data, and a higher-tier value set by import (or by a previous session at a higher tier) is preserved and still exported even while hidden.

Persona	Audience	Philosophy
Beginner	A first-time student standing up a classroom exchange	Show only what a minimal, safe exchange needs. Everything else uses safe defaults.
Intermediate	A TA or instructor tuning a scenario	Adds risk levels, circuit-breaker tuning, per-symbol overrides, indices, and the common auxiliary gateways.
Expert	A course author building the full environment	Everything: combos, per-gateway market-maker overrides, BALF/API gateways, engine tuning, and deterministic seeding.

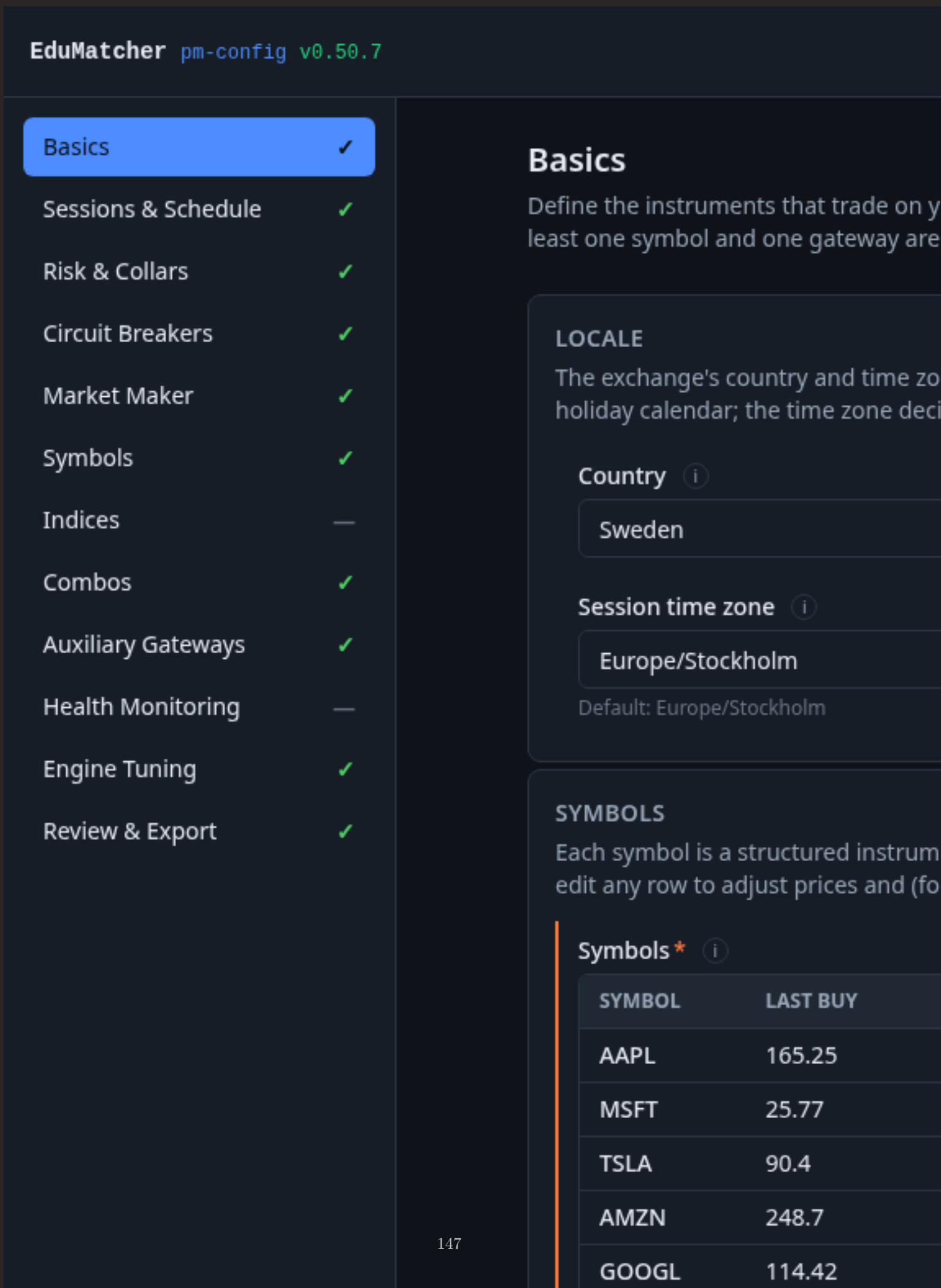
Because persona gates which tabs appear, the visible workspace grows as you move up:

Persona	Tabs shown
Beginner	Basics, Sessions & Schedule, Market Maker[1], Review & Export
Intermediate	…the above **plus** Risk & Collars, Circuit Breakers, Symbols, Indices, Auxiliary Gateways, Health Monitoring
Expert	…the above **plus** Combos, Engine Tuning

[1] The **Market Maker** tab appears at any persona. It is greyed out and disabled until at least one gateway has the `MARKET_MAKER` role, then becomes editable.

Beginner, Intermediate and Expert, with the current persona ticked. Switching hides or shows tabs and fields; it never discards a value.

The theme toggle (/) switches between the light and dark palettes; the choice is remembered in this browser.


EduMatcher pm-config v0.50.7
Basics ✓
Sessions & Schedule ✓
Risk & Collars ✓
Circuit Breakers ✓
Market Maker ✓
Symbols ✓
Indices —
Combos ✓
Auxiliary Gateways ✓
Health Monitoring —
Engine Tuning ✓
Review & Export ✓
Basics
Define the instruments that trade on y
least one symbol and one gateway are
LOCALE
The exchange's country and time zo
holiday calendar; the time zone deci
Country
Sweden
Session time zone
Europe/Stockholm
Default: Europe/Stockholm
SYMBOLS
Each symbol is a structured instrum
edit any row to adjust prices and (fo
Symbols *
SYMBOL LAST BUY
AAPL 165.25
MSFT 25.77
TSLA 90.4
AMZN 248.7
GOOGL 114.42

8.6 The tabs at a glance

Tab	Persona	Configures	Reference chapter
Basics	Beginner	The locale (country, session time zone), the symbol universe, and the participants list	Participants, ALF gateway
Sessions & Schedule	Beginner	Whether the scheduler drives the trading day, and the phase times	Auctions & Session Scheduling
Risk & Collars	Intermediate	The global collar band, named risk levels, and collar enforcement	Price collars
Circuit Breakers	Intermediate	The halt ladder, reference window, reopening auction (ACE), and CB enforcement	Circuit breakers
Market Maker	Beginner[1]	Obligation defaults and startup quote seeding	Market Making
Symbols	Intermediate	Per-symbol overrides for every setting, plus MM quotes	Symbol universe
Indices	Intermediate	Cap-weighted index definitions and constituents	Market Index
Combos	Expert	Multi-leg startup seed orders	Combo Orders
Auxiliary Gateways	Intermediate	Post-Trade, Market-Data, BALF, and API gateway processes	RALF, CALF, BALF, API
Health Monitoring	Intermediate	Health beacons and `pm-monitor`: liveness timing, history, alert thresholds (network at Expert)	Health Monitoring
Engine Tuning	Expert	Low-level runtime retention and throttling knobs, and the auction indicative-price interval	`engine_tuning`
Review & Export	Beginner	Diagnostics summary, YAML preview, download, and verify	Config Verifier

[1] Editable only once a `MARKET_MAKER` gateway exists; before that the tab is shown greyed out.

The rest of this chapter walks through the concepts behind these tabs. If you just want to produce a working file quickly, follow the market-maker workflow and then jump to Reviewing and exporting.

8.7 Basics: symbols and gateways

The **Basics** tab is the mandatory foundation: at least one **symbol** and one **participant** are required before anything can be exported.

- **Locale** — the exchange's **country** (pm-scheduler's holiday calendar) and, from Intermediate, its **session time zone** (the IANA zone the schedule's times are in and that every process uses to split events into trading days). Both are

written to the `locale` block, each only when it differs from its default (`Sweden`, `Europe/Stockholm`). An unknown time zone is an error: the engine refuses it.

- **Symbols** are the instruments that trade on the exchange. Add them with the IPO dialog; names are uppercased automatically and must be unique.
- **Participants** are the sessions permitted to connect (the `participants` list), each with a **role** — `TRADER`, `MARKET_MAKER`, or `ADMIN`. Roles determine which commands a session may send; see Role Privileges and the ALF gateway chapter. A `MARKET_MAKER` gateway unlocks the Market Maker tab and quote seeding; an `ADMIN` gateway can issue exchange-wide halt/resume commands (see ADMIN-role operator controls). Each row also sets the session's **disconnect behaviour** and **SMP** action, and a description.
- **Participant defaults** — an SMP action and a disconnect behaviour applied to every participant that does not set its own (`participant_defaults`).

At **Expert** persona, each `MARKET_MAKER` gateway row shows a **Advanced** button — see per-gateway market-maker overrides.

*Five TRADER desks, an ADMIN and two MARKET_MAKER gateways, above the participant defaults. The table is wider than the panel: scroll it sideways for the SMP and Description columns and, on each MARKET_MAKER row, the **Advanced** button (Expert).*

8.8 Core concepts and workflows

8.8.1 Symbols and the IPO dialog

Clicking **"+ Add symbol"** on the Basics or Symbols tab opens the **"List a new symbol (IPO)"** dialog rather than a plain name field.

The mental model is that adding a symbol is like an initial public offering: on the very first tick the engine has no trade history, so you must supply the starting prices explicitly — otherwise the collar and circuit-breaker reference prices have nothing to anchor to. The dialog captures a single **reference price** and derives everything that must agree with it (last-trade prices, and the market maker's opening quote) in one place, so the book, the last price, and the risk anchors all start consistent. See also Collar reference price selection.

The dialog collects:

Field	Purpose
Symbol name	Uppercased automatically; must be unique.
IPO reference price	The opening mid-price. Derives `last_buy_price` / `last_sell_price` and becomes the collar's static reference on day one.
Tick decimals (Intermediate+)	Price precision (0–8). Defaults to the global value. See tick size.
Outstanding shares	Always required to save, with a sensible default pre-filled; matters most if the symbol will be an index constituent.
Primary market maker	Which `MARKET_MAKER` gateway receives the auto-derived opening quote. Shown, and required, whenever at least one `MARKET_MAKER` gateway is configured — at any persona.
Opening spread (ticks)	Spread, in ticks, used to auto-derive the primary market maker's opening bid/ask around the reference price. Shown alongside **Primary market maker**.
Quote size	Quantity used for the auto-derived primary market maker quote. Shown alongside **Primary market maker**.
Additional market makers (Expert)	Extra bid/ask quote rows for MM gateways beyond the primary one.

If a `MARKET_MAKER` gateway is configured, the primary market-maker quote is always auto-derived from **Opening spread** and **Quote size** applied to the reference price — it is never left blank. Mid-range seeding is a separate mechanism that applies

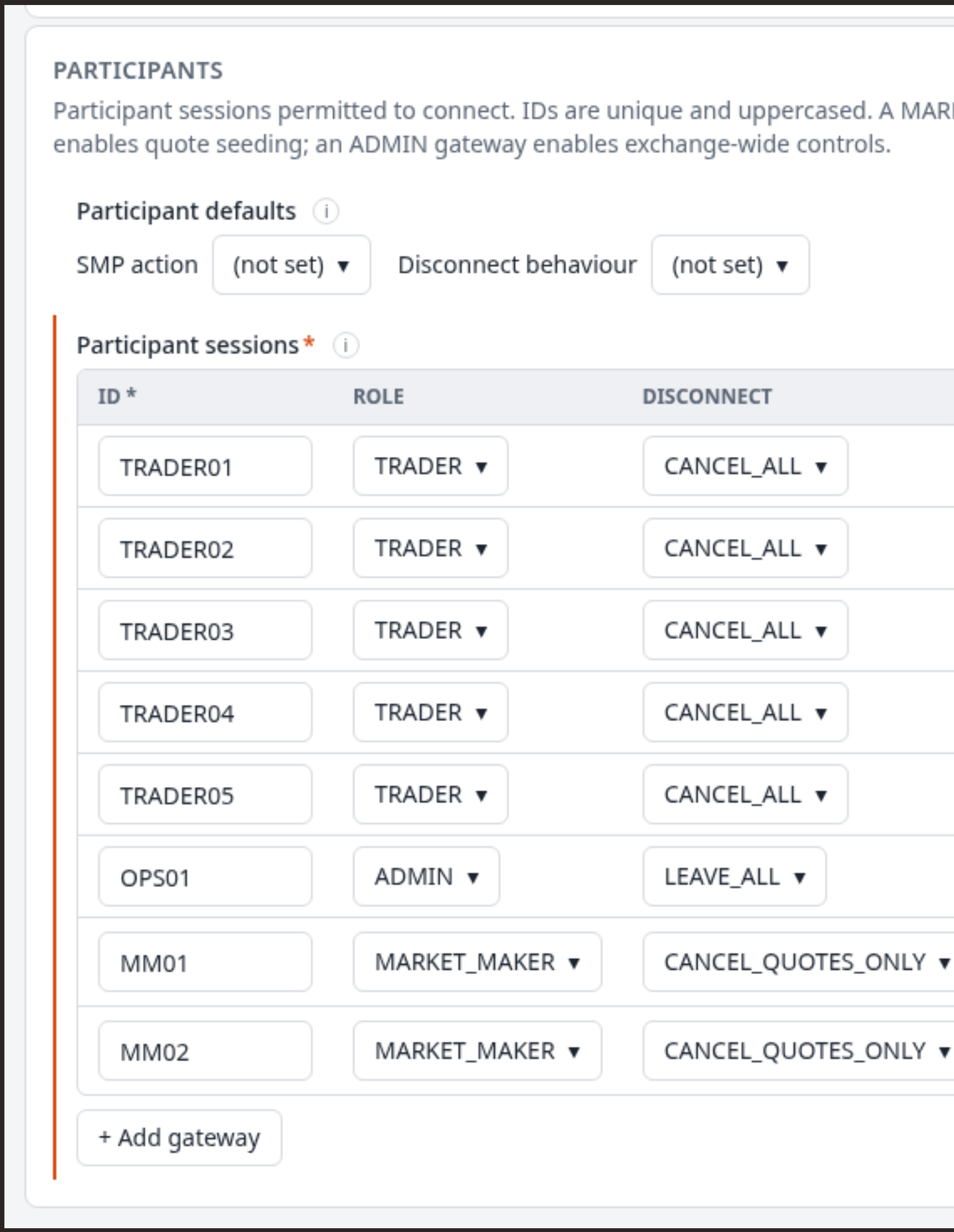


Figure 8.5: The Participants section of the Basics tab.

when a symbol has no explicit quote at all (for example, one added outside the IPO dialog).

AMD listed at 42.50. The primary market maker's opening quote is derived from the opening spread and quote size and shown below them (bid 42.48 / ask 42.52, 500 × 500); an additional market-maker row gives MM02 its own explicit quote.

8.8.2 Importing and exporting symbols

Next to **"+ Add symbol"** on the Basics and Symbols tabs, **Import symbols...** reads a CSV or JSON file (chosen by its `.csv` / `.json` extension) and lists every symbol in it at once. **Export CSV** and **Export JSON** write the current symbols in the same format, so an exported file can be edited and imported into another draft.

Each row holds exactly the IPO dialog's fields, and an imported row creates the same symbol the dialog would:

Column	Required	Blank means
`symbol`	yes	—
`reference_price`	yes	—
`outstanding_shares`	no	the dialog's default, 1,000,000,000
`tick_decimals`	no	the global tick decimals
`mm_gateway`	no	no explicit market-maker quote (mid-range seeding applies)
`spread_ticks`	no	2 (needs `mm_gateway`)
`quote_size`	no	1000 (needs `mm_gateway`)

```
symbol,reference_price,outstanding_shares,tick_decimals,mm_gateway,spread_ticks,quote_size
AAPL,182.50,15000000000,2,MM01,2,500
MSFT,410.00,,,,,
```

The JSON form is an array of objects with the same keys; a missing key or `null` counts as blank:

```
[
  { "symbol": "AAPL", "reference_price": 182.5, "mm_gateway": "MM01" },
  { "symbol": "MSFT", "reference_price": 410 }
]
```

An import is all-or-nothing. The whole file is checked first, and any problem rejects it: an unknown column or key, a symbol that already exists or appears twice, a value out of range, or an `mm_gateway` that is not a `MARKET_MAKER` gateway. The error dialog lists every problem with its line (CSV) or entry (JSON) number, and the draft is unchanged.

Export writes only these fields. Per-symbol overrides such as risk level, collars, order limits and circuit-breaker settings are not included. After editing, a symbol may no longer fit the IPO fields: its last buy and last sell prices may differ, or it may have a hand-edited or second market-maker quote. Such a symbol is exported as closely as the fields allow (last buy becomes `reference_price`), and a dialog names it and gives the reason, because re-importing the file would not reproduce it.

CSV files follow the usual quoting rules: a cell that contains a comma or a double quote is wrapped in double quotes, and a double quote inside it is doubled (`"Says ""hi"", twice"`). Export writes cells this way, and import reads them.

8.8.3 Importing and exporting participants

The Participants table on the Basics tab has the same three buttons next to **"+ Add gateway"**: **Import participants...**, **Export CSV** and **Export JSON**. Each row holds exactly the table's fields, and an imported row creates the gateway **"+ Add gateway"** would:

Column	Required	Blank means
`id`	yes	—
`role`	no	`TRADER`

Column	Required	Blank means
disconnect_behaviour	no	the role's default (CANCEL_ALL for TRADER, CANCEL_QUOTES_ONLY for MARKET_MAKER, LEAVE_ALL for ADMIN)
smp_action	no	inherit the participant default
description	no	no description

```
id,role,disconnect_behaviour,smp_action,description
DESK01,TRADER,,CANCEL_AGGRESSOR,"Equities desk, floor 2"
MM01,MARKET_MAKER,,,
OPS,ADMIN,,,Operations
```

IDs and the role, disconnect_behaviour and smp_action values are uppercased on import. As for symbols, the import is all-or-nothing: an ID that already exists or appears twice, or a value that is not one of the allowed ones, rejects the whole file.

Export writes only these fields. The Expert **Advanced** market-maker settings are not included. A participant whose disconnect behaviour is **(default)** is written with the value it inherits now, so the engine behaves the same after a re-import. The export dialog names it, because the re-imported participant no longer follows later changes to the participant default.

8.8.4 The Symbols workspace

The **Symbols** tab (Intermediate+) has three parts, top to bottom.

Global default — the global tick decimals and the symbol universe, a list of the symbols with **+ Add symbol** (the IPO dialog).

Per-symbol overrides — one row per symbol, with the per-symbol settings most often tuned across many symbols at once: the collar's static and dynamic bands, and the reopening auction's ACE switch, corridor and random end. A greyed cell inherits the exchange value; typing a value overrides it for that symbol, and clearing it returns the symbol to inheriting. A dot after a symbol's name marks one with overrides. For a large universe, **Filter symbols...**, **Only overridden** and the **Rows** page size (25, 50, 100) narrow the list. Tick rows to get the bulk toolbar: **Set** one of the five fields **to** a value and **Apply** it to every selected symbol, or **Clear overrides** to return them all to the exchange defaults.

META, NVDA and NFLX are selected, so the bulk toolbar is shown. TSLA carries a dynamic-band override of 4% (in full colour, marked by the dot); every other cell is greyed because it inherits.

The editor — the list of symbols on the left, the selected one on the right, through sub-tabs:

Sub-tab	Persona	Contents
General	Intermediate	Tick decimals, last buy and last sell prices, outstanding shares
Collar & Limits	Intermediate	The assigned risk level, static/dynamic band overrides, and the order limits (max order quantity, max order value)
Circuit Breaker	Expert	Per-symbol ladder overrides — see Circuit breakers
Market Maker	Intermediate	Per-symbol obligation overrides
MM Quotes	Intermediate	Explicit opening quotes per MM gateway

Two conveniences make it easy to check a symbol without losing your place:

- An **eye icon** on each row in the left list opens the read-only Symbol Overview *without changing* the current selection — so you can peek at any symbol while editing another.
- An **Overview** button in the detail-pane header opens the same overview for the currently selected symbol.

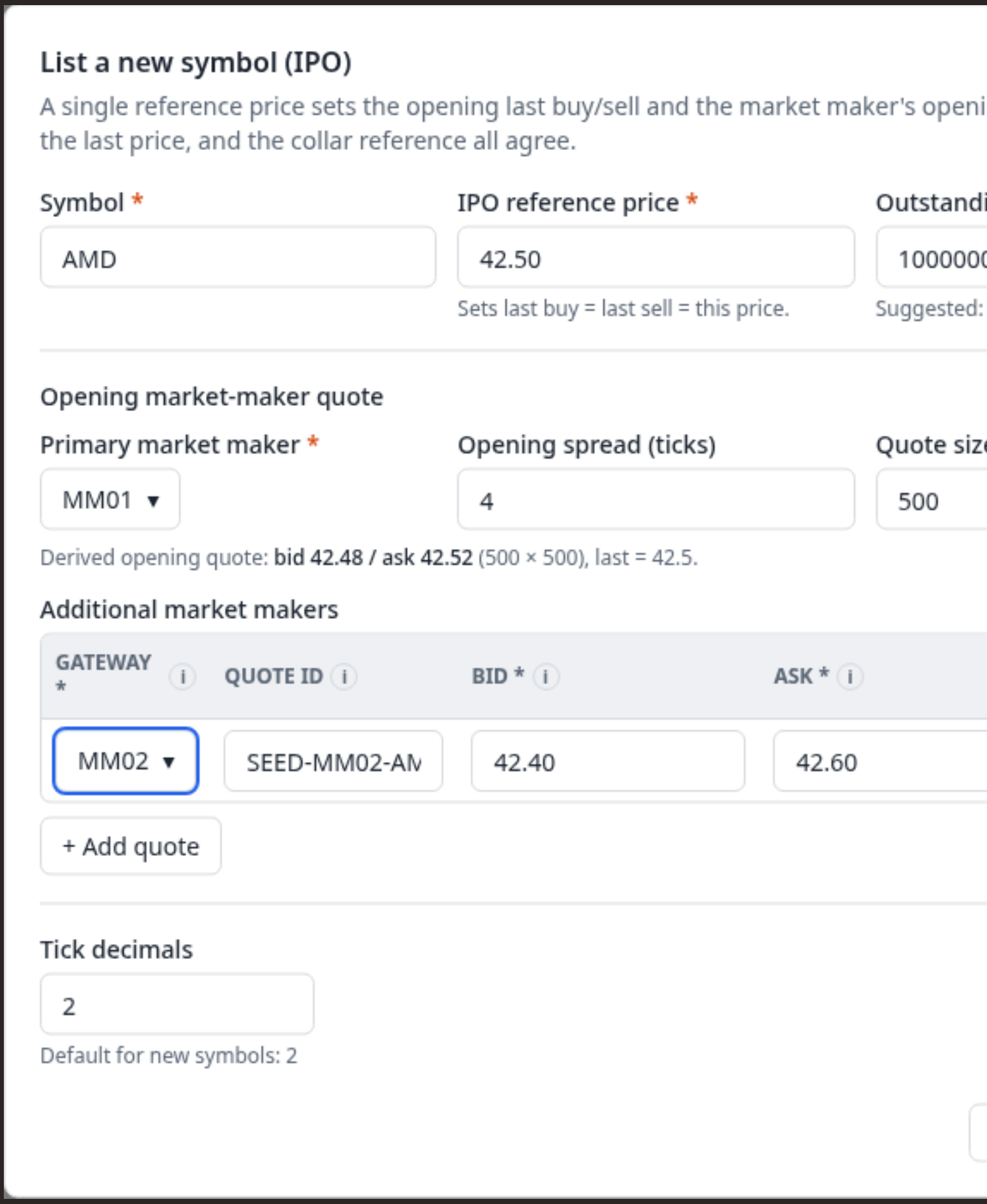


Figure 8.6: The List a new symbol (IPO) dialog at the Expert persona.

PER-SYMBOL OVERRIDES

Every symbol appears here. Greyed cells inherit the exchange default; type a value
symbol. Select rows to change many at once.

Filter symbols... Only overridden 10 symbols

3 selected Set ACE corridor ±% to Apply

Clear overrides

	SYMBOL	COLLAR STATIC %	COLLAR DYN %	ACE
☐	AAPL	18	2	
☐	MSFT	20	2	
☐	TSLA •	12	4	
☐	AMZN	20	2	
☐	GOOGL	20	2	
☑	META	20	2	
☑	NVDA	20	2	
☑	NFLX	20	2	
☐	INTC	20	2	
☐	ORCL	20	2	

Greyed values are inherited from the exchange defaults; typing a value creates a per-symbol ove

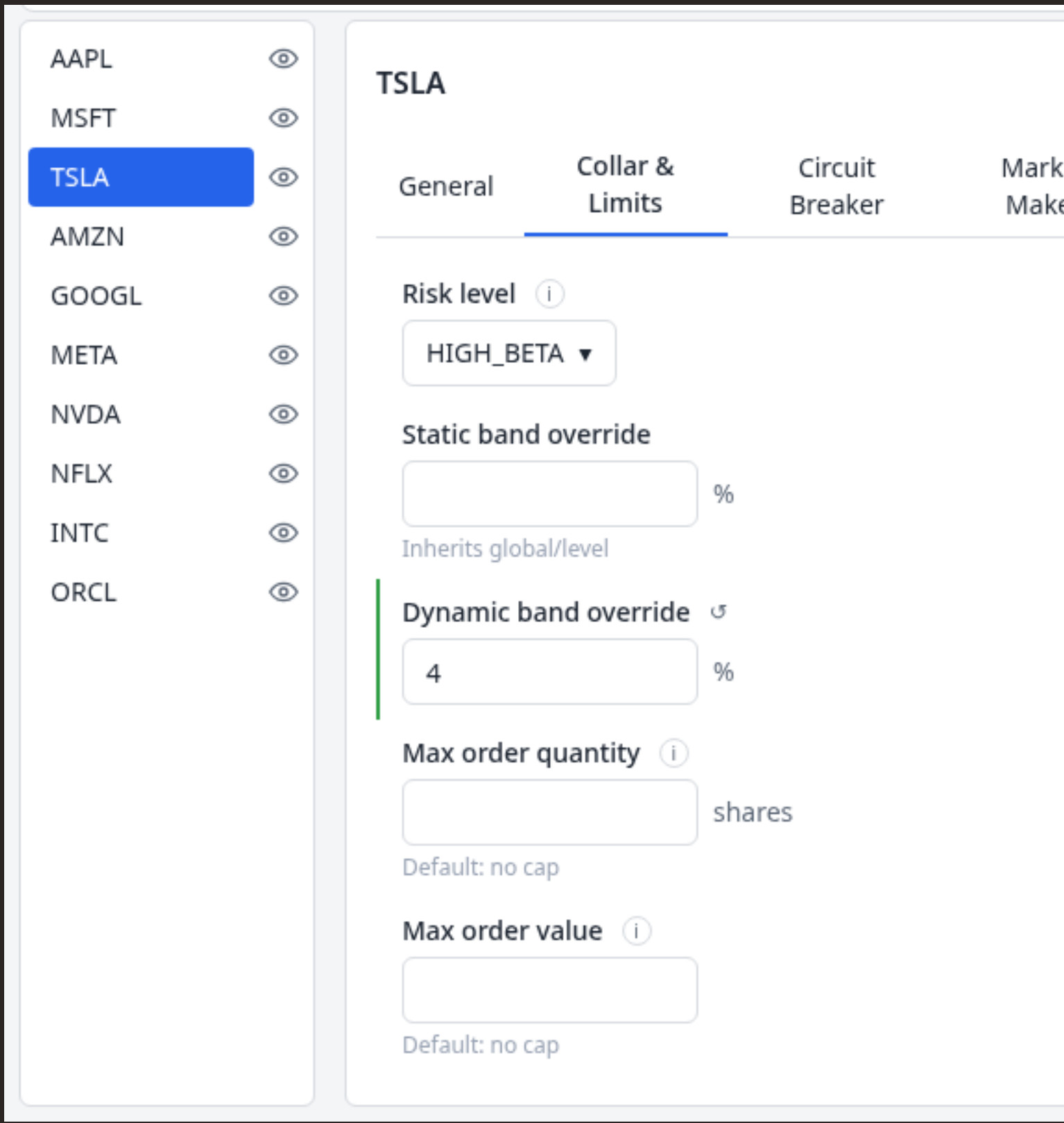


Figure 8.8: The symbol editor on TSLA's Collar & Limits sub-tab.

TSLA is assigned the HIGH_BETA risk level and overrides its dynamic band (4%, marked by the green edge and the reset); its static band and order limits are blank, so they inherit. The icons open any symbol's Overview without changing the selection.

8.8.5 Symbol overview

The **Overview** dialog is read-only and shows the **effective** values — what the engine will actually use after all inheritance and merge rules are applied — in one compact view, with each value annotated by its source. It consolidates what is otherwise spread across five sub-tabs and several global defaults.

Section	What it shows
General	Tick decimals (per-symbol override vs global), last buy/sell prices, outstanding shares, and the active risk level (assigned or inherited default).
Collar (effective)	The static and dynamic band percentages that will apply, each tagged with its source (`symbol override`, `level CORE`, `engine default`). Flags if collar enforcement is off globally.
Order limits (effective)	The max order quantity and value that apply, or that the symbol has none.
Circuit breaker (effective)	The resolved reference window and full ladder. **Per-symbol overrides appear in accent colour**; inherited values are shown normally. Below the ladder, the reopening auction (ACE on/off, initial corridor, random end) and how the corridor widens. Flags if CB enforcement is off globally.
Market maker (effective)	The resolved obligation (enforce/max-spread/min-qty) with source, any per-gateway overrides, and the effective opening quotes with an origin badge (`explicit`, `seeded`, or `fill in`).
Memberships	Which indices and combos reference this symbol — cross-references not otherwise visible from the Symbols tab.

Use it as a final sanity check before exporting, or to diagnose why a symbol behaves unexpectedly in the engine.

Every value carries its source: the static band comes from the HIGH_BETA level, the dynamic band and the 3-minute reference window are symbol overrides, and L1's 5% and 10 min are per-symbol ladder overrides, in accent colour. Both market makers' opening quotes are explicit.

8.8.6 Risk and collars

A **collar** rejects orders priced too far from a reference price, protecting the book from fat-finger errors. The static band anchors to a session reference; the dynamic band tracks near-live prices. The full mechanism — band definitions, which orders are checked, and how it differs from circuit breakers — is described in Price collars; order-level behaviour is covered in Order Types.

On the **Risk & Collars** tab (Intermediate+) you can:

- Set a **global band**, which creates the `DEFAULT` risk level applied to every symbol that does not specify its own.
- Define **named risk levels** (e.g. `CORE`, `HIGH_BETA`) that symbols reference by name — see Global level profiles and per-symbol risk-level assignment.
- Toggle **collar enforcement** globally (`enforce_collars`). Because this tab is Intermediate, a Beginner draft keeps enforcement at its safe default (on).

Per-symbol band overrides live on the Symbols → **Collar & Limits** sub-tab (or the per-symbol overrides table) and take precedence over the level and global values.

Collar enforcement on, the global band that defines the DEFAULT level (20% / 2%), and two named levels, CORE and HIGH_BETA, that symbols can be assigned.

Overview — TSLA

Risk level: H

GENERAL

Tick decimals	
Last buy price	
Last sell price	
Outstanding shares	
Risk level	HIG

COLLAR (EFFECTIVE)

Static band	12%
Dynamic band	4%

ORDER LIMITS (EFFECTIVE)

No order limits — orders for this symbol are not size- or notional-capped.

CIRCUIT BREAKER (EFFECTIVE)

Reference window 3 min ·

LEVEL	SHIFT %	HALT
L1	5%	10 min
L2	13%	15 min
L3	20%	rest of day

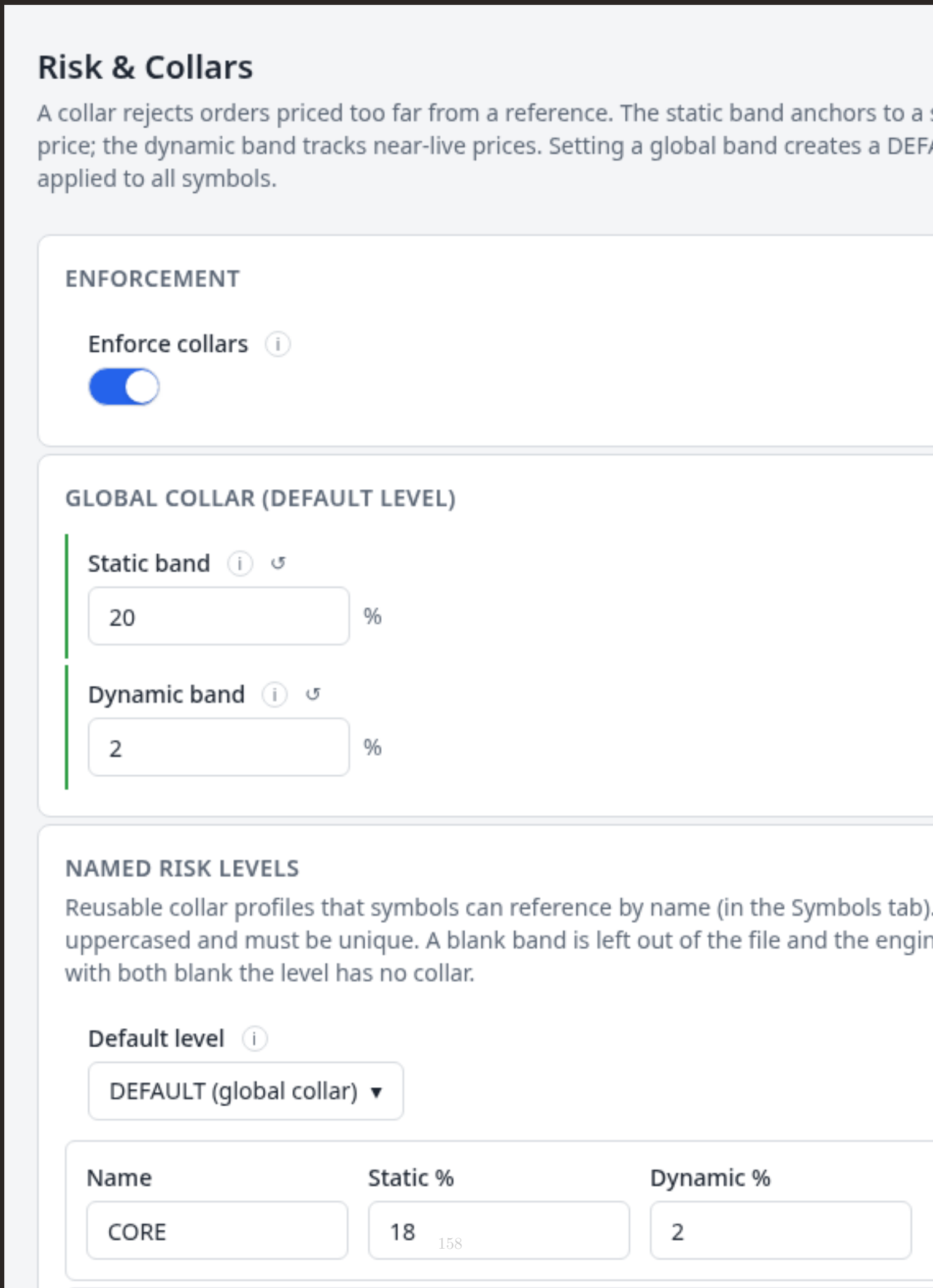
Values in accent are per-symbol overrides; the rest inherit the global ladder.

Reopening (ACE)

Initial corridor	±1
Random end	

Corridor widens ±10.0% → ±20.0% → ±40.0% → ±60.0% → … until the reopening price fits, then
the close? It prints at the corridor boundary.

MARKET MAKER (EFFECTIVE)


Risk & Collars
A collar rejects orders priced too far from a reference. The static band anchors to a
price; the dynamic band tracks near-live prices. Setting a global band creates a DEF
applied to all symbols.
ENFORCEMENT
Enforce collars
GLOBAL COLLAR (DEFAULT LEVEL)
Static band
20
%
Dynamic band
2
%
NAMED RISK LEVELS
Reusable collar profiles that symbols can reference by name (in the Symbols tab).
uppercased and must be unique. A blank band is left out of the file and the engi
with both blank the level has no collar.
Default level
DEFAULT (global collar)
Name
Static %
Dynamic %
CORE
18
2

8.8.7 Circuit breakers

A **circuit breaker** halts trading in a symbol when it moves too far from its rolling reference. A halt is the call phase of a **reopening auction**: each ladder level has a **shift %** (how far triggers it) and a **call-phase minimum** (how long the auction collects orders at least), or halts for the **rest of the day**. Automated Corridor Expansion (ACE) then decides whether the auction may end: the symbol reopens only if the indicative price falls inside a corridor around the reference price; otherwise the corridor widens and another call phase begins. The complete model — the rolling reference window, day-one behaviour, and ACE — is in Circuit breakers.

The **Circuit Breakers** tab (Intermediate+) has four parts:

- **Enforcement** — the global `enforce_circuit_breakers` switch, and **Write exchange defaults**: off leaves the `circuit_breaker_` block out of the file, so a symbol without a circuit-breaker override of its own has no circuit breaker at all.
- **Reference window** — the rolling reference window, in minutes.
- **Ladder** — Level, Shift %, Call phase min (min), and Rest of day. At **Expert** persona you can add or remove levels.
- **Reopening auction (ACE)** — whether ACE is on, the initial corridor (±%), the random end (seconds), an optional random seed, and the **expansion ladder**: how much each further call phase widens the corridor and how long it lasts. The last rung repeats indefinitely, so the corridor eventually contains any price and every halt resolves on its own.

Per-symbol overrides live on the Symbols → **Circuit Breaker** sub-tab (Expert), which mirrors the ladder but treats blank cells as "inherit the global ladder value". You can override the shift, the call-phase minimum (including "rest of day") and the reference window for a single symbol, or add a level only that symbol has. The ACE switch, corridor and random end are overridden per symbol in the per-symbol overrides table.

The `s10-complex` ladder: L1 at 7% for a 5-minute call phase, L2 at 13% for 15 minutes, and L3 at 20% for the rest of the day. ACE reopens inside ±10%, widening to ±20% and then ±40% (repeating) if the indicative price is still outside.

8.8.8 Sessions and schedule

A trading day moves through phases — pre-open, opening auction, continuous trading, closing auction — driven by the `pm-scheduler` process. The **Sessions & Schedule** tab (Beginner) toggles `sessions_enabled` and, from Intermediate, edits the phase times.

The GUI enforces that phase times are strictly increasing as you type — the same rule the engine applies — so an out-of-order schedule is caught immediately rather than at export. See Session phases for what each phase means.

Sessions on, so `pm-scheduler` must run; the weekday phase times from pre-open (08:45) to the end of the closing auction (16:10). Weekends and holidays are not scheduled.

Sessions imply a running scheduler

Turning `sessions_enabled` on means the engine starts CLOSED and waits for `pm-scheduler` to drive transitions. If you enable sessions, remember to run the scheduler alongside the engine, or the market stays closed.

8.8.9 Market makers

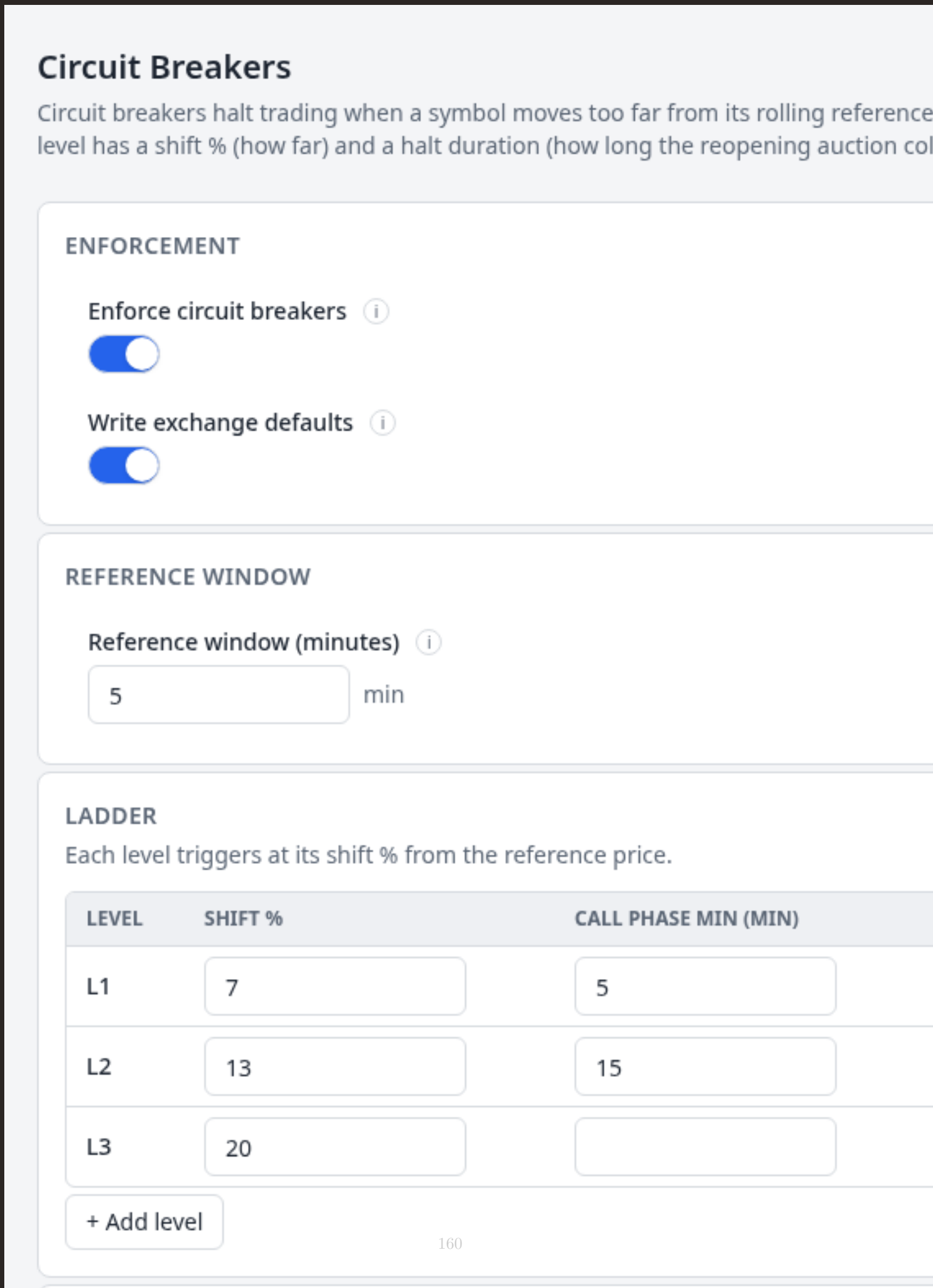
Market makers supply the two-sided quotes that give a symbol a tradeable book from the first moment. The GUI models this in three independent layers plus a quote-seeding mechanism. Understanding the layers avoids duplicate work and confusion about which value "wins". The engine-side reference is Market Making; obligation enforcement is detailed under MM obligations enforcement.

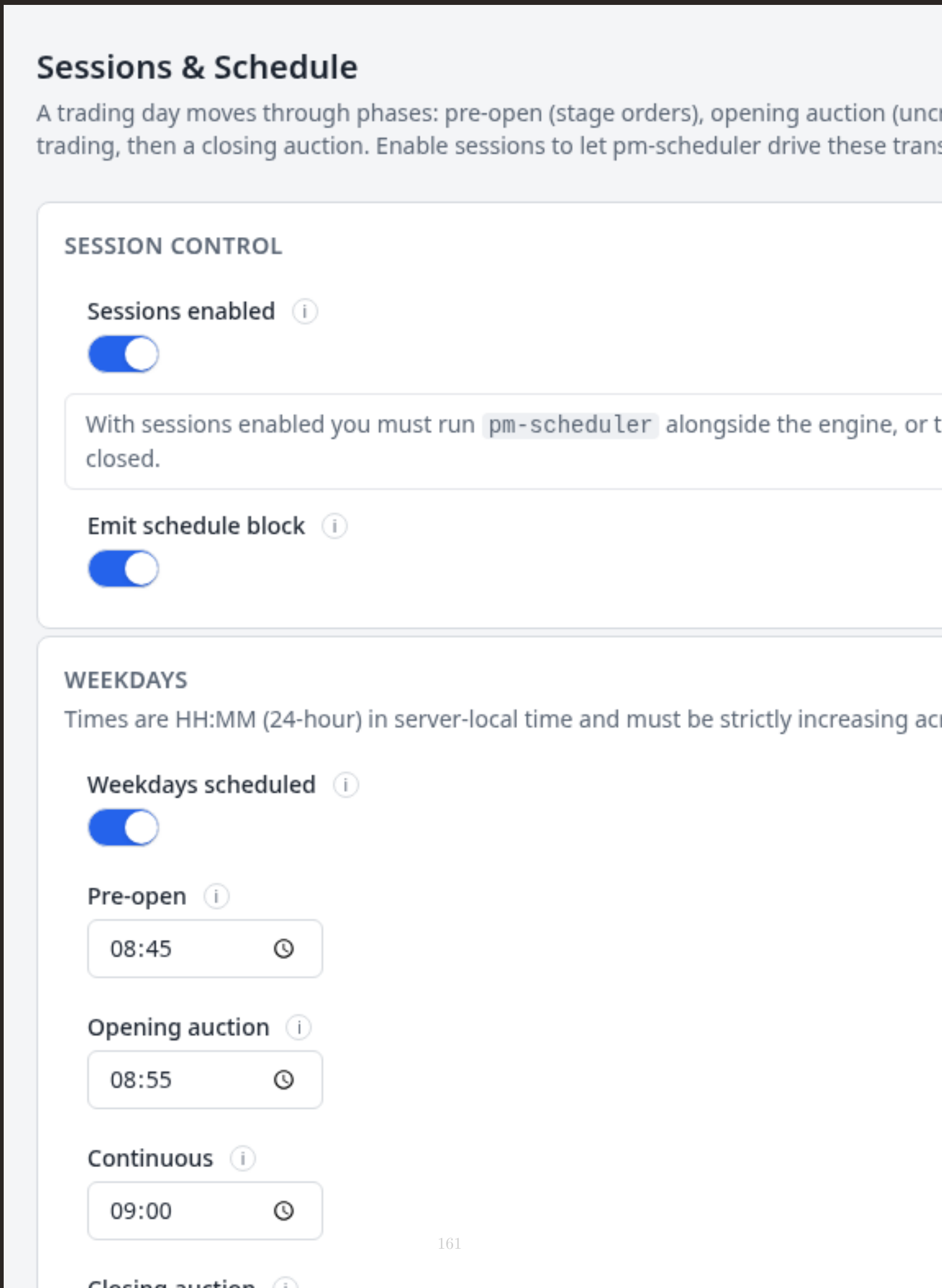
Layer 1 — Global obligation defaults (Market Maker tab)

The **Market Maker tab** holds the exchange-wide defaults applied to every symbol and every `MARKET_MAKER` gateway unless overridden:

- **Enforce MM obligations** — whether the engine checks quote-width and size compliance at all.
- **Max spread (ticks)** — the widest allowed bid–ask spread; wider quotes are rejected (spread constraint).
- **Min quantity** — the minimum displayed size on each side (minimum size constraint).

This layer maps to `mm_obligation_defaults` in the YAML (reference).


Circuit Breakers
Circuit breakers halt trading when a symbol moves too far from its rolling reference
level has a shift % (how far) and a halt duration (how long the reopening auction col
ENFORCEMENT
Enforce circuit breakers
Write exchange defaults
REFERENCE WINDOW
Reference window (minutes)
5
min
LADDER
Each level triggers at its shift % from the reference price.
LEVEL
SHIFT %
CALL PHASE MIN (MIN)
L1
7
5
L2
13
15
L3
20
+ Add level


Sessions & Schedule
A trading day moves through phases: pre-open (stage orders), opening auction (uncr
trading, then a closing auction. Enable sessions to let pm-scheduler drive these trans
SESSION CONTROL
Sessions enabled
With sessions enabled you must run pm-scheduler alongside the engine, or t
closed.
Emit schedule block
WEEKDAYS
Times are HH:MM (24-hour) in server-local time and must be strictly increasing acr
Weekdays scheduled
Pre-open
08:45
Opening auction
08:55
Continuous
09:00

Layer 2 — Per-symbol obligation overrides (Symbols → Market Maker)

The Symbols → **Market Maker** sub-tab relaxes or tightens the obligation for one instrument. `(inherit)` on a field writes nothing (the global default applies); setting a value emits an `mm_obligation_defaults.symbols.<SYM>` entry that overrides only that field. Use it when, say, a less liquid symbol needs a wider allowed spread. See per-symbol overrides.

Layer 3 — Per-gateway, per-symbol overrides

At **Expert** persona, the **Advanced** dialog on each `MARKET_MAKER` gateway row (Basics tab) provides the finest-grained control:

1. **Flat gateway overrides** — `enforce_mm_obligation`, `mm_max_spread_ticks`, `mm_min_qty` for *every symbol this gateway quotes.*
2. **Per-symbol table** — overrides for specific symbols on this gateway, written as `participants[*].mm_obligations.<SYM>` (the nested keys are `max_spread_ticks` / `min_qty`, without the `mm_` prefix — a format quirk the GUI handles for you).

The dialog also sets the gateway's **quote refresh policy** — when seeded quotes are inactivated after a fill. The default `INACTIVATE_ON_ANY_FILL` suits most scenarios.

Precedence (highest to lowest):

```
Per-gateway per-symbol  (mm_obligations.<SYM>)
  → Per-gateway flat    (gateway.mm_max_spread_ticks / mm_min_qty)
    → Per-symbol global (mm_obligation_defaults.symbols.<SYM>)
      → Global defaults (mm_obligation_defaults)
```

MM01 keeps the default quote refresh policy, allows itself a 25-tick spread instead of the global 20, and has a stricter AAPL override: at most 10 ticks wide, at least 500 on each side.

Quote seeding: mid-range vs explicit

Before the engine starts there must be resting quotes in the book (see startup seeding). The GUI offers two ways to supply them, plus an automatic fallback.

Option A — Explicit quotes (highest precedence). On the Symbols → **MM Quotes** sub-tab you enter a specific `bid_price` / `ask_price` / `bid_qty` / `ask_qty` for each symbol × MM gateway pair. Every column header has an **(i)** help popover; the most-asked-about one is **Seed once**:

> When on, the engine injects this quote only if it has no prior book state for the symbol, so a restart with existing history will not re-inject it. Turn it off to (re)seed on every startup regardless of history.

See controlling when seeds apply and choosing TIF for seeds.

Require MM seed quotes (the default, visible at every persona) controls whether stubs are written at all: on, every symbol must end up with at least one `market_maker_quotes` entry per `MARKET_MAKER` gateway, falling back to a null-price stub (below) when neither Option A nor Option B supplies one; off, a symbol with no explicit quote simply gets no stub, for a genuinely empty book at startup. This maps to `require_mm_seed_quotes` in the YAML (the CLI's `--no-mm-seed-quotes` flag turns it off).

Option B — Global mid-range seeding. On the Market Maker tab, set a **Seed MM mid-range** (e.g. `100` to `300`). The builder computes one deterministic midpoint and applies it to *every* symbol:

```
midpoint = (min + max) / 2     # arithmetic mean, snapped to the tick grid
bid      = midpoint − 1 tick
ask      = midpoint + 1 tick
```

The same midpoint applies to every symbol

A mid-range of `200–220` gives every symbol a midpoint of `210`, so AAPL and TSLA both open with `bid: 209.99 / ask: 210.01` (at `tick_decimals = 2`). The range is *averaged into a single value*, not distributed across symbols. If you need different opening prices per symbol, use explicit quotes (Option A).

Two related switches (Intermediate+) accompany mid-range seeding:

- **Seed last prices from MM** — derives `last_buy_price` / `last_sell_price` from the same midpoint so the collar's static reference matches the opening quote. Requires a mid-range.
- **Seed placeholder last prices** — emits `last_buy_price: null` / `last_sell_price: null` for instruments whose reference price is managed externally.

Fallback — null stub. If neither an explicit quote nor a mid-range is supplied (and **Require MM seed quotes** is on),

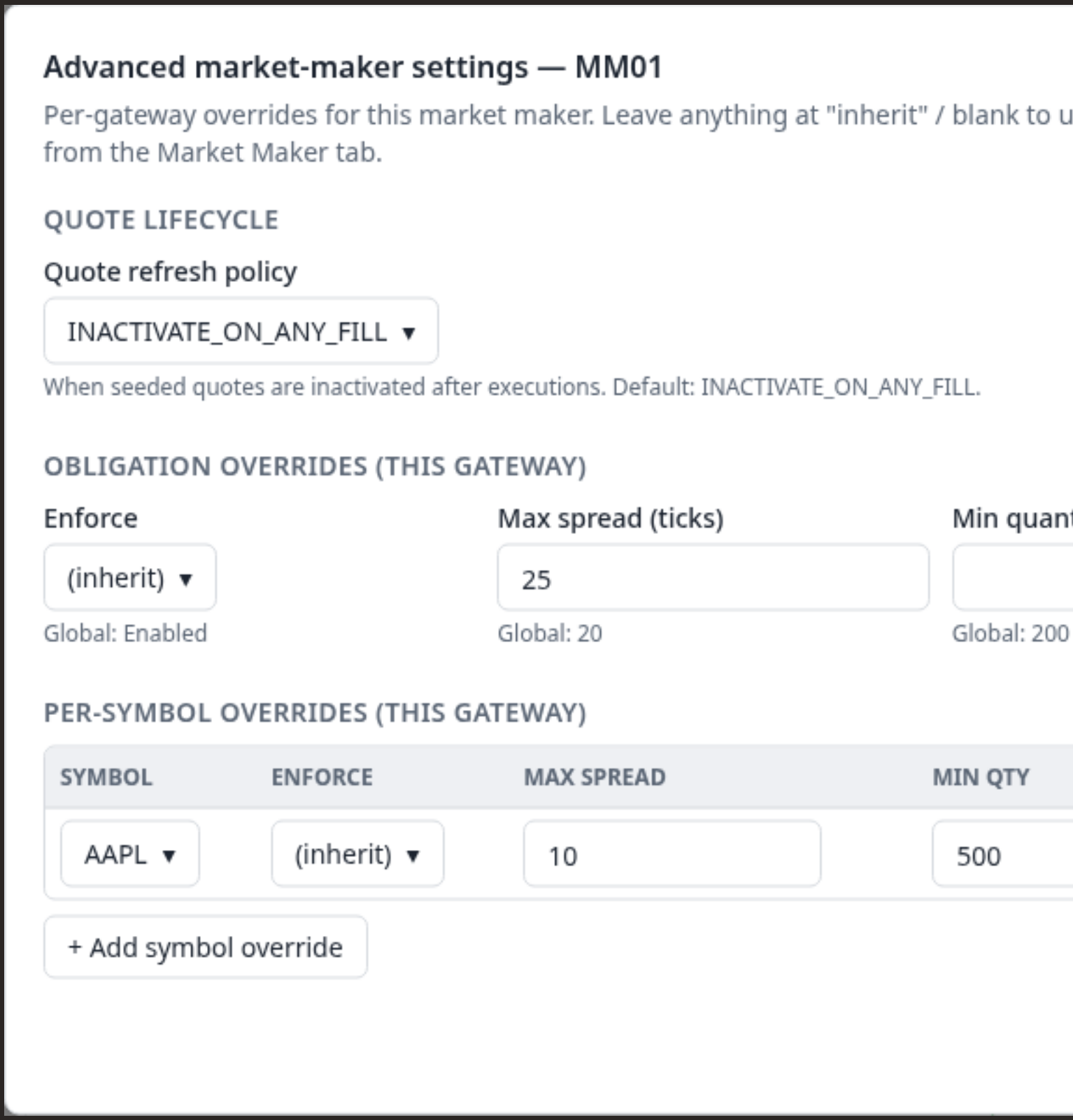


Figure 8.13: The Advanced market-maker settings for MM01.

the builder emits `bid_price: null` / `ask_price: null`. The Quote Stub Review flags this and the Review tab blocks export until it is resolved.

A workflow for a new market maker

A logical order that avoids rework:

1. **Add a `MARKET_MAKER` gateway** (Basics). Give it a clear ID like `MM01`. At Expert, review its quote refresh policy in the dialog.
2. **Set global obligation defaults** (Market Maker tab): max spread, min quantity, and whether to enforce. Leave defaults for a simple classroom exchange.
3. **Choose a seeding strategy.** *Quick:* set a mid-range (e.g. `100–100`) and enable **Seed last prices from MM**. *Realistic:* leave the mid-range unset and plan to enter explicit quotes per symbol.
4. **Add symbols** with the IPO dialog. If you chose explicit quotes, add one MM Quotes row per gateway for each symbol.
5. **Check the Quote Stub Review** (below) — every row must be green before export.
6. **Adjust per symbol** (optional) on the Symbols → Market Maker sub-tab, or per gateway in the dialog.
7. **Verify with the Symbol Overview** before exporting.

The Quote Stub Review

The Market Maker tab's **Quote Stub Review** gives a go/no-go status for every symbol × MM gateway pair:

Status	Meaning	Action
quote defined	An explicit quote with both prices is set	None
seeded from mid-range	The mid-range will compute prices at export	None
— no quote written	This gateway has no stub — the symbol's explicit quotes cover a different gateway only, or **Require MM seed quotes** is off	None (this is a normal, intentional state, not an error)
no prices — fill in before starting the engine	**Require MM seed quotes** is on but neither an explicit quote nor a mid-range supplied one	Add a quote or set a mid-range

Export is blocked while any row is unresolved.

AAPL's explicit quotes were removed and no mid-range is set, so its stubs would have null prices: the tab reports two errors, which are shown on the two fields that would resolve them.

AAPL has no prices for either gateway; MSFT has an explicit quote for MM01 only, so MM02 gets none written; TSLA has explicit quotes for both. Setting a seed mid-range turns AAPL's rows into " seeded from mid-range".

8.8.10 Indices

The **Indices** tab (Intermediate+) defines cap-weighted indices and their constituents. Each constituent must be a configured symbol with `outstanding_shares` set (the GUI flags any that are missing). The calculation model and field reference are in Market Index.

EDU1, a cap-weighted index of six of the ten symbols, picked by clicking them. Base value, publish interval and the history and state files show their defaults.

8.8.11 Combos

The **Combos** tab (Expert) builds multi-leg startup seed orders — pairs, spreads, hedged entries, and market-maker spreads. Prices are entered as decimal display values and converted to ticks per each leg symbol's precision. See Combo Orders for the strategy background and the COMBO order type for lifecycle behaviour.

The `s10-complex` pair: an all-or-nothing combo that buys 100 AAPL and sells 100 MSFT at their opening prices. The legs table scrolls sideways for the stop price and SMP.

Market Maker

Market-maker gateways supply two-sided quotes. Obligations enforce a maximum s
minimum size. Seeding pre-fills opening quotes so the book is not empty at startup.

2 errors on this tab must be resolved before export.

OBLIGATIONS

Enforce MM obligations

Max spread (ticks)

20

New configs: 20 (the engine's own default when the key is omitted is 10)

Min quantity

200

Default: 100

QUOTE SEEDING

Set a mid-range and the builder seeds a bid/ask one tick around a midpoint for ev
gateway. Without it, quote stubs are emitted with null prices that must be filled in
engine.

Require MM seed quotes

✗ Symbol AAPL would get a quote stub for MM01 with null prices, which the engine refuses.
explicit quotes for the symbol, or turn off "Require MM seed quotes".

✗ Symbol AAPL would get a quote stub for MM02 with null prices, which the engine refuses.
explicit quotes for the symbol, or turn off "Require MM seed quotes".

Seed MM mid-range

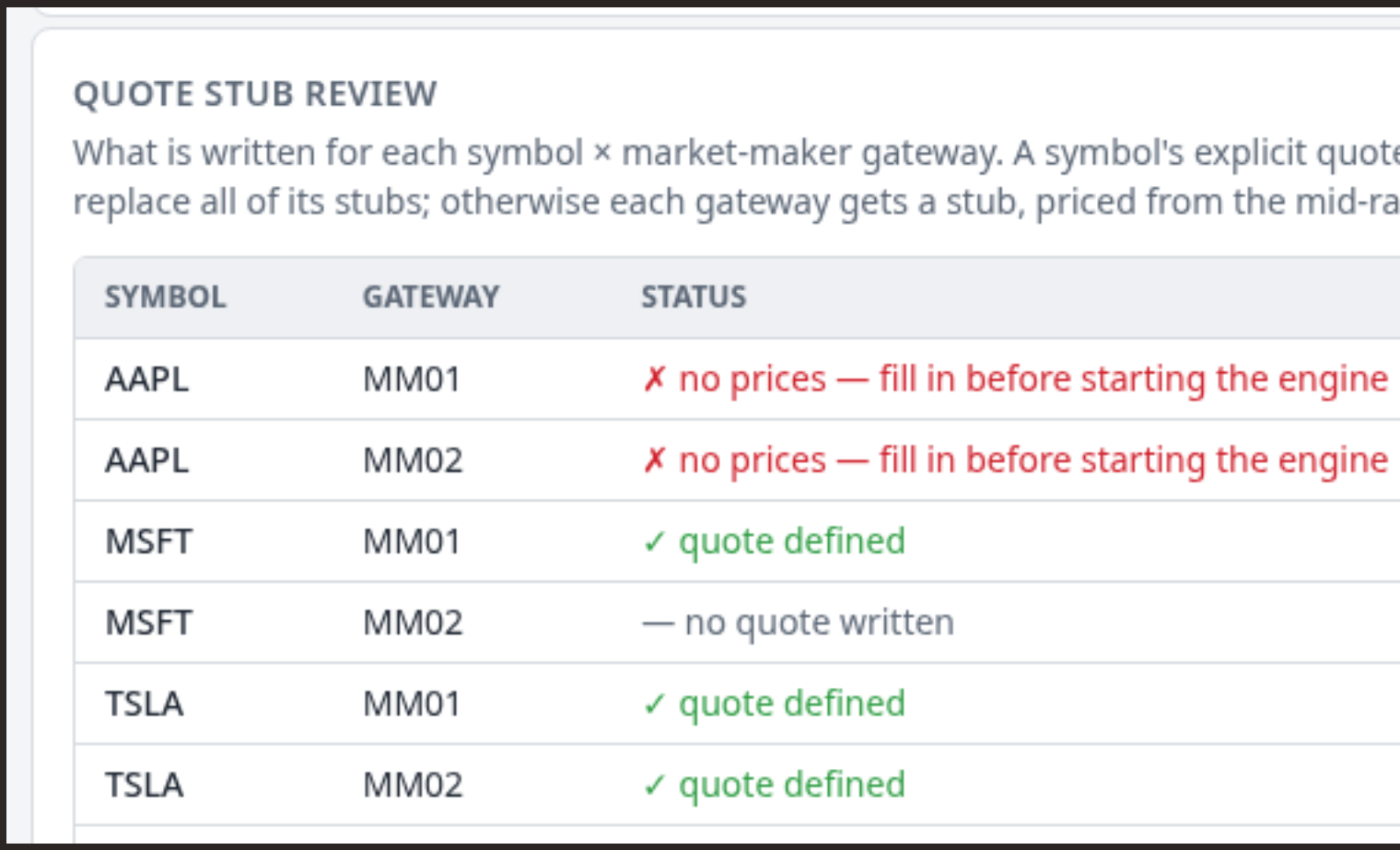


Figure 8.15: The top of the Quote Stub Review.

8.8.12 Auxiliary gateways

The **Auxiliary Gateways** tab (Intermediate+) configures the optional network services around the engine, each on its own sub-tab. Two sub-tabs are visible at Intermediate; the rest — including the primary **ALF** gateway's own advanced tuning — only appear at Expert:

Sub-tab	Persona	Process	Reference
ALF	Expert	ALF — the primary text order-entry gateway's own tuning (heartbeat/handshake/idle timeouts, connection and rate limits)	ALF Gateway
Post-Trade	Intermediate	RALF — fills, drop-copy, audit	Post-Trade (RALF)
Market-Data	Intermediate	CALF — top-of-book, trades, index	Market Data Feed (CALF)
BALF	Expert	Binary ALF gateway	BALF Gateway
Drop-Copy	Expert	DC1 — a lightweight TCP relay of live fill notifications, independent of RALF	Drop-Copy Gateway

Sub-tab	Persona	Process	Reference
Log Server	Expert	`pm-log-srv` — the centralized log collector every other process connects to, plus its optional LALF-PS live-distribution socket pair	Log Server
API	Expert	REST/WebSocket gateway	API Gateway

*The `desk` instance on port 8080 with one credential per participant; **Generate** makes a fresh random key. A second instance, `dashboards`, follows below it with the read-only key.*

Generate keys for participants... opens a list of every participant, all selected by default, with **Select all** / **Select none**. Each selected participant gets one random key bound to it, in pm-config-gen's `key-<id>-<token>` shape with the description `Generated key for <ID>`. **Key length** sets the number of random characters in the token. The default is 6, as pm-config-gen uses, which is easy to type in a class; use 24 or more for anything exposed to a network you do not control. A participant that already has a key, in this instance or another, is shown but cannot be selected, because a participant may have keys in only one instance. A row's **Generate** button and **+ Add credential** make keys in the same shape, labelled `readonly` for a read-only key (`key-readonly-…`). They use the length last used in the dialog, which this browser remembers (6 until then).

The **API credentials in separate files** switch on the Review & Export tab turns credentials files on for every instance at once. A single instance writes to `api_keys.yaml`, as `pm-config-gen --api-keys-for` does. Several instances each get `api_keys_<instance>.yaml`, because instances sharing a file would each accept every key in it. A name an instance already has is kept. Turning the switch off writes all credentials inline again. An instance added while it is on gets a file too.

An instance's **Credentials file** keeps its API keys out of the main file, as `pm-config-gen --api-keys-for` does. The main file then names the file with `credentials_file:`, and **Download** writes the keys to that file beside it. The file holds secrets, so keep it out of version control. Instances that name the same file all read the whole of it, so it is written once. The Review tab flags an error if they list different credentials, or if the name is absolute, contains `..`, is reserved (`engine_config.yaml`, `engine_config.json`, `market_sim.yaml`), or is the name of another output file.

Each sub-tab (other than ALF, which is always present) has its own **"Write `<section>` section"** toggle: off omits that section from the exported file entirely, rather than writing it with defaults. Enabling a gateway reveals its fields; disabling it keeps your values staged but excludes the section from the export. **Ports are collision-checked across every enabled gateway** — the diagnostics drawer warns if two services would bind the same address and port.

The **Log Server** sub-tab is the largest of the seven: beyond the collector's own name/bind/port/database-path/retention/batching settings, a **"LALF-PS — log distribution"** section (its own on/off toggle) exposes the ZeroMQ PUB/PULL pair that lets `pm-log-console` and other live viewers subscribe to rows as they are written — lease duration, subscriber and backfill limits, and the PUB socket's send high-water mark. See LALF-PS for what each setting controls at the protocol level.

At Expert the **Log Server** sub-tab ends with **Log client — every process**: the `log_server.client` settings that every *other* process reads — how long it waits to connect, how long it keeps trying before writing its log to a local file instead, and where. They are written only when one differs from its default.

The **API** sub-tab also takes the instance's **Clearing DB path** (realized P&L on position rows), **Market-data cache** (seconds of recent trades kept so a reconnecting WebSocket client can resume), and the four **engine and index bus addresses** — needed only when the API gateway runs on another host than the engine. Each is written only when set.

The **API** sub-tab includes **Order cache retention (s)** (`order_retention_sec`, default `3600`), which bounds how long a terminal order — `FILLED`, `CANCELLED`, `EXPIRED`, `REJECTED` — stays in the gateway's in-memory cache. That cache backs `GET /orders` and the cross-gateway admin order table, so the setting decides how far back those views reach. Resting orders are never evicted whatever their age. `0` disables eviction entirely, letting the cache grow for the lifetime of the process. See API Gateway → Order cache retention.

8.8.13 Health monitoring

The **Health Monitoring** tab (Intermediate+) edits the `monitor` block read by every process (its health beacon) and by `pm-monitor`, which judges each process and the exchange from those beacons — see Health Monitoring.

Write monitor section off leaves the block out of the file; monitoring then still runs, on the defaults the tab shows. To turn

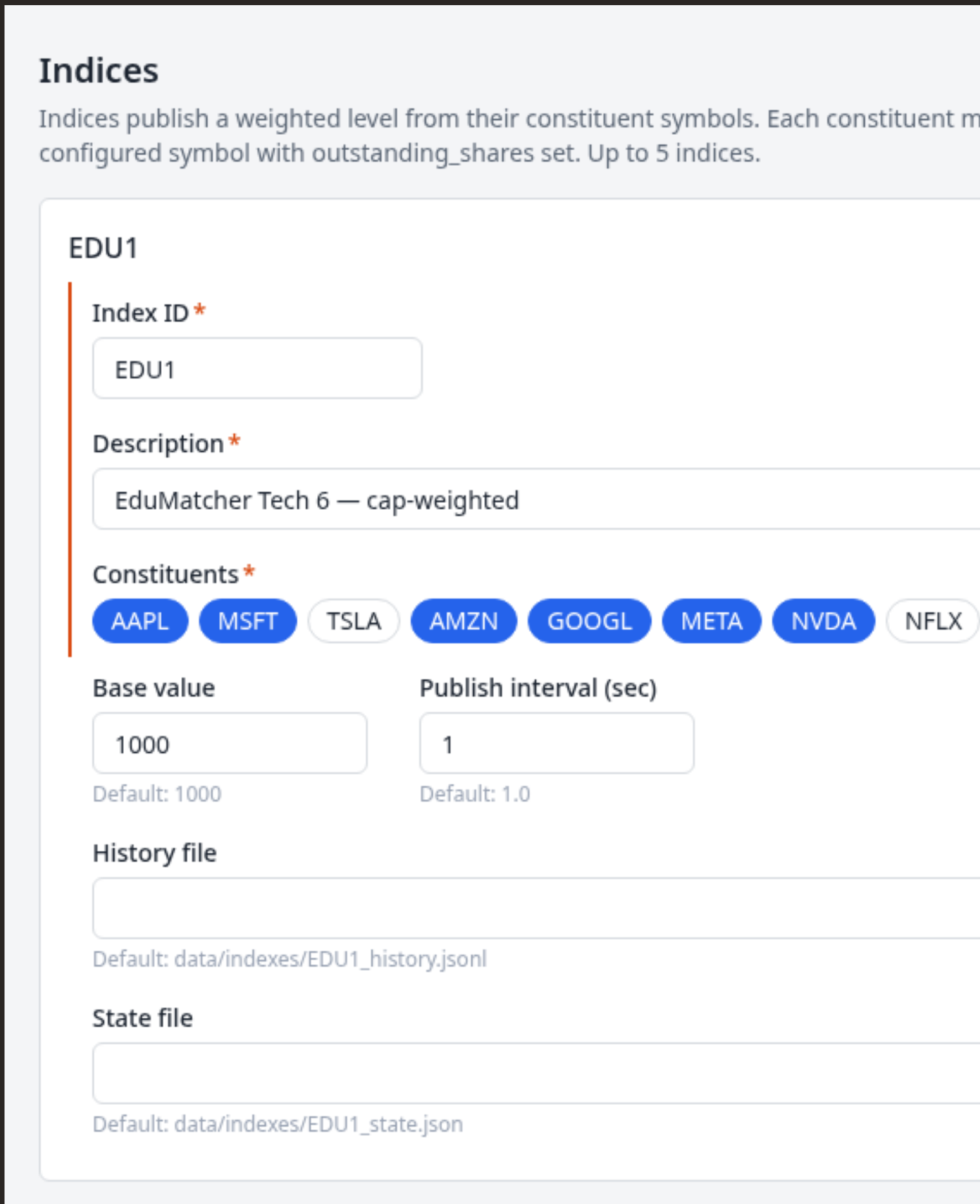


Figure 8.16: The Indices tab with one index.

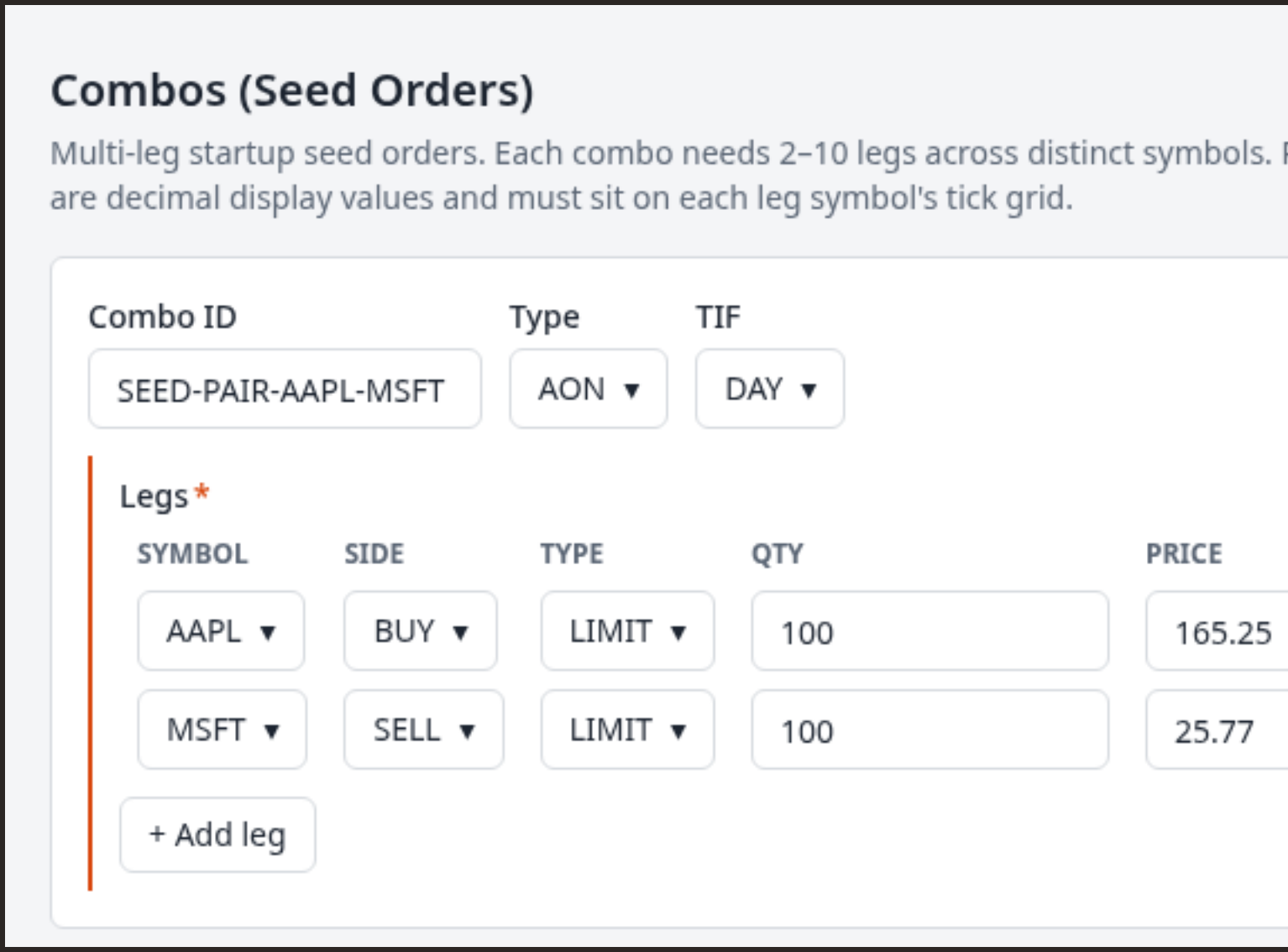


Figure 8.17: The Combos tab.

Auxiliary Gateways

Optional network services around the engine: post-trade (fills/drop-copy), market-d
BALF (binary access), drop-copy TCP relay, the centralized log server, and the REST/W
gateway. Enable only what your scenario needs. Ports are collision-checked across a

ALF | Post-Trade | Market-Data | BALF | Drop-Copy | Log Server | API

REST/WebSocket API gateway instances. Multiple instances can each be scoped to a
ALF gateway IDs.

desk

Instance name (i)

desk

Enabled (i)

Host (i)

0.0.0.0

Port (i)

8080

Credentials (i)

API KEY *		GATEWAY	DESCR
key-trader01-aesiv6	Generate	TRADER01 ▾	Ger
key-trader02-d159c3	Generate	TRADER02 ▾	Ger
key-trader03-ajbe67	Generate	TRADER03 ▾	Ger

monitoring off, write the section and switch **Monitoring enabled** off. With the section written:

Group	Persona	Fields
Liveness	Intermediate	Beacon interval, Hung after, Silent after (beacons)
Liveness	Expert	Book metrics interval
History	Intermediate	Raw samples kept (hours), Minute averages kept (days)
Alert thresholds	Intermediate	Warn and crit for each of the seven rules, each with a reset to its default
Network	Expert	Bind address, Host, Beacon (PULL) port, Publish (PUB) port

***Write monitor section** on, so the values are written; every field shows its default.*

The warn and crit level of each of the seven rules, with what it measures; disk free is the one where lower is worse.

The diagnostics mirror `pm-cverifier`: warn must be below crit (above, for **Disk free**, where lower is worse — `M029`); *Hung after* must exceed the beacon interval (`M030`); a host that cannot reach the bind address is a warning (`M031`); and the two ports must differ. pm-monitor's ports join the cross-gateway port-collision check even when the section is not written, since it then binds its defaults (5610, 5611).

8.8.14 Engine tuning

The **Engine Tuning** tab (Expert) exposes low-level performance knobs that rarely need changing. It mirrors the grouped `engine_tuning` block and currently exposes five fields:

- **Snapshot interval** — `engine_tuning.snapshot_interval_sec`, the minimum seconds between published book snapshots for a busy symbol.
- **Quote history** — `engine_tuning.quote_history_maxlen`, how many recently inactivated quotes per gateway are retained for QLEGS RECENT/ALL.
- **Drop copy buffer** — `engine_tuning.drop_copy_buffer_size`, how many drop-copy events are retained in memory for replay after reconnect.
- **Recent trades** — `engine_tuning.recent_trades_maxlen`, how many recent trade rows each order book keeps for snapshots and diagnostics.
- **Depth snapshot tolerance** — `engine_tuning.depth_snapshot_tolerance_ticks`, the published depth window around the last trade, measured in ticks.

A separate **Auctions** section holds **Indicative price interval** (`auction_indicative_interval_sec`, default `1.0`): how often the engine broadcasts each auctioning symbol's indicative price and volume.

The GUI presents these in the same way as the rest of the application: default values are shown as hints, changing a value marks the field as explicitly set, and the reset control returns it to the engine default.

8.9 Diagnostics and live validation

Every rule runs **live**, on each change, and again before export. Results appear in three coordinated places:

- **Field markers** — a red/amber left border, an inline icon, and a tooltip on the offending field.
- **Tab glyphs** — each tab shows the worst severity it contains (✗ > ! > ✓ > –).
- **Diagnostics drawer** — the global list of every issue, each with **Jump** → to navigate to and briefly highlight the linked field(s). When a rule links two fields (for example a symbol's risk level and the level catalogue), both are highlighted together.

The rules mirror the CLI's own checks (undefined risk level, single gateway, no ADMIN gateway, collars/CB disabled, sessions-with-default-schedule, port collisions, out-of-order schedule) and add GUI-only structural checks (index constituents, combo legs, unknown market-maker obligation symbols, and more).

Health Monitoring

Every pm-* process sends a small health beacon every few seconds; pm-monitor co
each process HEALTHY, DEGRADED, HUNG, SILENT or DEAD, keeps a metrics history
when a threshold is crossed. Shown by pm-monitor-cli and the operator dashboard.

MONITORING

Write monitor section



Monitoring enabled



LIVENESS

How often processes report, and how long a stall may last before pm-monitor cal
(beacons arrive but its main loop has stopped) or SILENT / DEAD (no beacons at a
process is gone).

Beacon interval

2 sec

Default: 2

Hung after

5 sec

Default: 5

Silent after

3 beacons

Default: 3

Book metrics interval

ALERT THRESHOLDS

Crossing warn raises a warning event and makes its subject DEGRADED (a process, status for host, reject and book rules); crossing crit raises a critical one. Warn must except for disk free, where lower is worse.

Rule	Warn		Crit
Host CPU `host_cpu_pct` CPU use of the whole host.	80	%	95
Host memory `host_mem_pct` Memory use of the whole host.	80	%	92
Disk free `disk_free_pct` Free space on the data directory's file system. (lower is worse)	15	%	5
Engine latency p99 `engine_handle_p99_us` 99th-percentile time the engine takes to handle one request.	2000	µs	10000
Reject rate (5 min) `reject_pct_5m` Share of inbound requests (orders, cancels, amends, quotes) the engine rejected over five minutes, judged from 20 requests.	10	%	30
Memory growth (1 h) `rss_growth_pct_1h` Growth of one process's resident memory over the last hour -- a leak. Judged once the process has been watched for 10 minutes.	50	%	100
Book heap garbage `heap_garbage_pct` Share of stale, lazily deleted entries in one symbol's order-book heaps, judged from 100 entries.	50	%	80

Export gate: the Review tab's **Download** is disabled while any **error**-severity diagnostic exists anywhere in the config. Warnings do not block export but require a one-time acknowledgement.

*Two errors and a warning across the whole configuration. **Jump →** on the first error opened the Market Maker tab and outlined both fields it links. The navigation shows the worst severity per tab: on Market Maker, ! on Risk & Collars.*

8.10 Importing an existing configuration

Import (top bar) loads an existing `engine_config.yaml` into the form so you can edit it visually. The GUI maps the file directly into its internal model and opens the Review tab first, reporting the health of what you loaded.

Anything the GUI does not model — a hand-added section, or a field from a newer schema — is preserved verbatim in a per-section **"unmapped YAML" passthrough** and re-emitted unchanged on export, so import → edit → export never silently drops data. A banner marks any tab that carries unmapped content.

A config split with `include:` (from `pm-config-gen --split`, or this GUI's **Separate symbol and participant files** option) is imported by selecting the main file *together with* every file it includes in the same file dialog. The GUI merges them with the engine's rules: lists concatenate, mappings merge one level deep, and a key defined twice is an error. It then turns on **Separate symbol and participant files**, so exporting writes the split again. A split exported by this GUI or by `pm-config-gen --split` round-trips file for file. Any other layout, such as schedules in an included file or different file names, merges just as the engine would. A note then says that export writes the standard layout instead: symbols in `symbols.yaml`, participants in `participants.yaml`, and everything else in the main file. An API gateway's `credentials_file` works the same way: select the file together with the config, and the GUI reads the credentials from it. The path is relative to the file that defines the gateway. The GUI keeps the file name, so the export writes the keys to that file again, not into the main file.

Import refuses what the engine refuses: an include or credentials file path that is absolute or contains `..`, a reserved name (`engine_config.yaml`, `engine_config.json` or `market_sim.yaml` beside the main file), an include listed twice, nested includes, and a gateway with both `credentials` and `credentials_file`. It also refuses a selection it cannot resolve: a file the config names that was not selected, a selected file the config does not name, or two named files with the same name in different directories. The browser sees only file names, not paths.

A file written by an earlier `config-gui` may carry a top-level `country`, which no process reads. Import does not convert it (it is kept as unmapped YAML); set the country again on the Basics tab so it is written under `locale`, where the scheduler looks.

Comments are not round-tripped

The GUI is a *structural* editor, not a text editor. It regenerates the header/comment block fresh (matching `pm-config-gen`'s convention) but does not preserve inline comments from the imported file.

The `s300-sim` file (300 symbols, 112 participants) imports without errors. The summary groups what remains by tab: collar enforcement is off (a warning), sessions use the default schedule, and the universe is large (both informational).

8.11 Reviewing and exporting

The **Review & Export** tab (Beginner) is the final pass:

1. **Diagnostics summary** — the full issue list grouped by tab.
2. **Output options** — the download filename and the "comment default fields" toggle (emits the full commented reference block, matching `pm-config-gen --comment-default-config-fields`; the top bar's **Comment defaults** checkbox is the same setting), **Separate symbol and participant files** (see below), and **API credentials in separate files** (see Auxiliary gateways).
3. **YAML preview** — a live, syntax-highlighted, read-only render of exactly what will be downloaded, with a copy-to-clipboard button. This preview *is* the dry-run view. With separate files, buttons above it switch between the files.
4. **Export actions:**
   - **Download** — writes `engine_config.yaml`, plus the included files when they are separate and any API credentials file (blocked while any error exists; warnings require a one-time acknowledgement).
   - **Verify with pm-cverifier** — optional; see server-side verification.

Separate symbol and participant files matches `pm-config-gen --split`. With it on, the symbols are written to `symbols.yaml` and the participants to `participants.yaml`. The main file lists both under `include:` and leaves the two keys out, which keeps a

EduMatcher pm-config v0.50.7

- Basics ✓
- Sessions & Schedule ✓
- Risk & Collars !
- Circuit Breakers ✓
- Market Maker ✗
- Symbols ✓
- Indices —
- Combos ✓
- Auxiliary Gateways ✓
- Health Monitoring —
- Engine Tuning ✓
- Review & Export ✓

200

Default: 100

QUOTE SEEDING

Set a mid-range and the builder see
gateway. Without it, quote stubs are
engine.

Require MM seed quotes (i)

✗ Symbol AAPL would get a quote stub
explicit quotes for the symbol, or turn o
✗ Symbol AAPL would get a quote stub
explicit quotes for the symbol, or turn o

Seed MM mid-range (i)

to

Default: none (null quote stubs)

✗ Symbol AAPL would get a quote stub
explicit quotes for the symbol, or turn o
✗ Symbol AAPL would get a quote stub
explicit quotes for the symbol, or turn o

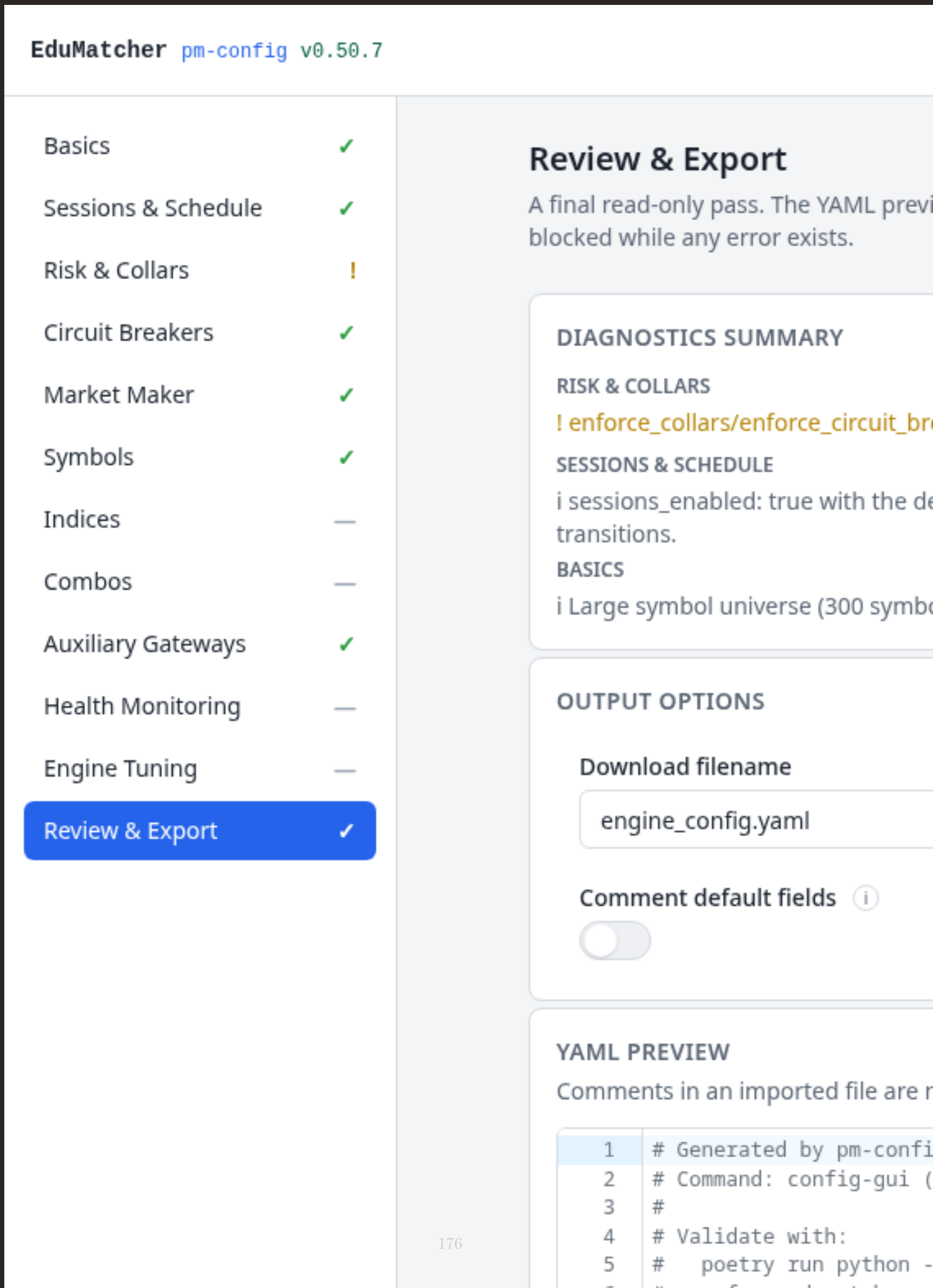
Seed last prices from MM (i)

Set a mid-range first

Seed placeholder last prices (i)

QUOTE STUB REVIEW

What is written for each symbol × m


EduMatcher pm-config v0.50.7
Basics ✓
Sessions & Schedule ✓
Risk & Collars !
Circuit Breakers ✓
Market Maker ✓
Symbols ✓
Indices —
Combos —
Auxiliary Gateways ✓
Health Monitoring —
Engine Tuning —
Review & Export ✓
Review & Export
A final read-only pass. The YAML previ
blocked while any error exists.
DIAGNOSTICS SUMMARY
RISK & COLLARS
! enforce_collars/enforce_circuit_br
SESSIONS & SCHEDULE
i sessions_enabled: true with the de
transitions.
BASICS
i Large symbol universe (300 symbo
OUTPUT OPTIONS
Download filename
engine_config.yaml
Comment default fields
YAML PREVIEW
Comments in an imported file are r
1 # Generated by pm-confi
2 # Command: config-gui (
3 #
4 # Validate with:
5 # poetry run python -

config with hundreds of symbols or participants readable. The engine merges the files back into the same configuration, so **Verify with pm-cverifier** checks the equivalent single file. **Download** saves all three files; your browser may ask once whether to allow multiple downloads. Keep the files in the same directory, and give the main file a name other than `symbols.yaml` or `participants.yaml` (the Review tab flags a clash as an error). `pm-config-deploy` installs the included files together with the main file.

The download is an **authored** `engine_config.yaml` — the file you keep and version, not the one the exchange runs. Install it before starting anything:

```
pm-config-deploy ~/Downloads/engine_config.yaml
```

That validates it, resolves every default, and writes the compiled artifact the processes actually read. Because the GUI already refuses to export while any error exists, a file downloaded from here should deploy first time; if it does not, the deploy output names the finding.

See Configuration — File Location for what compiling does, Running the Engine for starting the processes, and the Quick Start's Your Own Configuration for a first end-to-end run.

The preview is the exact file that will be downloaded — here with collar enforcement off, which is the one remaining warning.

***Download** stays disabled until the warning is acknowledged; **Verify with pm-cverifier** is always available.*

8.12 Deployment and operations

8.12.1 Single container (recommended)

For production the Fastify backend can serve the compiled frontend, so the whole application runs as **one container on one port** — no reverse-proxy juggling between a static host and an API, and no CORS configuration because everything is same-origin. `make up` (or `docker compose up --build`) is the entry point; see Running the application.

The image is multi-stage: it installs dependencies, builds the frontend, prunes dev dependencies, and starts the API with `STATIC_DIR` pointed at the built assets. The API serves the UI and falls back to `index.html` for client-side routes.

8.12.2 Manual production build

To run the pieces yourself:

```
cd web-apps/config-gui
npm install
npm run build           # emits the static UI to apps/web/dist
STATIC_DIR="$PWD/apps/web/dist" HOST=0.0.0.0 PORT=8092 \
  npm run start --workspace @edumatcher/server
```

Alternatively host `apps/web/dist` on any static server and run the API separately; then either reverse-proxy `/api` under the same origin or set `CORS_ORIGIN` to the UI's origin.

8.12.3 Backend environment variables

All optional, read by the Fastify server:

Variable	Default	Purpose
`HOST`	`127.0.0.1`	API bind address (use `0.0.0.0` in containers)
`PORT`	`5192`	API port (`8092` in the container image)
`STATIC_DIR`	*(unset)*	Absolute path to the built UI; enables single-origin/single-container mode
`MAX_IMPORT_BYTES`	`1000000`	Maximum accepted import payload (1 MB)

Variable	Default	Purpose
CVERIFIER_COMMAND	pm-cverifier	Command for the optional verify endpoint, e.g. "poetry run pm-cverifier"
CORS_ORIGIN	*	Allowed CORS origin; restrict on shared deployments
LOG_LEVEL	info	Fastify log level

8.12.4 Building behind a corporate proxy or firewall

The build downloads npm packages from a registry. **The Docker build does not inherit your host's npm or proxy settings** — it has its own network and an empty npm config. Behind a proxy or a TLS-intercepting firewall, `npm install` may **hang** (the connection is silently dropped) rather than fail fast.

First confirm the registry is reachable from inside a container:

```
docker run --rm node:22-slim sh -c "npm config set fetch-timeout 30000 && npm ping"
```

If that hangs, pass build arguments to route around it. `docker compose` picks up your shell's proxy variables automatically:

Build arg	Purpose
HTTP_PROXY / HTTPS_PROXY / NO_PROXY	Route npm through your corporate proxy
NPM_REGISTRY	Use a corporate mirror (Artifactory / Nexus) instead of the public registry
NPM_STRICT_SSL	Set to `false` **only** as a last resort for TLS interception when you cannot install the CA

```
export HTTPS_PROXY=http://proxy.corp.example:8092
export NPM_REGISTRY=https://artifactory.corp.example/api/npm/npm-remote/
docker compose up --build
```

TLS interception (custom root CA)

If your proxy re-signs TLS with a corporate root CA, the cleaner fix than disabling `strict-ssl` is to trust that CA: copy the CA `.crt` into the image and set `NODE_EXTRA_CA_CERTS=/path/to/ca.crt` before `npm install`. The build also sets a fetch timeout, so a blocked network now fails in about two minutes with a clear error rather than hanging indefinitely.

8.12.5 Optional: server-side verification

The Review tab's **Verify with pm-cverifier** button POSTs the generated YAML to the backend, which shells out to `pm-cverifier` and renders its authoritative report inline. It is optional and pluggable:

- If the tool is not installed, the endpoint returns `503` and the UI shows a clear "not available" message; nothing else is affected. The default container image does **not** include the Python toolchain, so this button is inactive there by design.
- To enable it, run the backend where `pm-cverifier` is on `PATH`, or set `CVERIFIER_COMMAND="poetry run pm-cverifier"`.
- The backend invokes the tool with a fixed argument array over a temporary file — never a shell string — so no user input is interpolated into a command line.

8.12.6 Security notes

This is primarily a single-user local tool. Before exposing it on a shared network:

- Drafts and any generated API keys live **only in the browser** (`localStorage`) and in the file you download. The backend does not persist drafts and does not log request/response bodies containing credentials.
- The only subprocess-spawning endpoint is `POST /api/config/verify`. Put it behind authentication and rate limiting on shared deployments, and set a restrictive `CORS_ORIGIN`.
- Import payloads are size-capped (`MAX_IMPORT_BYTES`) and parsed with a safe YAML schema.

Review & Export

A final read-only pass. The YAML preview below is exactly what will be downloaded. D
blocked while any error exists.

DIAGNOSTICS SUMMARY

RISK & COLLARS

! enforce_collars/enforce_circuit_breakers disabled. Suitable for tests only.

OUTPUT OPTIONS

Download filename

engine_config.yaml

Comment default fields

YAML PREVIEW

Comments in an imported file are not preserved on export; the GUI is a structural

```yaml
# Generated by pm-config-gen v1.0.0-gui on 2026-10-20
# Command: config-gui (interactive)
#
# Validate with:
#   poetry run python -c 'from pathlib import Path; \
#     from edumatcher.engine.config_loader import load_engine_con
#     print(load_engine_config(Path("engine_config.yaml")))'

# -- Session control --
# true lets pm-scheduler drive session transitions;
# false keeps the engine in continuous mode
sessions_enabled: true

# -- Engine behavior --
```

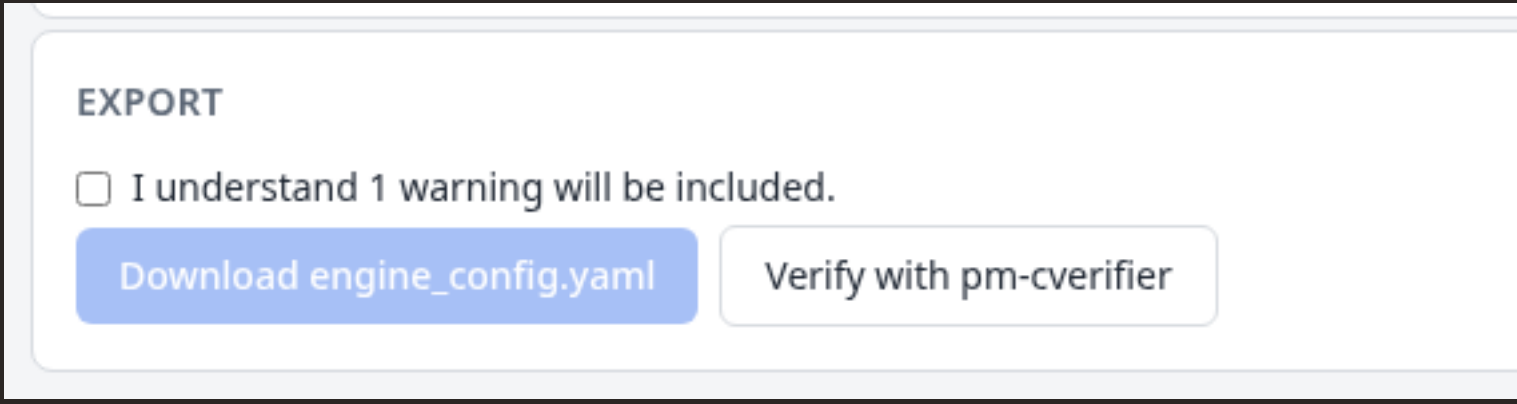
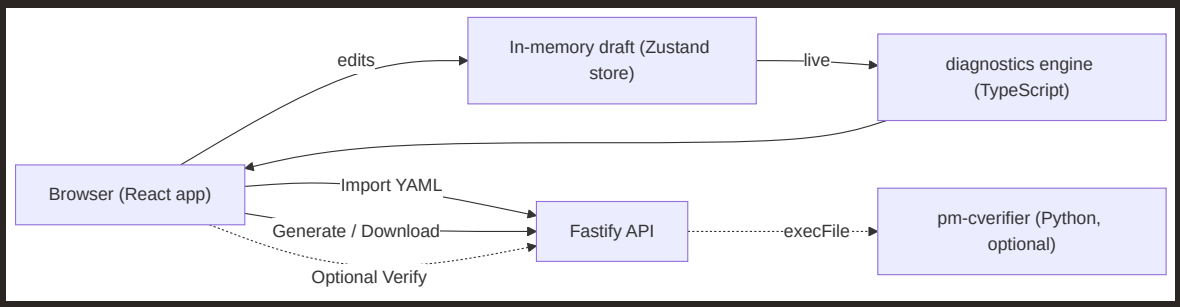


Figure 8.24: The Export actions while a warning is unacknowledged.

8.13 How the GUI stays valid

The GUI re-implements the `engine_config.yaml` schema in TypeScript and serializes it with the same section ordering and comment conventions as `pm-config-gen`. Because the format now has two implementations, a golden-file test (`npm run verify:python`) generates representative configs and runs each through the engine's real `load_engine_config()`. Keeping that check green is what guarantees GUI output loads in the engine. The developer checklist for keeping the two in sync lives in the `config-gui/README.md`.

8.14 Architecture at a glance



The application is two processes: a **web** frontend (Vite + React) that in development proxies `/api/*` to the backend, and a **server** (Fastify) that imports, validates, generates, and — optionally — verifies. In production the server can also serve the built UI, collapsing the two into one container.

8.15 Troubleshooting

Symptom	Cause	Fix
`docker compose up --build` hangs at `RUN npm install`	Corporate proxy/firewall blocks the registry; the build does not inherit host settings	Confirm with `docker run --rm node:22-slim sh -c "npm config set fetch-timeout 30000 && npm ping"`, then pass proxy/registry build args — see Building behind a corporate proxy.
`make up` fails: neither podman nor docker found	No container runtime on the host	Install Podman or Docker, or use local development.
`make up` on macOS: podman machine not running	The Podman VM is not started	`make up` starts it automatically; otherwise `podman machine init && podman machine start`.

Symptom	Cause	Fix
API calls fail in dev (ECONNREFUSED / 404 on /api)	The backend is not running	Start npm run dev:server; the web dev server proxies /api to http://127.0.0.1:5192.
"root" option must be an absolute path on startup	STATIC_DIR is a relative path	Use an absolute path, e.g. STATIC_DIR="$PWD/apps/web/dist".
Blank page in production, API works	UI not built or STATIC_DIR wrong	Run npm run build, then point STATIC_DIR at apps/web/dist.
Client route (e.g. /review) 404s in production	Static host without SPA fallback	Let the API serve the UI (STATIC_DIR) — it falls back to index.html.
"pm-cverifier is not available" in the Review tab	Verifier not on PATH (expected in the default container)	Optional; set CVERIFIER_COMMAND="poetry run pm-cverifier" or run where the tool is installed.
npm run verify:python cannot import edumatcher	Python env not installed	Run poetry install at the repository root.
Import rejected as too large	File exceeds MAX_IMPORT_BYTES (1 MB)	Raise the limit via the env var, or trim the file.
Port already in use	Another process holds 8192/5192/8092	Change PORT (API) or server.port / proxy target in web-apps/config-gui/apps/web/vite.config.ts (web).
Imported config shows an "unmapped" banner	The file has sections the GUI does not model	Expected — those sections are preserved read-only and re-emitted unchanged.
Quote Stub Review shows "! fill in" after import	No mid-range seeding and no explicit quotes for some symbols	Set a mid-range on the Market Maker tab, or enter explicit bid/ask on each flagged symbol's MM Quotes sub-tab.
A tab you expected is missing	It is above the current persona	Raise the persona.
Market Maker tab is greyed out	No MARKET_MAKER gateway is defined yet	Add a MARKET_MAKER gateway on the Basics tab.

8.16 See Also

In this book:

- Engine Configuration — the engine_config.yaml format and the pm-config-gen CLI
- Config Verifier (pm-cverifier) — the authoritative validator behind the Verify button
- Risk Controls — collars, circuit breakers, and day-one (IPO) behaviour
- Auctions & Session Scheduling — session phases and pm-scheduler
- Market Index — the Indices tab
- Example Engine Configs — reference configurations to import and study
- Running the Engine — using the file you exported

Participant Guide:

- Market Making — obligations, quote lifecycle, and startup seeding
- Order Types — how orders interact with collars and circuit breakers
- Combo Orders — the Combos tab

CHAPTER 9

Example Engine Configs

Learning objectives

After reading this chapter you will understand:

- which ready-to-use engine configuration templates are available under `docs/examples/ref_data`
- how those templates were generated
- how the different book-count and setup profiles are intended to be used in training scenarios

9.1 Scope

The `docs/examples/ref_data` tree contains generated `engine_config.yaml` templates that can be used directly for training and workshop exercises.

These reference configs are organized by book count and setup profile:

- `s1-basic-setup`
- `s1-nominal-setup`
- `s1-complex-setup`
- `s3-basic-setup`
- `s3-nominal-setup`
- `s3-complex-setup`
- `s10-basic-setup`
- `s10-nominal-setup`
- `s10-complex-setup`
- `s30-basic-setup`
- `s30-nominal-setup`
- `s30-complex-setup`
- `s150-basic-setup`
- `s150-nominal-setup`
- `s150-complex-setup`
- `s300-load-setup` — AI-trader design-limit load test bed: 300 symbols, 500 AI participants (`AI001`–`AI500`); no `-nomm` variant
- `s300-sim-setup` — the long-running simulated market used by the AI Traders and Market Simulation book and the `edumatcher-ai-sim` container image's default config: 300 symbols, 100 AI traders (`AI001`–`AI100`) plus a `market_sim.yaml` hidden-true-value market model with news/rumours; no `-nomm` variant

The `s150-*` configs also list `AI001`–`AI020` so a 20-bot `pm-ai-swarm` runs unchanged. Each profile also has a `-nomm` variant without seeded market-maker quotes. Every directory contains a small generator script, `mkrefdata.sh`, and the resulting `engine_config.yaml`.

9.2 How they are generated

The example configs are produced by running `mkrefdata.sh` in each profile directory. The scripts invoke `pm-config-gen` with the appropriate symbol set and gateway layout, then apply a small amount of profile-specific post-processing where needed.

In practice this means:

- the engine config structure is generated by the same config generator used elsewhere in the project
- RALF and CALF gateway sections are emitted natively by `pm-config-gen`
- every symbol includes `outstanding_shares` data for statistics and future index-style consumers
- the basic profiles use `--seed-mm-mid-range` and `--seed-last-prices-from-mm` so each book starts with a reproducible bootstrap market-maker quote and matching last-price references
- the nominal and complex profiles include richer session, risk, clearing, and market-data settings
- complex profiles define a fuller trader set and market-maker bootstrap inputs for more involved scenarios

Several of the generator scripts accept `--seed` so the bootstrap prices can be reproduced when needed.

9.3 Training usage

These templates are ready to use in training scenarios as soon as they are generated.

- Basic profiles are suitable for first-touch configuration, startup, and book-entry exercises.
- Nominal profiles are suitable for normal trading-day walkthroughs with market-data and post-trade enabled.
- Complex profiles are suitable for advanced training that combines multiple traders, market-making, scheduling, risk controls, and market-data scenarios.

For example, a training session can start from one of the generated `engine_config.yaml` files instead of assembling a config manually.

9.4 Related material

- Reference data templates live under `docs/examples/ref_data`
- Protocol support examples: Protocol Support Library Examples
- Installation and setup training: 00 — Installation & Setup

CHAPTER 10

Running the Exchange

Learning objectives

After reading this page you will understand:

- The three ways to run the exchange — containers, `pm-opctl-cli`, or one terminal per process — and which one fits what you are doing
- How to prepare a session before the first process starts
- Why the deployed configuration artifact is the operational source of truth
- Which processes to start for a minimum session, a recorded session, a classroom session, and an externally connected session
- How to verify that the exchange is healthy after startup
- How to monitor, troubleshoot, restart and shut down a running exchange
- Which operator playbook to follow for common scenarios

Prerequisites: work through the Quick Start Guide first. For configuration syntax and validation rules, read The Configuration Workflow. For the full process catalog, read Processes, Environment and Ports.

10.1 Operator model

EduMatcher is a multi-process exchange. `pm-engine` owns the order books and binds the core ZeroMQ sockets. Every other process connects to it, sends commands, subscribes to events, records data, or exposes an external interface.

The operator's job is to make five things true before trading starts:

1. Every process sees the same `EDUMATCHER_DATA_DIR`.
2. The authored `engine_config.yaml` has been validated and deployed.
3. Loss-sensitive recorders are running before the first meaningful event.
4. Gateways and external feeds are started only after the engine is ready.
5. There is a clear shutdown and recovery plan.

For the deep architectural explanation, see Processes, Environment and Ports. This chapter is the practical runbook.

10.2 Three ways to run it

The processes are the same in all three. What differs is who starts them, and where they live.

	Containers	`pm-opctl-cli` on the host	One terminal per process
Start the exchange	`./edumatcher.sh start` or `make up-all`	`pm-opctl-cli start`	8–14 commands, in order
Web applications	four by default, `config-gui` opt-in[1]	started separately	started separately
Process table	`make status`	`pm-opctl-cli list`	`ps`, and your own notes

	Containers	`pm-opctl-cli` on the host	One terminal per process
Process logs	`data/emo/<name>.log`	`<DATA_DIR>/emo/<name>.log`	each terminal's scrollback
Best for	classrooms, demos, anyone who wants it *running*	scripted or repeated local runs	learning what each process does, and debugging one of them

[1] The curl installer's `./edumatcher.sh start` brings up all six web applications unconditionally. From a source checkout, `make up-all` starts the five GUIs that talk to a running exchange (`terminal-gui`, `log-gui`, `trader-gui`, `book-gui`, `dash-gui`); `config-gui` — which needs no running exchange — is opt-in via `make up-all CONFIG_GUI=1`. See Installation.

If you want a running exchange, use the containers. That is Installation; the rest of this chapter still applies, because the container runs exactly these processes:

```
cd ~/.edumatcher && ./edumatcher.sh start     # a released install
cd deployment/docker && make up-all           # from a checkout
```

If you are learning the system, start processes by hand. Nothing below assumes a container, and the manual startup order is the best way to see why each process exists and what it depends on.

The container is not a different system

Inside the container it is `pm-opctl-cli` starting the same `pm-*` processes, reading the same deployed artifact, writing the same databases into a directory on your own disk. Every operator command, CLI and troubleshooting step in this chapter works there too — `make shell`, or `./edumatcher.sh shell`, puts you at a prompt inside it.

10.3 Runtime source of truth

Modern EduMatcher separates the file you edit from the file the exchange runs.

File	Who uses it	Operator action
`engine_config.yaml`	Humans, source control, review tools, config generators	Edit and review this file
`<EDUMATCHER_DATA_DIR>/ref_data/engine_config.json`	Every running `pm-*` process	Install it with `pm-config-deploy`; do not edit by hand

No runtime process accepts a config path. The engine, scheduler, gateways, recorders, market-data gateway, API gateway, log server and index process all read the deployed artifact from the data directory. That prevents one process from accidentally running against a different file from the rest of the exchange.

The practical rule is simple:

```
pm-config-deploy --check engine_config.yaml  # validate only
pm-config-deploy engine_config.yaml          # validate, compile, install
pm-config-deploy --show                      # print deployed paths
```

After deployment, restart any running process that must pick up the change. Deploying a new artifact is atomic, but it does not hot-reload processes that already loaded the previous artifact.

Do not skip deployment

Editing `engine_config.yaml` is not enough. A running exchange reads `<EDUMATCHER_DATA_DIR>/ref_data/engine_config.json`. If startup logs warn that the authored source changed after deployment, deploy again and restart.

10.4 Running modes

EduMatcher runs installed on the host, from a source checkout, or in a container. The behaviour is the same; the command prefix and the default data directory differ.

	Installed mode	Developer mode	Container
Typical user	Instructor, student, demo operator	Contributor, test runner, docs author	Anyone who wants the whole system, GUIs included
Install	`pipx install edumatcher`	`poetry install --with dev,docs`	`install.sh`, or `make up-all` from a checkout
Command style	`pm-engine --verbose`	`poetry run pm-engine --verbose`	`make shell`, then `pm-engine --verbose`
Default data directory	`~/.local/share/edumatcher`	`<repo>/src/data/`	`/data` inside; `~/.edumatcher/data` or `deployment/docker/data` outside
First setup	`pm-setup`	Usually none, but deployment is still recommended	Done by the entrypoint on first start

Throughout this chapter commands are shown in installed form. In developer mode, prefix each `pm-*` command with `poetry run`. In a container, run them inside it — `make shell` or `./edumatcher.sh shell` — where they are already on `PATH` and `EDUMATCHER_DATA_DIR` is already `/data`.

10.5 Data directory

`EDUMATCHER_DATA_DIR` is the one location knob for a running exchange. Set it once per shell, service unit, tmux session, container, or launcher.

Path under `EDUMATCHER_DATA_DIR`	Written or read by	Purpose
`ref_data/engine_config.json`	all processes	compiled runtime configuration
`ref_data/engine_config.yaml`	`pm-config-deploy`	copy of the source used to build the artifact
`stats.db`	`pm-stats`, `pm-stats-cli`, API history reads	OHLCV, trades, midpoint and related statistics
`clearing.db`	`pm-clearing`, `pm-clearing-cli`	positions, trades, P&L summaries
`audit.log` or configured audit path	`pm-audit`, `pm-audit-cli`	event audit trail
`log.db`	`pm-log-srv`, `pm-log-cli`	centralized operational logs
`metrics.db`	`pm-monitor`, `pm-monitor-cli`, the operator dashboard	health history: samples, daily totals, events
`gtc_orders.json`, `gtc_combos.json`	`pm-engine`	clean-shutdown persistence for GTC state

Example per-session isolation:

```
export EDUMATCHER_DATA_DIR="$HOME/edumatcher-sessions/morning"
mkdir -p "$EDUMATCHER_DATA_DIR"
pm-config-deploy ./configs/morning.yaml
pm-engine --verbose
```

For the complete file map, see Persistence -> Data files at a glance.

Where this is in a container install

`EDUMATCHER_DATA_DIR` is `/data` inside the container, bind-mounted from a directory on your disk — `~/.edumatcher/data` for a released install, `deployment/docker/data` from a checkout. Everything in the table above is therefore a real file you can open, back up and delete with your own tools; `./edumatcher.sh mounts` (or `make mounts`) prints which host directory is

behind each container path, which is the fastest way to settle "*which* exchange's data am I looking at".

10.6 Preflight checklist

Run this before a classroom, demo, test session, or integration exercise.

Check	Command or question	Why it matters
Correct shell mode	`which pm-engine` or `poetry run pm-engine --version`	Confirms whether commands are installed or source-prefixed
One data directory	`echo "$EDUMATCHER_DATA_DIR"`	Prevents split stats, logs and config
Authored config validates	`pm-config-deploy --check engine_config.yaml`	Catches YAML, schema and semantic errors before startup
Config is deployed	`pm-config-deploy engine_config.yaml`	Installs the artifact every process reads
Deployed paths are expected	`pm-config-deploy --show`	Confirms where the runtime artifact lives
Ports are free	`lsof -i :5555 -i :5556 -i :5557`	Finds an old engine before bind failure
Recorder policy is clear	Decide whether `pm-stats`, `pm-audit`, `pm-clearing` start before trading	Missed early events cannot always be reconstructed
Timezone is consistent	Set `locale.session_timezone` in the deployed config, or leave it at the default	Both `pm-stats` and `pm-clearing` read it from the same config, so daily reports always reconcile
Operator gateway exists	Confirm an `ADMIN` gateway ID if using halts/resumes	Admin-only commands require an admin role
External clients are expected	Decide whether to start ALF/BALF/CALF/RALF/API/DC gateways	Avoid exposing unused ports

Prefer a clean rehearsal

For a new class or public demo, run the full startup once with `pm-scheduler --now --delay 5`, make one trade, query stats and clearing, then shut down cleanly. It is much easier to fix a config in rehearsal than while participants are waiting.

10.6.1 Preflight for a container install

Most of the checklist above is handled for you: the entrypoint deploys the configuration, `EDUMATCHER_DATA_DIR` is fixed, and one container means one data directory. What is left is shorter:

Check	Command	Why it matters
The right configuration is deployed	`make config-show` (or `./edumatcher.sh shell` then `pm-config-show`)	`EM_CONFIG` picks a bundled example; `EM_CONFIG_FILE` overrides it. `./edumatcher.sh config <name>` *selects* a configuration and requires an argument — it does not show what is currently deployed
Every process came up	`./edumatcher.sh status` or `make status`	Runs `pm-opctl-cli list` inside the container
It is *your* exchange on those ports	`./edumatcher.sh mounts` or `make mounts`	A released and a source-built stack use the same container names and host ports

Check	Command	Why it matters
The applications answer	open 8090, 8091, 8093, 8094, 8095	The two-phase start can succeed for the backend and still leave a GUI unhealthy
Timezone matches the calendar	`TZ` in `.env`	Set it before the session; it affects every timestamp and the trading date

10.7 Minimum viable run

The absolute minimum exchange is one engine and one or more order-entry gateways. This is enough to learn order flow, but it is not enough for an operator who needs records afterwards.

Open one terminal per process, or use `tmux`/`screen`.

10.7.1 Step 1 - start the engine

```
pm-engine --verbose
```

Wait until the engine reports that it loaded the deployed configuration and is listening on the core sockets.

Typical signals:

```
Loaded deployed config .../ref_data/engine_config.json
Session handling: enabled (startup state: CLOSED)
Drop copy PUB bound on port 5557
Listening on PULL=tcp://127.0.0.1:5555  PUB=tcp://127.0.0.1:5556
```

The exact wording may vary by release, but the important facts are: config loaded, session mode known, and sockets bound.

10.7.2 Step 2 - connect two traders

```
pm-alf-console --id TRADER01
pm-alf-console --id TRADER02
```

The gateway IDs must exist under `participants` in the deployed configuration, unless the engine is intentionally running unrestricted with no deployed config.

10.7.3 Step 3 - submit a test order

From one gateway:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
```

An accepted order proves the path from gateway to engine and back is alive. A fill requires a crossing order or resting liquidity on the other side.

Minimum is not operator-safe

If only the engine and gateways are running, there is no durable audit log, no statistics database and no P&L database. That may be fine for a five minute demo, but it is usually not enough for a real exercise.

10.8 Starting the stack with `pm-opctl-cli`

Starting eight to fourteen processes by hand, in the right order, is exactly the kind of thing that goes wrong five minutes before a class. `pm-opctl-cli` does it for you. It is not a container tool — it works the same on the host — and it is what the container itself runs internally.

```
pm-opctl-cli start          # start the 'default' profile
pm-opctl-cli start mini     # ... a smaller one
pm-opctl-cli list           # one row per process, with health
pm-opctl-cli health -q      # exit 0 when everything is running
pm-opctl-cli stop           # stop what it started
```

`start` is idempotent: a process it finds already running is left alone and reported as such, so it is safe to re-run after adding one by hand.

10.8.1 The six built-in profiles

Profile	Processes	Use it for
`micro`	`pm-log-srv`, `pm-engine`	The smallest thing that can match an order
`mini`	`micro` plus `pm-monitor`, `pm-stats`, `pm-scheduler`, `pm-md-gwy`, `pm-api-gwy` (desk), `pm-alf-gwy`, `pm-ralf-gwy`, `pm-dc-gwy`	A trading-capable venue with external access, without clearing or audit
`default`	`mini` plus `pm-audit`, `pm-clearing`, `pm-index`, `pm-balf-gwy` and the second `pm-api-gwy` instance (`dashboards`)	Everything — the operational baseline below
`mm-demo`	`default` plus one `pm-mm-bot` (`passive` strategy, gateway `MM01`, every symbol of the deployed configuration)	Showing students a market maker at work; the default of the `deployment/curl` installer
`ai-swarm`	`mm-demo` with its market maker anchored to the model (`--anchor-sim 0.3`), plus `pm-market-sim` and `pm-ai-swarm --count 20 --budget 40 --symbols-per-agent 1000` (AI traders `AI001`–`AI020`, every built-in preset, each on every symbol)	A market with background flow and news; needs a config that registers `AI001`–`AI020` and ships a `market_sim.yaml` (the `s150` examples, `s300-load`)
`ai-sim`	`ai-swarm` on the `s300-sim` example, set up to run for weeks: `AI001`–`AI100`, a symmetric market maker, quieter logs and bounded statistics	The long-running simulation of the AI Traders and Market Simulation book and the `edumatcher-ai-sim` image

The built-ins are used as they are when no configuration file exists. `pm-opctl-cli init` writes all six to `<DATA_DIR>/emo-config.yaml`, and once that file exists **its profiles replace the built-ins entirely**. That is the supported way to add a process, change a flag or define a profile of your own.

Each entry in a profile is a `name`, a `command`, and optionally a `healthcheck` command or a `tcp host:port` to probe.

10.8.2 What it gives you that a row of terminals does not

- **Ordering.** Processes are started in the profile's order, which encodes the dependencies described below. It does not wait for each one to become healthy — for a slow first start, check `pm-opctl-cli list` before inviting participants in.
- **A process table.** `list` reports each process as `running`, `not responding` (the PID is alive but its check failed) or `dead`. A `healthcheck` outranks a `tcp` probe, because a TCP connect to a ZeroMQ socket proves only that libzmq's I/O thread is accepting — not that the application is still processing messages.
- **Log files instead of scrollback.** Combined stdout and stderr are appended to `<DATA_DIR>/emo/<name>.log`, with PID files beside them.
- **A stop that stops only what it started**, from those PID files. `kill` is the blunt instrument: it sends `SIGTERM` to *every* process whose command line contains `pm-`, including ones it never started.

`pm-opctl-cli` does not start the browser applications. Those are containers (`make up-all`) or a local dev server — see Installation.

In a container, use the Makefile wrappers

`make status`, `make health` and `make proc-logs P=engine` in `deployment/docker` run `pm-opctl-cli` inside the container for you, so you do not need a shell for the routine checks.

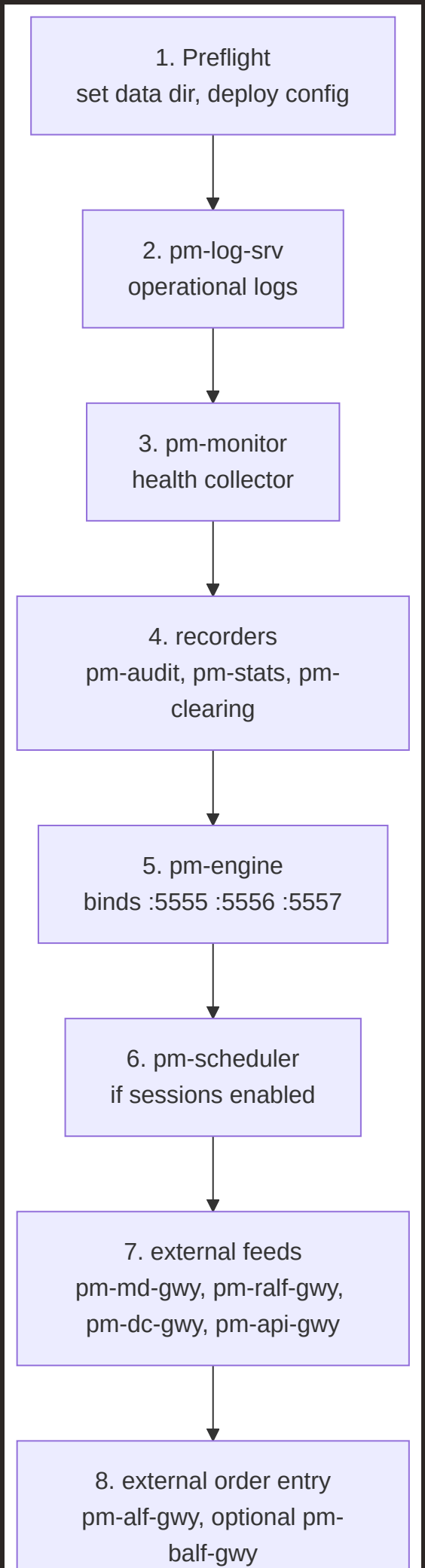
10.9 Recommended startup order

For any session where records matter, use the **operational baseline** below. This is stricter than the technical minimum. The technical minimum is still just `pm-engine` plus one or more order-entry clients, but that is not enough for a properly operated class, demo, test venue, integration environment, or production-like rehearsal.

This baseline *is* the `default` profile: `pm-opctl-cli start` and a container start both perform this sequence. Read this section to understand why the order is what it is — and follow it by hand when you want to watch each process come up, or when you are debugging one of them.

The ordering has two principles:

1. Start observability before the engine emits anything worth preserving.
2. Start the engine before any process that drives state transitions or exposes live services to users.


1. Preflight
set data dir, deploy config
2. pm-log-srv
operational logs
3. pm-monitor
health collector
4. recorders
pm-audit, pm-stats, pm-clearing
5. pm-engine
binds :5555 :5556 :5557
6. pm-scheduler
if sessions enabled
7. external feeds
pm-md-gwy, pm-ralf-gwy, pm-dc-gwy, pm-api-gwy
8. external order entry
pm-alf-gwy, optional pm-balf-gwy

Order	Process	Typical command	Required options	Why here
0	Preflight	`pm-config-deploy --check engine_config.yaml` then `pm-config-deploy engine_config.yaml`	`EDUMATCHER_DATA_DIR` set consistently	Every later process reads the deployed artifact; do this before anything long-running starts.
1	`pm-log-srv`	`pm-log-srv`	Optional `--host`, `--port`, `--db`; usually none	Operational logs should have somewhere to go before other long-running services start. No process fails to start without it — each one queues log records and falls back to a local log file after a 30-second grace window — but starting it first keeps every process's logs centralized instead of scattered on disk.
2	`pm-monitor`	`pm-monitor`	None; everything comes from the deployed config's `monitor` section	Health collector. After the log server so it can log, before the engine so the engine's first beacons are received. Nothing depends on it: every process sends its beacon without waiting, and without `pm-monitor` the beacons are simply dropped. Check it with `pm-monitor-cli health`, and the whole exchange with `pm-monitor-cli status` — see Health Monitoring.
3	`pm-audit`	`pm-audit --terminal`	Optional `--audit-log-file`; use `--terminal` when watching live	Audit is the durable event trail. Start it before the engine so initial session, seed and trade events are not missed.

Order	Process	Typical command	Required options	Why here
4	`pm-stats`	`pm-stats`	Optional `--snapshot-interval`; the session timezone comes from the deployed config's `locale.session_timezone`	Statistics powers reports, history and many displays. Start it before trades so OHLCV and history are complete.
5	`pm-clearing`	`pm-clearing`	Reads the same `locale.session_timezone` as `pm-stats` from the deployed config, so the two always agree	Clearing records trades, positions and P&L. Start it before trading; daily reconciliation depends on timezone consistency.
6	`pm-engine`	`pm-engine --verbose`	Usually none; all config comes from the deployed artifact	The engine owns the books and binds `:5555`, `:5556`, `:5557`. Start it after subscribers that must not miss early events.
7	`pm-scheduler`	`pm-scheduler` or `pm-scheduler --now --delay 5`	Only needed when scheduled sessions are enabled	The scheduler drives phase transitions. Start it after the engine so transitions have a live target and before participants are invited to trade.
8	`pm-md-gwy`	`pm-md-gwy`	Optional `--bind`, `--port`, `--engine-pub`	CALF market data is the live feed used by external clients and TapeDeck. Start after the engine is alive.
9	`pm-ralf-gwy`	`pm-ralf-gwy`	Optional `--bind`, `--port`, `--engine-pub`	Post-trade dissemination (RALF). Start after the engine is alive.
10	`pm-dc-gwy`	`pm-dc-gwy`	Optional `--bind`, `--port`, `--engine-dc-pub`	Drop-copy feed (DC1). Start after the engine is alive.
11	`pm-api-gwy`	`pm-api-gwy`	Optional `--instance NAME`, `--host`, `--port`, `--engine-host`; API keys come from config	The API gateway has no `--id`. Use `--instance` only when multiple `api_gateways` entries are configured. Start after engine and stats history are available.

Order	Process	Typical command	Required options	Why here
12	`pm-alf-gwy`	`pm-alf-gwy`	Optional `--bind`, `--port`, `--engine-host`; gateway IDs come from client `HELLO` and config	The ALF TCP gateway has no process-level `--id`. Start after the engine is healthy, then external text clients can connect.
13	`pm-balf-gwy`	`pm-balf-gwy`	Optional `--bind`, `--port`, `--engine-host`; identity is configured/client-provided	Optional binary order-entry gateway. Start only for BALF client exercises or integrations.
14	`pm-index`	`pm-index`	Index definitions in deployed config	Started last in the `default` profile; index publications depend on the engine and its own configured indices, not on any other process here.
15	Web applications	`cd deployment/docker && make up-all`	None — `up-all` resolves the read-only API key and the backend hostname itself	TapeDeck needs `pm-md-gwy` for live data and `pm-api-gwy` for history; the order book viewer needs `pm-api-gwy` for everything; the log console needs `pm-log-srv`; the operator dashboard needs `pm-monitor`. Starting them last is why `up-all` runs in two phases.

This order is approximate for independent consumers, but not arbitrary. The recorders can safely wait for the engine to appear, so starting them first is a good habit when completeness matters. The scheduler and external gateways should wait until the engine is actually up. Browser UIs should be last because they depend on the services underneath them.

Running the web applications against a host install

`make up-all` starts the backend *and* the applications as one container stack, which is the simplest thing when nothing else is running. To point the applications at an exchange you started by hand instead, run them from `web-apps/<app>/` — see The Development Loop, whose "hybrid" setup is exactly this case.

Where are participant terminals?

Local interactive participant terminals (`pm-alf-console --id TRADER01`) are not part of the baseline service stack. Start them after step 7, once the engine, scheduler policy and operator checks are ready. For supervised sessions, start `pm-admin --id OPS01` before inviting traders in.

10.10 Starting each process

This section gives operational startup commands. The full flag reference for each command lives in Processes, Environment and Ports.

10.10.1 Core engine

```
pm-engine --verbose
```

Use `--verbose` during learning, demos and incident work. For a quiet long run, use the default warning-level logging.

10.10.2 Loss-sensitive recorders

Start these before the first trade if you want complete records:

```
pm-audit --terminal
pm-stats
pm-clearing
```

Both read the session timezone from the deployed config's `locale.session_timezone`, so they always agree. If the exchange runs in UTC, set `session_timezone: UTC` under `locale:` in the config.

Both recorders grow without limit unless you tell them otherwise. The options that matter for a long-running exchange:

Process	Option	Default	Effect
`pm-stats`	`--keep-days N`	`0` (keep all)	Keep N days of `trade_log`, `order_events`, `price_snapshots` and `feed_gaps`; daily rollups are kept for good
`pm-stats`	`--keep-order-events-days N`	as `--keep-days`	Separate (shorter) retention for order events, which dominate the file under an AI swarm
`pm-stats`	`--human-gateways GLOB,…` and `--keep-human-order-events-days N`	none	Give the order events of people (e.g. `'TRADER*'`) a longer retention than bots
`pm-clearing`	`--retention-days N`	`90`	Delete individual `trade_events` rows older than N days at startup and at every close; `0` disables pruning. Daily summaries are kept
`pm-clearing`	`--print-every N`	`100`	Print a P&L table every N trades; `0` never
`pm-clearing`	`--flush-size N`, `--flush-interval SEC`	`100`, `5.0`	Write the buffered trades to SQLite when N have accumulated (1–100) or SEC seconds (0.1) have passed
`pm-clearing`	`--datapath PATH`, `--db-name NAME`	data directory, `clearing.db`	Where the database lives

See Statistics and Reporting for the `pm-stats` retention rules.

10.10.3 Session scheduler

```
# Normal schedule from deployed config, or built-in defaults if no config exists
pm-scheduler

# Rehearsal mode: run the whole trading day quickly
pm-scheduler --now --delay 5
```

The scheduler sends session transitions to the engine. It does not match orders and it does not replace the engine's risk checks.

10.10.4 Interactive traders and operators

```
pm-alf-console --id TRADER01
pm-alf-console --id TRADER02
pm-admin --id OPS01
```

`pm-admin` can connect with any configured gateway ID for read-only and gateway-scoped commands. Exchange-wide halt/resume and symbol-wide mass cancel commands require a gateway with `role: ADMIN`.

10.10.5 Terminal observers

```
pm-viewer --symbol AAPL
pm-orders
pm-board
pm-ticker --db-interval 15
```

`pm-board` and `pm-ticker` become much more useful when `pm-stats` is already running and has observed trades.

10.10.6 Automation

```
pm-mm-bot --symbol AAPL
pm-ai-trader --id AI001 --preset scalper --symbols AAPL,MSFT
pm-ai-swarm --count 5 --duration 60
```

Automated participants still use gateway IDs and must be allowed by the configuration. The AI traders log on through `pm-alf-gwy`, which must be running; the `value` and `news` presets also need `pm-market-sim` (see Running the AI-Trader Swarm). For market making, see Market Making and Market-Maker Bot.

10.10.7 External order entry

```
pm-alf-gwy
pm-balf-gwy
```

Use `pm-alf-gwy` for text ALF clients over TCP and `pm-balf-gwy` for binary order-entry clients. Both ultimately send orders into the same engine.

10.10.8 External market data, post-trade and API services

```
pm-md-gwy      # CALF market data, default TCP :5570
pm-ralf-gwy    # RALF post-trade dissemination
pm-dc-gwy      # DC1 TCP relay for engine drop copy
pm-api-gwy     # REST/WebSocket API, default HTTP :8080
pm-index       # optional real-time index calculator
```

Start only the services your session needs. Each exposed gateway is another port to document, monitor and protect.

10.10.9 Centralized operational logs

```
pm-log-srv
pm-log-cli diagnose
pm-log-cli query --limit 20
```

`pm-log-srv` records operational logging, not trading events. Use it alongside `pm-audit`, not instead of it. Automatic logging into `pm-log-srv` is being rolled out process by process; see Centralized Log Server for current support and CLI workflows.

10.10.10 TapeDeck trader information terminal

TapeDeck (`pm-terminal`) is the browser-based read-only market display in `web-apps/terminal-gui/`. It is not a Python `pm-*` console script. It depends on:

- `pm-md-gwy` for live CALF market data
- `pm-api-gwy` with a read-only API key for history and charts
- optionally `pm-log-srv` for bridge logs

The usual way to start it is as part of the container stack, which resolves the read-only API key from the deployed configuration and wires the terminal to the exchange for you:

```
cd deployment/docker && make up-all      # or: ./edumatcher.sh start
```

Then open http://localhost:8090. The same applies to the log console (8091), the Trading GUI (8093), the order book viewer (8094) and the operator dashboard (8095).

To run it against an exchange you started by hand instead, start it from `web-apps/terminal-gui/` with the key and the gateway URL in its environment:

```
export PM_TERMINAL_API_KEY='key-readonly-...'   # the credential with gateway_id: null
export API_GATEWAY_URL='http://127.0.0.1:8081'  # the 'dashboards' instance, not 'desk'
make dev                                        # or: make up, for a container
```

That key is generated per engine configuration and is issued on the `dashboards` API gateway instance. Without it the live book works and the history panels stay empty. See Trader Information Terminal (TapeDeck) for remote display servers and troubleshooting, and The Development Loop for the `make dev-env` helper that prints both values for you.

10.10.11 Order book viewer (`pm-book`)

`pm-book` is `pm-viewer` in a browser: one symbol's full order book, the session statistics and a trade tape, in `web-apps/book-gui/`. Press `s` or `F1` to switch symbol, as in `pm-viewer`. It depends on:

- `pm-api-gwy` with the read-only API key, for **all** of its data — the live book and trades over the `/api/v1/ws/market-data` WebSocket, and the session's earlier trades and the previous close from the history endpoints
- `pm-stats`, which records those history endpoints' data; without it the statistics cover only the trades seen since the book was opened
- optionally `pm-log-srv` for bridge logs

The container stack starts it with the other applications and hands it the key; open http://localhost:8094. Against an exchange you started by hand, run it from `web-apps/book-gui/` with `PM_BOOK_API_KEY` and `API_GATEWAY_URL` set exactly as for TapeDeck above — unlike TapeDeck it shows nothing without the key, so its `make up` refuses to start without one. See Order Book Viewer for the details.

10.10.12 Operator dashboard (`pm-dash`)

`pm-dash` is the browser view of `pm-monitor`: every process's health, trading activity, participants, books and gateways, live and over time, in `web-apps/dash-gui/`. It depends on:

- `pm-monitor`, for its live snapshot (port `5611`) and the history in `metrics.db`, which it reads read-only
- the deployed configuration, for the threshold bands it draws
- optionally `pm-log-srv` for bridge logs

The container stack starts it with the other applications; open http://localhost:8095. See Operator Dashboard for running it against an exchange you started by hand, and for reading it.

10.11 Process groups by scenario

Scenario	Start these processes
Quick trade demo	`pm-engine`, two `pm-alf-console` terminals

Scenario	Start these processes
Recorded classroom session	`pm-engine`, `pm-audit`, `pm-stats`, `pm-clearing`, `pm-scheduler`, participant gateways, `pm-admin`
Market-making exercise	recorded classroom set plus `pm-viewer`, `pm-mm-bot` or `MM01` gateway, possibly `pm-orders`
External client integration	core engine/recorders plus `pm-alf-gwy` or `pm-balf-gwy`, `pm-md-gwy`, `pm-ralf-gwy` as needed
Browser market display	core engine/recorders plus `pm-md-gwy`, `pm-api-gwy`, TapeDeck
Browser order book	core engine/recorders plus `pm-api-gwy` (`dashboards`), `pm-book`
Exchange health at a glance	running system (with `pm-monitor`) plus the operator dashboard `pm-dash`
Operational investigation	running system plus `pm-monitor-cli`, `pm-audit-cli`, `pm-stats-cli`, `pm-clearing-cli`, `pm-log-cli`, `pm-admin-cli`
Everything, with the web applications	the container stack: `./edumatcher.sh start`, or `make up-all` from a checkout

The first five rows are `pm-opctl-cli` profiles or subsets of them: `micro` is the quick demo, `mini` adds external access, `default` is the recorded classroom set plus the index and the second API instance.

10.12 The `tools/launch_all.sh` convenience launcher

`tools/launch_all.sh` is a macOS convenience launcher. It opens each process in its own Terminal window and automatically falls back to `poetry run` when `pm-engine` is not on PATH.

```
./tools/launch_all.sh              # default viewer symbol
./tools/launch_all.sh AAPL         # one viewer
./tools/launch_all.sh AAPL MSFT    # one viewer window per symbol
```

Use it for demos and local rehearsals. For repeated operations, prefer a checked-in script, `tmux` session, container compose file, or service supervisor that explicitly sets `EDUMATCHER_DATA_DIR` and starts only the processes needed for that scenario.

Launcher scope

The launcher starts a traditional local demo stack. It does not start newer external services such as `pm-md-gwy`, `pm-api-gwy`, `pm-ralf-gwy`, `pm-log-srv` or TapeDeck. Start those explicitly when your playbook needs them.

Prefer pm-opctl-cli for anything repeated

The launcher's value is that you can *see* each process in its own window, which is useful while learning. For a stack you start more than once, `pm-opctl-cli start` covers the full process set, is portable beyond macOS, writes proper log files, and stops cleanly.

10.13 Verifying the system is running correctly

10.13.1 Immediate checks after startup

1. Confirm engine sockets are bound

```
lsof -i :5555 -i :5556 -i :5557
```

Expected: one `pm-engine`/Python process listening on all three ports.

2. Confirm the engine sees the intended config

```
pm-config-deploy --show
```

Then compare the printed compiled path with the path named in the engine startup logs. If they differ, your shell or launcher is using a different `EDUMATCHER_DATA_DIR`.

3. Confirm gateways authenticate

```
pm-alf-console --id TRADER01
```

The gateway should connect and show a prompt. A timeout usually means the engine is not reachable or the gateway ID is not configured.

4. Query state through the operator console

```
pm-admin --id OPS01
```

Then run:

```
SYMBOLS
GATEWAYS
SESSION_STATUS
SCHEDULE
```

Check that the symbol list, gateway list and session state match the intended session.

5. Submit a harmless resting order

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
```

Then verify it appears in `pm-viewer --symbol AAPL` or `pm-admin BOOK|SYM=AAPL`.

6. Confirm durable recorders are writing

```
pm-stats-cli daily
pm-clearing-cli pnl
pm-audit-cli events --limit 5
```

If a CLI reports no database or no events, check that the corresponding recorder was started in the same data directory and has observed relevant activity.

7. Confirm external feeds only if used

```
pm-calf-spy --channels TOP,TRADE --symbols AAPL --format human
pm-ralf-spy --role AUDIT --format human
pm-dc-spy --format human
curl -s http://127.0.0.1:8080/api/v1/healthz
```

Use the checks that match the services you actually started.

In a container, the first two checks collapse into one, and the rest are run the same way with a shell inside:

```
make status                    # or ./edumatcher.sh status — the process table
make health                    # exit code 0 when every process is up
make ports                     # what is actually published on the host
make shell                     # then any pm-*-cli command, as above
```

`make status` runs `pm-opctl-cli list` inside the container, so a `not responding` row means the same thing there as on the host.

10.14 Monitoring a running exchange

10.14.1 Operator surfaces

Tool	Best for
`pm-admin`	Live session, symbol, gateway and risk-control commands
`pm-admin-cli`	Scripts, health checks, one-shot operator queries
`pm-viewer --symbol <SYM>`	One order book in detail
`pm-book` (book-gui, port 8094)	The same order book view in a browser, with symbol switching

Tool	Best for
`pm-orders`	Resting orders across gateways
`pm-board`	Multi-symbol terminal board
`pm-ticker`	Scrolling market tape based on stats
TapeDeck	Browser display for live prices, trades, movers, index, auctions and halts
`pm-dash` (dash-gui, port 8095)	Every process's health, trading activity and events in a browser, live and over time
`pm-audit --terminal`	Raw event flow while recording to disk
`pm-log-cli diagnose`	Operational log diagnosis when log server data exists
Log Operator Console (log-gui)	Browsing, searching and acknowledging centralized logs in a browser
`make status` / `./edumatcher.sh status`	The container process table, without opening a shell
`make logs` / `./edumatcher.sh logs`	The container's own entrypoint output — startup and deployment problems
`make proc-logs P=engine`	Follow one process log from `data/emo/`

10.14.2 Common operator queries

Interactive `pm-admin` examples:

```
SESSION_STATUS
SCHEDULE
SYMBOLS
GATEWAYS
BOOK|SYM=AAPL
ORDERS|GW=TRADER01
VOLUME
HALT_SYM|SYM=AAPL
RESUME_SYM|SYM=AAPL
```

Equivalent one-shot CLI examples:

```
pm-admin-cli --id OPS01 session-status
pm-admin-cli --id OPS01 symbols
pm-admin-cli --id OPS01 gateways
pm-admin-cli --id OPS01 book --sym AAPL
pm-admin-cli --id OPS01 orders --gw TRADER01
pm-admin-cli --id OPS01 halt-sym --sym AAPL
pm-admin-cli --id OPS01 resume-sym --sym AAPL
```

10.14.3 Lightweight health check

```
#!/usr/bin/env bash
set -euo pipefail

lsof -i :5555 >/dev/null
lsof -i :5556 >/dev/null
lsof -i :5557 >/dev/null

pm-admin-cli --id OPS01 session-status >/dev/null
pm-admin-cli --id OPS01 symbols >/dev/null

echo "OK: engine sockets and admin queries are healthy"
```

Add scenario-specific checks for `pm-md-gwy`, `pm-api-gwy`, TapeDeck, clearing, stats or logs when those services are required.

With `pm-opctl-cli` managing the stack, most of that script is one command:

```
pm-opctl-cli health -q || echo "one or more processes are down"
```

and against a container, from the host:

```
make -C deployment/docker health
```

10.15 Logging levels

Most long-running `pm-*` processes share the same logging flags.

Flag	Effect
(none)	`WARNING` and above
`-v`, `--verbose`	`INFO`: startup, config, lifecycle and connection messages
`-vv`	`DEBUG`: detailed message flow, useful during local debugging
`--log-level LEVEL`	Explicit level such as `ERROR`, `INFO` or `DEBUG`
`-q`, `--quiet`	Explicit warning-level output

Use verbose logging for rehearsals and incident response. For long unattended runs, combine normal process output with `pm-audit`, `pm-stats`, `pm-clearing` and, where configured, `pm-log-srv`.

10.16 Troubleshooting startup problems

10.16.1 Engine exits immediately

Symptom	Likely cause	Fix
Address already in use on `:5555`, `:5556` or `:5557`	Another engine is still running	`lsof -i :5555`, stop the old process, then restart
Config digest or source warning	Authored YAML changed after deploy, or deployed artifact was modified	Run `pm-config-deploy engine_config.yaml`, then restart
Unknown artifact schema	Artifact was compiled by an incompatible version	Re-run `pm-config-deploy` with the current package
Invalid config	Validation or loader rejected the authored file before deployment, or the deployed artifact is stale	Run `pm-config-deploy --check engine_config.yaml` and fix reported errors

10.16.2 Gateway authentication timeout

Most common causes:

1. `pm-engine` is not running or has not finished binding sockets.
2. The gateway is using a different `EDUMATCHER_DATA_DIR` from the engine.
3. The gateway ID is not in `participants` in the deployed configuration.
4. Local firewall, VPN or container networking prevents access to port `5555`.

Checks:

```
lsof -i :5555
pm-config-deploy --show
pm-admin-cli --id OPS01 gateways
```

10.16.3 Viewer or board shows an empty book

An empty book is normal until orders rest in it. Submit a resting order, connect a market maker, seed market-maker quotes in config, or start `pm-mm-bot`.

If the book should already have liquidity, check:

- The symbol exists in the deployed config.
- The market-maker gateway connected and authenticated.
- The market-maker quote seed references a `MARKET_MAKER` gateway.
- The session phase allows the expected behavior.

10.16.4 Stats, ticker or board show no activity

`pm-stats` must be running before trades occur if you want complete statistics. `pm-ticker` and much of `pm-board` depend on `stats.db`.

Checks:

```
pm-stats-cli daily
pm-stats-cli trades --limit 5
```

If there are no rows, verify that `pm-stats` is running in the same data directory and that at least one trade has occurred.

10.16.5 Clearing and stats do not reconcile

Since both `pm-stats` and `pm-clearing` read `locale.session_timezone` from the same deployed config, they cannot disagree unless the config is changed and redeployed between restarts. If they do disagree, stop both, redeploy a consistent config, and restart.

10.16.6 Scheduler transitions do not happen

Check these in order:

1. Is `pm-scheduler` running?
2. Is `sessions_enabled: true` in the deployed config?
3. Does the schedule use the `locale.session_timezone` and `locale.country` you expect?
4. Is today a trading day under that country calendar?
5. Did the scheduler start before the transition time passed?

For a rehearsal independent of wall-clock time:

```
pm-scheduler --now --delay 5
```

10.16.7 Admin command rejected

Read-only and gateway-scoped commands can be issued by configured gateways, but exchange-wide halt/resume and symbol-wide mass cancel commands require `role: ADMIN`.

Fix the gateway role in `engine_config.yaml`, deploy it, and restart affected processes:

```
participants:
  - id: OPS01
    role: ADMIN
    description: Operator console
```

10.16.8 External feed is reachable but silent

For CALF/RALF/DC/API issues, separate connectivity from data availability:

- Connectivity: can the client reach the TCP or HTTP port?
- Session: did the client complete `HELLO` or authentication?
- Subscription: did the client subscribe to a channel and symbol that exists?
- Source events: has the engine actually published the kind of event expected?

Useful probes:

```
pm-calf-spy --channels TOP,TRADE --symbols AAPL
pm-ralf-spy --role AUDIT
pm-dc-spy
curl -s http://127.0.0.1:8080/api/v1/healthz
curl -s -H "Authorization: Bearer <api-key>" http://127.0.0.1:8080/api/v1/status
```

10.16.9 TapeDeck loads but shows offline or missing history

If the browser loads, the TapeDeck bridge is running. Then check upstreams:

Symptom	Likely cause
`RECONNECTING` or `OFFLINE` live state	bridge cannot reach `pm-md-gwy` on CALF port `5570`
Live prices tick but charts are empty	bridge cannot reach `pm-api-gwy`, the API key is missing, or `pm-stats` has no history
Index view is empty	no `pm-index` process or no index configured
Logs absent	`pm-log-srv` disabled or unreachable; TapeDeck may be using local fallback logs

See Trader Information Terminal for the full TapeDeck runbook.

10.16.10 Container-specific problems

Symptom	Likely cause	Fix
A GUI shows data from an exchange you did not start	Another install's containers own these names and ports	`./edumatcher.sh mounts` or `make mounts` names the host directory behind each container path. Stop the other stack first
The stack starts but the terminal's history panels are empty	The read-only API key was not resolved — the configuration has no credential with `gateway_id: null`	Add one under `api_gateways` and redeploy. The live feed does not need it; history does
`make status` is all green but a sibling container cannot connect	A listener was narrowed to loopback inside the container	Check `EDUMATCHER_GATEWAY_BIND_HOST` and any `bind_address:` in your configuration — see Installation
A source change has no effect	The image was not rebuilt	`make build` for the backend, `make build-guis` for a web app
No entries in the Log Operator Console's live tab	`pm-log-srv`'s ZeroMQ sockets are not published	Only matters when running the app outside the stack; inside it, nothing needs publishing

A container that will not start at all is usually explained by its own output: `./edumatcher.sh logs` or `make logs` shows the entrypoint's configuration deployment, which is where a bad `EM_CONFIG` or an invalid supplied configuration surfaces.

10.17 Restart and shutdown

10.17.1 Restarting non-engine processes

Most non-engine processes can be stopped and restarted independently. They reconnect to the engine on startup. The main caveat is data completeness:

- A restarted viewer can recover its current view from snapshots.
- A restarted external gateway can accept new clients.
- A stopped `pm-audit`, `pm-stats` or `pm-clearing` misses events while offline.
- A stopped `pm-md-gwy` may cause external clients to reconnect and replay only within the configured replay window.

10.17.2 Restarting the engine

Restarting `pm-engine` disconnects every gateway and invalidates live subscriber state. Use it for planned maintenance, not casual operator cleanup.

Before restarting:

1. Halt or close the session if appropriate.
2. Tell participants to stop sending orders.
3. Stop external gateways if clients should not reconnect during the restart.
4. Press `Ctrl-C` in the engine terminal or send `SIGINT`.
5. Wait for clean shutdown messages.
6. Start the engine, then restart or verify dependent processes.

On clean shutdown, the engine persists resting order state — both GTC and DAY orders — to disk and publishes end-of-day style events. A DAY order is not discarded at shutdown; it is only dropped on the *next* startup if the business day has since moved on (a shutdown mid-session is not itself a day boundary).

10.17.3 Full clean shutdown

Recommended order:

1. Stop order entry: participant gateways, ALF/BALF gateways, bots.
2. Stop external feeds and displays: CALF/RALF/DC/API, TapeDeck, viewers.
3. Stop scheduler.
4. Stop the engine with `Ctrl-C` or `pkill -INT -f pm-engine`.
5. Stop recorders after the engine has emitted final events.
6. Archive the data directory if this was a class, demo or test run.

Example archive:

```
tar -czf edumatcher-session-$(date +%Y%m%d-%H%M%S).tgz \
    -C "$EDUMATCHER_DATA_DIR" .
```

10.17.4 Shutting down a managed stack

`pm-opctl-cli stop` sends `SIGTERM` to everything it started, in alphabetical order by process name — which is *not* the startup order above. For a session whose records matter, stop order entry first (close the participant terminals, or halt the session with `pm-admin`), then let it stop the rest.

For a container:

```
./edumatcher.sh stop        # or: make down-all
```

The container stops the processes and exits; the data directory is on your disk and survives untouched, so `start` resumes the same exchange. `make clean-data` — and only that — throws the exchange state away.

10.18 Appendix: operator playbooks

These playbooks are intentionally explicit. Copy them into a session-specific runbook and replace IDs, symbols, timezones and ports with your own.

10.18.1 Playbook 1 - five-minute local smoke test

Goal: prove that a local install can start, accept orders and match one trade.

1. Prepare:

   ```
   mkdir -p ~/edumatcher-smoke
   cd ~/edumatcher-smoke
   pm-setup
   pm-config-deploy --show
   ```

2. Start engine:

```
pm-engine --verbose
```

3. Start two gateways:

```
pm-alf-console --id TRADER01
pm-alf-console --id TRADER02
```

4. In `TRADER02`, place a sell:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
```

5. In `TRADER01`, place a crossing buy:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
```

6. Success criteria: both gateways show fills.

10.18.2 Playbook 2 - recorded classroom session

Goal: run a session with complete audit, stats and clearing records.

1. Set data directory and deploy config:

```
export EDUMATCHER_DATA_DIR="$HOME/edumatcher-sessions/class-01"
mkdir -p "$EDUMATCHER_DATA_DIR"
pm-config-deploy --check engine_config.yaml
pm-config-deploy engine_config.yaml
```

2. Start core services:

```
pm-engine --verbose
pm-audit --terminal
pm-stats
pm-clearing
```

3. Start session control and participants:

```
pm-scheduler
pm-admin --id OPS01
pm-alf-console --id TRADER01
pm-alf-console --id TRADER02
pm-alf-console --id MM01
```

4. Start displays:

```
pm-viewer --symbol AAPL
pm-orders
pm-board
pm-ticker --db-interval 15
```

5. During the session, check:

```
pm-admin-cli --id OPS01 session-status
pm-admin-cli --id OPS01 gateways
pm-stats-cli daily
pm-clearing-cli pnl
```

6. After shutdown, archive `$EDUMATCHER_DATA_DIR`.

10.18.3 Playbook 3 - external client integration test

Goal: expose order entry, market data and post-trade feeds to client developers.

1. Start the core recorded stack: engine, audit, stats, clearing.
2. Start only the required external gateways:

```
pm-alf-gwy
pm-balf-gwy
pm-md-gwy
pm-ralf-gwy
pm-dc-gwy
```

3. Verify feeds locally before handing endpoints to clients:

```
pm-calf-spy --channels TOP,TRADE --symbols AAPL
pm-ralf-spy --role AUDIT
pm-dc-spy
```

4. Document exposed ports, expected protocol, gateway IDs, credentials and replay limits for each client team.

10.18.4 Playbook 4 - TapeDeck wallboard

Goal: run the browser terminal for a classroom screen or observer desk.

1. Start core engine and recorders.
2. Start market data and API history services:

```
pm-md-gwy
pm-api-gwy
```

3. Optionally start operational logs:

```
pm-log-srv
```

4. Start TapeDeck:

```
cd web-apps/terminal-gui
export PM_TERMINAL_API_KEY='...'
make up
```

5. Open `http://localhost:8090` and check live state, Overview, Symbol Detail, Tape and Session/Halt views.

10.18.5 Playbook 5 - incident response: bad or unexpected fills

Goal: stop damage, preserve evidence, and determine scope.

1. Stop new matching if needed:

```
pm-admin-cli --id OPS01 halt
# or only one symbol
pm-admin-cli --id OPS01 halt-sym --sym AAPL
```

2. Preserve current state:

```
pm-admin-cli --id OPS01 gateways
pm-admin-cli --id OPS01 orders --gw TRADER01
pm-audit-cli events --limit 50
pm-clearing-cli pnl
```

3. Identify affected gateway IDs and symbols from audit, fills and clearing.
4. Cancel risky outstanding interest if appropriate:

```
pm-admin-cli --id OPS01 cancel-sym --sym AAPL
pm-admin-cli --id OPS01 kill --gw TRADER01
```

5. Resume only after the cause is understood:

```
pm-admin-cli --id OPS01 resume-sym --sym AAPL
pm-admin-cli --id OPS01 resume
```

10.18.6 Playbook 6 - config change between sessions

Goal: change symbols, gateways, risk controls or schedules without split-brain configuration.

1. Stop trading and shut down affected processes.

2. Edit `engine_config.yaml`.
3. Validate and deploy:

```
pm-config-deploy --check engine_config.yaml
pm-config-deploy engine_config.yaml
```

4. Restart the engine and dependent processes.
5. Verify with `SYMBOLS`, `GATEWAYS`, `SCHEDULE` and one test order.

10.19 See also

In this book:

- The Configuration Workflow - authored YAML and deployed artifact
- Auctions & Scheduling - session phases and trading date
- Risk Controls - collars, circuit breakers and halts
- Persistence - all files written by the exchange
- Installation - the container stack, its networking and every directory it uses

Quick Start Guide:

- What EduMatcher Is, Install and Start, Your First Trade - first concepts and a first trade
- Run a Trading Session - phases, auctions and the records, by hand

Participant Guide:

- The Trader Console (pm-alf-console) - participant command syntax
- Trader Information Terminal - TapeDeck browser display
- Order Book Viewer - `pm-book`, the browser companion to `pm-viewer`

Reference Manual:

- Processes - full process inventory and message flow

Architecture and Developer Guide:

- The Development Loop - running a web application against a backend you started yourself

CHAPTER 11

The Admin Console and Exchange Commands

Learning objectives

After reading this page you will understand:

- What Exchange Commands are and how they differ from raw ZeroMQ messages
- How to build an operator client that wraps the message layer
- The full catalogue of commands available to an `ADMIN`-role gateway
- Which commands are ADMIN-restricted and which any connected gateway may send
- How to extend the framework with new commands as the exchange evolves
- How to actively manage an exchange

Prerequisites: Messages for the raw two-frame format. Risk Controls for the halt/resume operational flow.

11.1 Concept

Every engine interaction is ultimately a two-frame ZeroMQ multipart message:

```
frame[0]  b"<topic>"
frame[1]  b'{"field": "value", ...}'
```

Exchange Commands are a thin Python wrapper layer that turns those frames into named method calls. They exist so that operators can control the exchange without writing raw socket code, and so that the growing set of operator interactions has a single, consistent home that can be extended with new commands as the system evolves.

```
Operator script
    └─ ExchangeCommandClient.halt_all()
            └─ PUSH  b"risk.circuit_breaker_halt_all"  {"gateway_id": "GW_ADMIN"}
            └─ SUB   b"risk.circuit_breaker_halt_all_ack.GW_ADMIN"  ← ack
```

The client holds a PUSH socket (port **5555**) for commands and a SUB socket (port **5556**) for receiving acks and events.

11.1.1 How to send ADMIN commands

There are three ways to issue exchange commands, suited to different workflows:

Tool	What it is	Best for
`pm-admin`	Interactive REPL with tab completion and history	Human operator — incident response, market supervision
`pm-admin-cli`	Single-shot CLI command, exits with code 0/1	Scripting, CI/CD, automation pipelines
`ExchangeCommandClient`	Python API class	Custom operator tooling, integration tests

All three share the **same command execution logic** (`execute_command` in `src/edumatcher/commands/console.py`), so adding a new exchange command only requires changes in one place.

11.2 ADMIN console (`pm-admin`)

For operators who want to control the exchange **without writing Python code**, EduMatcher ships a dedicated interactive console:

```
poetry run pm-admin --id GW_ADMIN
```

The gateway ID must match an entry in `engine_config.yaml` with `role: ADMIN`.

11.2.1 What you see at startup

```
ADMIN console — GW_ADMIN connected  Exchange operator (read/write)
Type HELP for commands.  Tab=complete  ↑↓=history  Ctrl-C=exit

[GW_ADMIN|ADMIN]>
```

The prompt is red to distinguish it from the regular `pm-alf-console` prompt. Tab completion and arrow-key history work exactly as in `pm-alf-console`.

11.2.2 Console command syntax

Commands follow the same `CMD|KEY=VALUE|KEY=VALUE` pipe syntax as the ALF gateway command language (see ALF Protocol Reference). Single-word commands (no arguments) need no pipes.

Command	Syntax	Notes
`HALT`	`HALT`	Exchange-wide circuit-breaker halt — requires ADMIN
`RESUME`	`RESUME`	Lift the exchange-wide halt — requires ADMIN
`HALT_SYM`	`HALT_SYM|SYM=AAPL`	Halt trading on a single symbol — requires ADMIN
`RESUME_SYM`	`RESUME_SYM|SYM=AAPL`	Resume a single halted symbol — requires ADMIN
`CANCEL_SYM`	`CANCEL_SYM|SYM=AAPL`	Cancel **all** resting orders for a symbol across every gateway — requires ADMIN
`REOPEN`	`REOPEN|SYM=AAPL|DRY_RUN=1` or `REOPEN|SYM=AAPL|PRICE=100.00|NOTE=manual open`	Force a halted symbol to uncross and clear the halt in one step — requires ADMIN
`KILL`	`KILL|GW=TRADER01` or `KILL|GW=TRADER01|SYM=AAPL`	Cancel all orders/quotes for a gateway (optional: scope to one symbol)
`KICK`	`KICK|GW=TRADER01` or `KICK|GW=TRADER01|REASON=Compliance hold`	Forcefully disconnect a gateway
`QCANCEL`	`QCANCEL|GW=MM01|SYM=AAPL`	Cancel an MM's active quote on one symbol
`BOOK`	`BOOK|SYM=AAPL`	Print L1/L2 order-book snapshot
`ORDERS`	`ORDERS|GW=TRADER01`	List resting orders for a gateway
`LEVEL`	`LEVEL|SYM=AAPL` or `LEVEL|SYM=AAPL|PRICE=189.50`	Show every resting order making up a symbol, or one price level, across every gateway — requires ADMIN
`SYMBOLS`	`SYMBOLS`	List all instruments configured in the engine

Command	Syntax	Notes
POSITION	POSITION|GW=MM_AAPL_01 or POSITION|GW=MM_AAPL_01|SYM=AAPL,MSFT	Net position, average cost, and live bid/ask/spread/mid per symbol for any gateway (optional: narrow to one or more symbols)
SESSION	SESSION|STATE=CONTINUOUS	Advance session phase
SESSION_STATUS	SESSION_STATUS	Show the current session state (read-only)
SCHEDULE	SCHEDULE	Show the automatic session-transition schedule
GATEWAYS	GATEWAYS	List all configured gateways and connection status
VOLUME	VOLUME	Show daily traded volume per symbol and in total
HELP	HELP	Show the command reference
EXIT / QUIT	EXIT	Disconnect and exit

11.2.3 Example session

```
[GW_ADMIN|ADMIN]> SYMBOLS
┌──────────────────────────┐
│   Configured instruments │
├────┬─────────────────────┤
│ #  │ Symbol              │
├────┼─────────────────────┤
│ 1  │ AAPL                │
│ 2  │ MSFT                │
│ 3  │ TSLA                │
└────┴─────────────────────┘

[GW_ADMIN|ADMIN]> BOOK|SYM=AAPL
┌──────────────────────────────────────────────┐
│              Order Book — AAPL               │
├──────────────┬──────────────┬────────────────┤
│   Bid Qty    │  Bid Price   │  Ask Price     │       Ask Qty  │
├──────────────┼──────────────┼────────────────┤
│          300 │       149.50 │       150.00   │            100 │
│          200 │       149.25 │       150.25   │            250 │
└──────────────┴──────────────┴────────────────┘
  Last trade: 149.75 × 200

[GW_ADMIN|ADMIN]> HALT
HALTED  3 symbol(s), 6 quote leg(s) cancelled

[GW_ADMIN|ADMIN]> KILL|GW=TRADER01
KILL OK  TRADER01  orders=4  quotes=0

[GW_ADMIN|ADMIN]> ORDERS|GW=TRADER01
No resting orders for TRADER01

[GW_ADMIN|ADMIN]> LEVEL|SYM=AAPL|PRICE=149.50
┌──────────────────────────────────────────────────────────────────┐
│              Price-level composition — AAPL @ 149.50             │
├──────────────┬─────────────┬──────┬──────────────┬──────────┬─────┤
│ ID           │ Gateway     │ Side │ Type         │ Remaining │ Price │
├──────────────┼─────────────┼──────┼──────────────┼──────────┼─────┤
│ 66d5940c-f1a7│ MM01        │ BUY  │ LIMIT        │       200 │149.50 │
│ 9a2b1c04-88de│ TRADER01    │ BUY  │ LIMIT        │       100 │149.50 │
└──────────────┴─────────────┴──────┴──────────────┴──────────┴─────┘

[GW_ADMIN|ADMIN]> RESUME
RESUMED  3 symbol(s)

[GW_ADMIN|ADMIN]> SESSION_STATUS
  Session state     : CONTINUOUS
  Auto-scheduling   : ON

[GW_ADMIN|ADMIN]> VOLUME
┌──────────┬──────────────┬──────────────────┬──────────┐
│ Symbol   │          Qty │            Value │ Trades   │
├──────────┼──────────────┼──────────────────┼──────────┤
│ AAPL     │        5,000 │      750,000.00  │     12   │
│ MSFT     │        3,200 │      576,000.00  │      8   │
├──────────┼──────────────┼──────────────────┼──────────┤
│ TOTAL    │        8,200 │    1,326,000.00  │     20   │
└──────────┴──────────────┴──────────────────┴──────────┘

[GW_ADMIN|ADMIN]> EXIT
ADMIN console GW_ADMIN disconnected.
```

11.2.4 Relationship to the other operator tools

`pm-admin`, `pm-admin-cli`, and `ExchangeCommandClient` all share the same `execute_command()` function. Every command maps 1-to-1:

Console (`pm-admin`)	CLI (`pm-admin-cli`)	Python API
`HALT`	`halt`	`client.halt_all()`
`RESUME`	`resume`	`client.resume_all()`
`HALT_SYM|SYM=X`	`halt-sym --sym X`	`client.symbol_halt("X")`
`RESUME_SYM|SYM=X`	`resume-sym --sym X`	`client.symbol_resume("X")`
`CANCEL_SYM|SYM=X`	`cancel-sym --sym X`	`client.cancel_symbol("X")`
`REOPEN|SYM=X[|PRICE=Y][|DRY_RUN=1][|NOTE=Z]`	`reopen --sym X [--price Y] [--dry-run] [--note Z]`	`client.force_uncross("X", price=Y, dry_run=..., note=Z)`
`KILL|GW=X|SYM=Y`	`kill --gw X --sym Y`	`client.kill_switch_gateway("X", symbol="Y")` (own gateway: `client.kill_switch`)
`KICK|GW=X|REASON=Z`	`kick --gw X --reason Z`	`client.gateway_kick("X", reason="Z")`
`QCANCEL|GW=X|SYM=Y`	`qcancel --gw X --sym Y`	`client.kill_switch_gateway("X", symbol="Y", quotes_only=True)` (own gateway: `client.quote_cancel`)
`BOOK|SYM=X`	`book --sym X`	`client.book_depth("X")`
`ORDERS|GW=X`	`orders --gw X`	`client.order_list("X")`
`LEVEL|SYM=X[|PRICE=Y]`	`level --sym X [--price Y]`	`client.price_level_orders("X", price=Y)`
`SYMBOLS`	`symbols`	`client.symbol_list()`
`POSITION|GW=X[|SYM=Y,...]`	`position --gw X [--sym Y,...]`	`client.position_snapshot("X")` + `client.quote_bootstrap("X")`
`SESSION|STATE=X`	`session --state X`	`client.session_advance("X")`
`SESSION_STATUS`	`session-status`	`client.session_status()`
`SCHEDULE`	`schedule`	`client.session_schedule()`
`GATEWAYS`	`gateways`	`client.gateway_list()`
`VOLUME`	`volume`	`client.volume()`

Use `pm-admin` for interactive human-driven operations, `pm-admin-cli` for scripting and automation, and `ExchangeCommandClient` for custom Python tooling.

11.3 CLI tool (`pm-admin-cli`)

`pm-admin-cli` sends **one command per invocation** and exits. It is designed for shell scripts, CI/CD pipelines, and automation where you want the exchange to be driven non-interactively.

```
poetry run pm-admin-cli --id GW_ADMIN <subcommand> [options]
```

Exit code is **0** on success and **1** on failure (auth refused, engine rejection, or timeout). This makes it safe to use in `set -e` shell scripts.

11.3.1 Global flags

Flag	Default	Description
`--id GW_ID`	*(required)*	ADMIN gateway ID (must match `engine_config.yaml`)
`--push ADDR`	`tcp://127.0.0.1:5555`	Engine PULL socket address
`--sub ADDR`	`tcp://127.0.0.1:5556`	Engine PUB socket address
`--timeout MS`	`3000`	Ack wait timeout in milliseconds
`--format {text,json}`	`text`	Output format — `text` is the human-readable form shown throughout this page; `json` prints the same result as machine-readable JSON to stdout instead

`--format json` works identically for **every** subcommand below — it is a global flag handled once in the shared `execute_command` dispatcher, not a per-command option. This is useful for scripting: pipe the output through `jq` rather than parsing the text tables.

```
pm-admin-cli --id GW_ADMIN --format json volume | jq '.total_qty'
```

11.3.2 Subcommands

`halt` — Exchange-wide halt

```
pm-admin-cli --id GW_ADMIN halt
```

```
HALTED  3 symbol(s), 6 quote leg(s) cancelled
```

Requires `role: ADMIN`. Exit code 0 if accepted, 1 if rejected.

`resume` — Lift the exchange-wide halt

```
pm-admin-cli --id GW_ADMIN resume
```

```
RESUMED  3 symbol(s)
```

Requires `role: ADMIN`.

`halt-sym` — Halt trading on a single symbol

```
pm-admin-cli --id GW_ADMIN halt-sym --sym AAPL
```

```
HALTED  AAPL  0 quote leg(s) cancelled
```

Halts only the specified symbol. All other symbols continue trading. Any active MM quote legs for that symbol are cancelled. The symbol remains halted until `resume-sym` is called. Requires `role: ADMIN`.

Flag	Required	Description
`--sym SYMBOL`	yes	Symbol to halt

`resume-sym` — Resume a single halted symbol

```
pm-admin-cli --id GW_ADMIN resume-sym --sym AAPL
```

```
RESUMED  AAPL
```

Resumes a symbol halted by `halt-sym` **or** by an automatic circuit-breaker trigger. Requires `role: ADMIN`.

Flag	Required	Description
`--sym SYMBOL`	yes	Symbol to resume

`cancel-sym` — Cancel all resting orders for a symbol

```
pm-admin-cli --id GW_ADMIN cancel-sym --sym AAPL
```

```
CANCEL_SYM OK  AAPL  orders=12  quotes=2
```

Cancels **every** resting order and active quote for `AAPL` across all connected gateways. This is an emergency book-clearing command — unlike `kill`, which targets a single gateway, `cancel-sym` clears the entire order book for one symbol regardless of who placed the orders. The symbol remains in its current halt state; no halt or resume is triggered. Requires `role: ADMIN`.

Flag	Required	Description
`--sym SYMBOL`	yes	Symbol whose orders to cancel

`reopen` — Force a single symbol to uncross now

```
# Peek the indicative print WITHOUT changing any state:
pm-admin-cli --id GW_ADMIN reopen --sym AAPL --dry-run
```

```
REOPEN DRY-RUN  AAPL  indicative=100.00 x 500  imbalance=BUY (200)
```

```
# Open at the naturally computed equilibrium (if any):
pm-admin-cli --id GW_ADMIN reopen --sym AAPL
```

```
REOPEN OK  AAPL  printed=100.00 x 500  (indicative was 100.00 x 500)
```

```
# Assert an operator opening price (failed-auction recovery):
pm-admin-cli --id GW_ADMIN reopen --sym AAPL --price 100.00 --note "manual open"
```

```
REOPEN OK  AAPL  printed=100.00 x 500  (indicative was no cross)
```

Force-uncrosses one symbol and **clears any halt in the same step**. This is the hands-on recovery for a failed auction — one that discovered no price and left the symbol closed with its orders untouched (see the recovery recipe below). Whatever crosses at the chosen price executes with normal price-time priority; residual interest stays resting.

- Omit `--price` to open at the naturally computed equilibrium. If the book has no crossing interest, nothing prints (the reply shows `nothing printed`) and you can retry with an asserted `--price`.
- Pass `--price` to assert an operator opening price when a failed auction discovered none. Whatever crosses at that price trades; the rest stays put.
- `--dry-run` peeks the indicative print via a read-only calculation and **mutates nothing** — no halt is cleared and no uncross runs. It is the on-demand way to see what would print for a halted or closed symbol, whose `auction.indicative` feed is otherwise silent. With `--price`, the peek reports the quantity that would cross at that asserted price.

Requires `role: ADMIN`.

Flag	Required	Description
`--sym SYMBOL`	yes	Symbol to reopen
`--price PX`	no	Operator opening price; omit to use the computed equilibrium
`--dry-run`	no	Peek the indicative print without changing any state
`--note TEXT`	no	Reason string recorded on the admin monitor

`kill` — Cancel all orders/quotes for a gateway

```
# All symbols
pm-admin-cli --id GW_ADMIN kill --gw TRADER01

# Scoped to one symbol
pm-admin-cli --id GW_ADMIN kill --gw TRADER01 --sym AAPL
```

```
KILL OK  TRADER01  orders=4  quotes=0
```

Flag	Required	Description
`--gw GW_ID`	yes	Target gateway to cancel for
`--sym SYMBOL`	no	Scope to one symbol (omit for all)

For another gateway, `kill` sends the ADMIN kill switch (`risk.kill_switch_gateway`): the audit trail names you, and the target need not be connected. On your own gateway it is the self-service `risk.kill_switch`.

`kick` — Forcefully disconnect a gateway

```
pm-admin-cli --id GW_ADMIN kick --gw TRADER01
pm-admin-cli --id GW_ADMIN kick --gw TRADER01 --reason "Compliance hold"
```

No ack is published. Exit code is always 0 if the engine is reachable. Verify with `orders --gw TRADER01`.

Flag	Required	Description
`--gw GW_ID`	yes	Target gateway to disconnect
`--reason TEXT`	no	Reason string recorded in the engine log

`qcancel` — Cancel a market-maker's active quote on one symbol

```
pm-admin-cli --id GW_ADMIN qcancel --gw MM01 --sym AAPL
```

```
QCANCEL OK  MM01  AAPL
```

Cancels both bid and ask legs of the active quote. Resting limit orders are unaffected. Use `kill` to also remove those. For another gateway this is the ADMIN kill switch limited to quotes, so the audit trail names you and the market maker need not be connected.

`book` — Print the order-book snapshot for a symbol

```
pm-admin-cli --id GW_ADMIN book --sym AAPL
```

```
┌──────────────────────────────────────────────────┐
│              Order Book — AAPL                   │
├─────────────┬─────────────┬─────────────┬────────┤
│   Bid Qty   │  Bid Price  │  Ask Price  │ Ask Qty │
├─────────────┼─────────────┼─────────────┼────────┤
│         300 │      149.50 │      150.00 │    100 │
│         200 │      149.25 │      150.25 │    250 │
└─────────────┴─────────────┴─────────────┴────────┘
  Last trade: 149.75 × 200
```

`orders` — List resting orders for a gateway

```
pm-admin-cli --id GW_ADMIN orders --gw TRADER01
```

Useful to confirm a `kill` or `kick` took effect:

```
pm-admin-cli --id GW_ADMIN kill --gw TRADER01
pm-admin-cli --id GW_ADMIN orders --gw TRADER01   # should print 'No resting orders'
```

`level` — Show per-order price-level composition

```
# Every resting order for the symbol, across every gateway
pm-admin-cli --id GW_ADMIN level --sym AAPL

# Narrowed to a single price level
pm-admin-cli --id GW_ADMIN level --sym AAPL --price 149.50
```

```
┌──────────────────────────────────────────────────────────────────┐
│              Price-level composition — AAPL @ 149.50             │
├──────────────┬─────────────┬──────┬──────────────┬──────────┬─────┤
│ ID           │ Gateway     │ Side │ Type         │ Remaining │ Price │
├──────────────┼─────────────┼──────┼──────────────┼──────────┼─────┤
│ 66d5940c-f1a7│ MM01        │ BUY  │ LIMIT        │       200 │149.50 │
│ 9a2b1c04-88de│ TRADER01    │ BUY  │ LIMIT        │       100 │149.50 │
└──────────────┴─────────────┴──────┴──────────────┴──────────┴─────┘
```

Unlike `book`, which only ever reports the aggregate `{price, qty, count}` per level, `level` lists every order that makes up the level — its id, owning gateway, side, and remaining quantity — ordered by price and then by arrival sequence within a price level, so time priority is visible directly. Requires `role: ADMIN`; any other caller gets a `REJECTED` line instead of a table.

Flag	Required	Description
`--sym SYMBOL`	yes	Symbol to inspect
`--price PRICE`	no	Narrow to orders resting at exactly this price (omit for the whole symbol)

`symbols` — List all configured instruments

```
pm-admin-cli --id GW_ADMIN symbols
```

```
    Configured
    instruments
┌──────┬──────────┐
│ #    │ Symbol   │
├──────┼──────────┤
│ 1    │ AAPL     │
│ 2    │ MSFT     │
│ 3    │ TSLA     │
└──────┴──────────┘
```

`session` — Request a session-phase transition

```
pm-admin-cli --id GW_ADMIN session --state CONTINUOUS
```

```
SESSION  OPENING_AUCTION → CONTINUOUS
```

Flag	Required	Description
`--state STATE`	yes	Target state (case-insensitive): `PRE_OPEN OPENING_AUCTION CONTINUOUS CLOSING_AUCTION CLOSED`

Invalid transitions are rejected by the engine before anything is published, so no reply of any kind is sent back for a rejected transition — the CLI call times out (`Timeout: ...` on stderr, exit code 1) rather than printing a result to check. A successful transition, by contrast, is confirmed by the `session.state` broadcast.

`session-status` — Show current session state (read-only)

```
pm-admin-cli --id GW_ADMIN session-status
```

```
  Session state     : CONTINUOUS
  Auto-scheduling   : ON
```

Returns the current phase without triggering any transition. Useful for monitoring scripts that need to know the exchange state before taking action.

`schedule` — Show the session-transition schedule

```
pm-admin-cli --id GW_ADMIN schedule
```

```
                                Session schedule
┌──────────────────────────┬────────┬────────┬────────┬────────┬────────┬─────┬─────┬──────────┐
│ Phase                    │    Mon │    Tue │    Wed │    Thu │    Fri │ Sat │ Sun │ Holidays │
├──────────────────────────┼────────┼────────┼────────┼────────┼────────┼─────┼─────┼──────────┤
│ Pre-Open                 │ 09:00  │ 09:00  │ 09:00  │ 09:00  │ 09:00  │   – │   – │        – │
│ Opening Auction Start    │ 09:25  │ 09:25  │ 09:25  │ 09:25  │ 09:25  │   – │   – │        – │
│ Continuous Trading Start │ 09:30  │ 09:30  │ 09:30  │ 09:30  │ 09:30  │   – │   – │        – │
│ Closing Auction Start    │ 16:00  │ 16:00  │ 16:00  │ 16:00  │ 16:00  │   – │   – │        – │
│ Closing Auction End      │ 16:05  │ 16:05  │ 16:05  │ 16:05  │ 16:05  │   – │   – │        – │
└──────────────────────────┴────────┴────────┴────────┴────────┴────────┴─────┴─────┴──────────┘
```

One column per weekday plus **Holidays**, filled from the `schedule` block; a dash means the exchange is `CLOSED` that day (no block resolves for it). The times are in `locale.session_timezone`. If `sessions_enabled` is false in the engine config, a message is printed before the table explaining that automatic scheduling is disabled.

`gateways` — List all configured gateways

```
pm-admin-cli --id GW_ADMIN gateways
```

```
                      Configured gateways
┌──────────────┬──────────────┬──────────────────────┬──────────────┐
│ ID           │ Role         │ Description          │ Connected    │
├──────────────┼──────────────┼──────────────────────┼──────────────┤
│ GW_ADMIN     │ ADMIN        │ Operator console     │    YES       │
│ MM01         │ MARKET_MAKER │ Market maker 1       │     no       │
│ TRADER01     │ TRADER       │ Proprietary desk 1   │     no       │
└──────────────┴──────────────┴──────────────────────┴──────────────┘
```

Shows every gateway entry from `engine_config.yaml` with its role and current connection status. A gateway not listed in the config but that somehow connected will not appear (it would have been rejected during auth).

`volume` — Show daily traded volume

```
pm-admin-cli --id GW_ADMIN volume
```

```
                   Daily traded volume
┌──────────────┬──────────────┬──────────────────┬──────────┐
│ Symbol       │          Qty │            Value │   Trades │
├──────────────┼──────────────┼──────────────────┼──────────┤
│ AAPL         │        5,000 │       750,000.00 │       12 │
│ MSFT         │        3,200 │       576,000.00 │        8 │
│ TSLA         │        1,100 │       275,000.00 │        5 │
├──────────────┼──────────────┼──────────────────┼──────────┤
│ TOTAL        │        9,300 │     1,601,000.00 │       25 │
└──────────────┴──────────────┴──────────────────┴──────────┘
```

Counters reset automatically at the start of each trading day — on the transition into `PRE_OPEN` — as well as when the engine restarts, so a multi-day run (auto-scheduling or manual `SESSION|STATE=PRE_OPEN`) reports each day's volume separately rather than an ever-growing cumulative total.

`position` — Show a gateway's net position and live quote

```
pm-admin-cli --id GW_ADMIN position --gw MM_AAPL_01
```

```
MM_AAPL_01
  AAPL
    Position : +500   Avg cost: 149.7500
    Quote    : bid 149.7000  ask 149.9000   spread 0.2000   mid 149.8000
```

Combines two independent engine queries into one row per symbol:

- **Position** — signed net quantity and volume-weighted average cost, from the engine's own fill ledger (`_gateway_positions/_gate` in `engine/main.py`). Works for **any** gateway, not just a market maker — a `TRADER` or `MM` gateway ID is equally valid.
- **Quote** — the gateway's live resting bid/ask, if it currently has an active two-sided quote on that symbol, plus the derived spread and mid. A gateway with a position but no active quote (e.g. a `TRADER` that bought via limit orders)

shows `(no active quote)` for that line.

Add `--sym` to narrow to one or more comma-separated symbols; a symbol with neither a position nor an active quote is omitted from the result:

```
pm-admin-cli --id GW_ADMIN position --gw MM_AAPL_01 --sym AAPL,MSFT
```

Machine-readable form (`--format json`):

```
pm-admin-cli --id GW_ADMIN --format json position --gw MM_AAPL_01
```

```
[
  {
    "symbol": "AAPL",
    "net_qty": 500,
    "avg_cost": 149.75,
    "bid_price": 149.70,
    "ask_price": 149.90,
    "spread": 0.20,
    "mid": 149.80
  }
]
```

This is the same `system.position_request/system.position_snapshot.{GW}` message pair used by `POS|GW=<gateway_id>` on `pm-alf-console` and `pm-alf-gwy` and by `GET /api/v1/admin/gateways/{gateway_id}/positions` on the REST API — all four surfaces agree on the position figures, since they all read the same engine ledger. The bid/ask/spread/mid addition here is specific to `pm-admin-cli`'s `position` command.

11.3.3 Shell scripting example

```
#!/bin/bash
set -e
ID="--id GW_ADMIN"

# Halt the entire exchange while investigating
pm-admin-cli $ID halt

# Cancel all exposure and disconnect the offending participant
pm-admin-cli $ID kill --gw ROGUE01
pm-admin-cli $ID kick --gw ROGUE01 --reason "Automated risk breach"

# Confirm no orders remain
pm-admin-cli $ID orders --gw ROGUE01

# Resume when clear
pm-admin-cli $ID resume
```

```
#!/bin/bash
# Targeted: suspend one symbol and clear its book, keep everything else trading
set -e
ID="--id GW_ADMIN"

# Halt only AAPL — MSFT and TSLA keep trading
pm-admin-cli $ID halt-sym --sym AAPL

# Clear all resting orders on AAPL across every participant
pm-admin-cli $ID cancel-sym --sym AAPL

# ... investigate ...

# Reopen AAPL when satisfied
pm-admin-cli $ID resume-sym --sym AAPL
```

```
#!/bin/bash
# Recover a failed auction: an opening/closing uncross that found no price
# leaves the symbol closed with its orders untouched. Reopen it by hand.
set -e
ID="--id GW_ADMIN"

# 1. Peek what would print — the auction.indicative feed is silent for a
#    halted/closed symbol, so this on-demand read is the only view.
pm-admin-cli $ID reopen --sym AAPL --dry-run

# 2a. If an equilibrium exists, open at it:
pm-admin-cli $ID reopen --sym AAPL

# 2b. If it would not cross, assert an operator opening price instead:
pm-admin-cli $ID reopen --sym AAPL --price 100.00 --note "manual open"

# (Or abandon the book entirely and let it re-collect next phase:)
# pm-admin-cli $ID cancel-sym --sym AAPL
```

11.4 `pm-index-cli` — Index Structural/Audit History Query Tool

`pm-index-cli` is a read-only command-line tool for querying the structural/ corporate-action audit JSONL files written by `pm-index`. It reads files directly from disk — no running `pm-index` process is required. The one exception is `health`, which pings the running `pm-index` to confirm it is alive.

Not for level or EOD history

`pm-index-cli` only exposes structural/audit records (`INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST`). It has no `level` or `eod` subcommand. For index level ticks or daily OHLC history, use `pm-stats-cli index-snapshots` / `index-daily` instead, which reads from pm-stats' SQLite database. See Statistics and Reporting.

```
pm-index-cli [--config PATH] [--data-dir DIR] [--format table|json|csv] [--no-header] COMMAND [options]
```

11.4.1 Global options

Option	Default	Description
`--config PATH` / `-c PATH`	unset	Path to `engine_config.yaml`; used to discover history file paths and index IDs automatically. Without it, the deployed compiled configuration is used instead
`--data-dir DIR`	`data/indexes`	Fallback directory for history JSONL files, used only once an index ID is already known and its `history_file` isn't in the config. It does **not** help discover *which* indices exist
`--format table|json|csv`	`table`	Output format
`--no-header`	off	Suppress header row (CSV only)

11.4.2 Subcommands

Subcommand	Purpose
`events`	Structural events: `INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST`, `REBALANCE`
`indices`	List configured indices from `engine_config.yaml`
`health`	Check the `pm-index` process and, with `--extended`, every index's state and history files

The `events` subcommand accepts:

Option	Description
`--index ID` / `-i ID`	Index ID to query (repeatable); defaults to all indices in `--config`, or in the deployed compiled configuration if `--config` is omitted. If neither is available, `events` exits with an error (code 1) rather than falling back to `--data-dir`
`--days N`	Return records from the last N days (mutually exclusive with `--from`)
`--from DATE_OR_TS`	Start of time range: `YYYY-MM-DD` or ISO-8601 (mutually exclusive with `--days`)
`--to DATE_OR_TS`	End of time range: `YYYY-MM-DD` or ISO-8601 (default: now)
`--limit N`	Maximum rows per index
`--type TYPE` / `-t TYPE`	Filter to one event type (repeatable): `INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST`, `REBALANCE`

11.4.3 Output columns

`events` subcommand:

Column	Description
`ts`	UTC timestamp
`index_id`	Index identifier
`type`	Event type: `INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST`, or `REBALANCE`
`symbol`	Affected symbol (empty for `INIT` and `REBALANCE` — a rebalance's affected symbol list isn't surfaced by this column in any `--format`)
`detail`	Human-readable summary (e.g. `SPLIT 2:1`, `ref_price=195.0`)
`old_divisor`	Divisor before adjustment (empty for `INIT`)
`new_divisor`	Divisor after adjustment (empty for `INIT`)
`level`	Index level after the event

`indices` subcommand:

Column	Description
`id`	Index identifier
`description`	Human-readable label
`history_file`	Path to the structural/audit JSONL file
`state_file`	Path to the divisor/last-price checkpoint file
`constituents`	Comma-separated constituent symbol list

11.4.4 Examples

Corporate actions and constituent changes in the last 30 days:

```
pm-index-cli --config engine_config.yaml events --days 30
```

Only stock-split/dividend/issuance events, all time:

```
pm-index-cli --config engine_config.yaml events --type CORP_ACTION --index EDU100
```

Events as CSV for spreadsheet import:

```
pm-index-cli --config engine_config.yaml --format csv events > index_events.csv
```

Events as JSON for scripting:

```
pm-index-cli --config engine_config.yaml --format json events --index EDU100 \
  | python3 -c "import json,sys; [print(r['ts'], r['type'], r['detail']) for r in json.load(sys.stdin)]"
```

Date-range query on a specific index:

```
pm-index-cli \
  --config engine_config.yaml \
  --format csv \
  events \
  --index EDU100 \
  --from 2026-05-01 \
  --to 2026-06-25 \
  > edu100_events.csv
```

List all configured indices:

```
pm-index-cli --config engine_config.yaml indices
```

```
id     | description              | history_file                          | state_file                          | constituents
-------+--------------------------+---------------------------------------+-------------------------------------+----------------
EDU100 | EduMatcher broad index   | data/indexes/EDU100_history.jsonl     | data/indexes/EDU100_state.json      | AAPL,MSFT,TSLA
TECH2  | Technology pair          | data/indexes/TECH2_history.jsonl      | data/indexes/TECH2_state.json       | AAPL,MSFT
```

Without a config file (direct path):

```
pm-index-cli --format table --data-dir /var/edumatcher/indexes events \
  --index EDU100 \
  --days 30
```

11.4.5 `health` — Process and index file health

Check that `pm-index` is alive and actually processing messages. By default two fast checks run:

- **Process** — a `pm-index` process is running (found with `pgrep`).
- **ZMQ ping** — `pm-index` answers a ping on its health socket (port `5564`) with its current UTC time. The reply comes from the index process's own poll loop, so it proves the loop is running, not just that the process exists.

The file check is opt-in with `--extended`. It loads the configuration the same way `events` does (`--config`, or the deployed compiled configuration), then checks every configured index:

- **State file** — exists, is a JSON object, belongs to that index (its `index_id` matches), and can be opened for writing.
- **History file** — exists, its last record is a well-formed structural record (`INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST` or `REBALANCE` with an integer `ts_ns`), and it can be opened for append. An empty history is fine.

Nothing is ever written to either file, and only the tail of each history file is read, so the check stays fast however large the histories grow; it is still bounded at 15 seconds so a stalled or network-mounted volume cannot hang the command. Every failing file is reported, prefixed with its index ID. A configuration with no indices is healthy, with nothing to check; a configuration that cannot be loaded is a failure.

```
pm-index-cli health
pm-index-cli health --extended
pm-index-cli --config engine_config.yaml health --extended --verbose
pm-index-cli --format json health
pm-index-cli health --quiet
```

```
pm-index process: OK - pm-index is running
index files:      OK - skipped (pass --extended to check state and history files)
index ZMQ ping:   OK - replied at 2026-10-17T14:23:37.797827+00:00
health: OK
```

Options:

Option	Description
`--extended`	Also check every configured index's state and history files (up to 15 s)
`-v`, `--verbose`	List exactly which checks ran; with `--extended`, also print each index's file paths, history size, last record and last level
`-q`, `--quiet`	Print nothing; report only through the exit code

With `--extended --verbose` the report ends with the checks performed and one block per index:

```
Checks performed:
  pm-index process (pgrep):     yes
  ZMQ health ping:               yes
  Index state/history files:     yes

Index files:
  EDU100:
    state:       data/indexes/EDU100_state.json
    history:     data/indexes/EDU100_history.jsonl (3.2KB)
    last record: CORP_ACTION at 2026-07-08T09:31:12.004000+00:00
    last level:  1043.27
    state saved: 2026-07-08T15:59:59.812000+00:00
```

The exit code is `0` when healthy and `1` otherwise. With `--quiet` nothing is printed — not even a configuration error under `--extended` — and the exit code is `0` when healthy or `-1` (255 in a shell) otherwise. `--format json` prints the same report as a JSON object; `csv` prints the same text as `table`.

The health socket binds to `127.0.0.1:5564` by default, on the same `EDUMATCHER_INDEX_BIND_HOST` as `pm-index`'s other two sockets; set `EDUMATCHER_INDEX_HEALTH_PORT` to change the port, and `EDUMATCHER_ENGINE_HOST` to change where `pm-index-cli` connects.

11.4.6 No-row behaviour

Format	Output when no rows match
`table`	`No rows found.`
`json`	`[]`
`csv`	Header row only (unless `--no-header`)

11.4.7 Plotting index level history (Python)

Looking for the write-side tool?

`pm-index-cli` is read-only. To *apply* corporate actions or constituent changes to a running `pm-index` process, use `pm-index-admin-cli` — see the next section.
Level/EOD history is not in `pm-index-cli` — pull it from `pm-stats-cli` instead:

```
import subprocess, json, matplotlib.pyplot as plt

data = json.loads(
    subprocess.check_output([
        "pm-stats-cli", "index-daily",
        "--index-id", "EDU100",
        "--format", "json",
    ])
)

dates  = [r["date"]        for r in data]
closes = [r["close_level"] for r in data]

plt.figure(figsize=(10, 4))
plt.plot(dates, closes, marker="o")
plt.title("EDU100 — daily closing level")
plt.ylabel("Index level")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("edu100_history.png", dpi=150)
```

11.5 `pm-index-admin-cli` — Index Corporate Action / Constituent Change CLI

`pm-index-admin-cli` is a one-shot command-line tool for applying corporate actions (splits, cash dividends, share issuance/buy-backs) and constituent changes (add/delist) to a running `pm-index` process. It talks directly to `pm-index`'s PUSH/SUB sockets (default `tcp://127.0.0.1:5559` / `tcp://127.0.0.1:5558`) — no `pm-engine` connection or `connect()`/auth handshake is involved, since `pm-index`'s socket has no authentication.

```
pm-index-admin-cli --id OPS01 COMMAND [options]
```

11.5.1 Subcommands

Subcommand	Purpose
`split`	Apply a stock split (`--index`, `--sym`, `--ratio N:M`)
`dividend`	Apply a cash dividend (`--index`, `--sym`, `--amount`)
`shares`	Set shares outstanding — issuance or buy-back (`--index`, `--sym`, then `--new-shares` or `--delta`)
`add`	Add a constituent (`--index`, `--sym`, `--shares`, `--price`)
`delist`	Remove a constituent (`--index`, `--sym`)
`history`	Show recent structural/corp-action history for an index

Every mutating subcommand (`split`, `dividend`, `shares`, `add`, `delist`) takes `--index` and `--sym`; `history` takes `--index` but no `--sym` (it has its own `--from`/`--to`/`--types`/`--limit`). The global options below — `--id`, `--format`, `--dry-run`, `--yes`, `--push`, `--sub`, `--timeout` — belong to `pm-index-admin-cli` itself and must be given **before** the subcommand name:

```
pm-index-admin-cli --id OPS01 --yes split --index EDU100 --sym AAPL --ratio 2:1
```

11.5.2 Global options

Flag	Default	Description
`--id`	*(required)*	Ack-routing label — not a credential; see the full chapter
`--push`	`tcp://127.0.0.1:5559`	`pm-index` PULL socket address
`--sub`	`tcp://127.0.0.1:5558`	`pm-index` PUB socket address
`--timeout`	`3000`	Ack timeout in milliseconds
`--dry-run`	off	Validate and print the outbound payload; do not send it
`-y`, `--yes`	off	Skip the confirmation prompt
`--format`	`table`	Output format: `table` or `json`

11.5.3 Exit codes

Code	Meaning
`0`	Accepted by `pm-index` (or `--dry-run` validated cleanly)
`1`	Rejected by `pm-index`, ack timeout, or confirmation prompt declined
`2`	Usage error — bad flags or a client-side validation failure

Under the hood this tool is built on the same `ExchangeCommandClient` class described below — specifically its `index_corp_action()`, `index_delist()`, `index_add_constituent()`, and `index_history()` methods. For the full subcommand reference, worked examples, and the confirmation-prompt/`--dry-run` behaviour, see Index Administration (`pm-index-admin-cli`).

11.6 `ExchangeCommandClient`

`ExchangeCommandClient` lives in `src/edumatcher/commands/client.py` and is importable as:

```
from edumatcher.commands import ExchangeCommandClient, CommandTimeoutError
```

Each command method sends a two-frame ZMQ multipart message over the PUSH socket and blocks on the SUB socket until the matching ack topic arrives or the timeout elapses. On timeout the method raises `CommandTimeoutError`.

`ExchangeCommandClient` is a **context manager** — use `with` to ensure sockets are closed on exit.

11.6.1 Constructor parameters

Parameter	Default	Description
`gw_id`	*(required)*	Gateway ID to authenticate as — must be configured in `engine_config.yaml`
`push_addr`	`tcp://127.0.0.1:5555`	Engine PULL socket address
`pub_addr`	`tcp://127.0.0.1:5556`	Engine PUB socket address
`timeout_ms`	`3000`	Milliseconds to wait for an ack before raising `CommandTimeoutError`

11.6.2 Usage example

```
from edumatcher.commands import ExchangeCommandClient, CommandTimeoutError

with ExchangeCommandClient("GW_ADMIN") as client:
    auth = client.connect()
    if not auth["accepted"]:
        raise RuntimeError(f"Auth failed: {auth['reason']}")

    # ADMIN: halt, inspect, then resume
    try:
        result = client.halt_all()
        print(f"Halted {result['halted_symbols']} symbols, "
              f"cancelled {result['cancelled_quotes']} quote legs")

        orders = client.order_list("TRADER01")
        for o in orders:
            print(f"  {o['id'][:8]}  {o['symbol']}  {o['side']}  qty={o['remaining_qty']}")

        client.resume_all()
    except CommandTimeoutError as e:
        print(f"Engine did not respond: {e}")
# Sockets are closed automatically when the with-block exits.
```

11.7 Command reference

Command	Arguments	Auth required	Underlying message	Ack topic
`connect()`	—	—	`system.gateway_connect`	`system.gateway_auth.{GW}`
`disconnect()`	—	—	`system.gateway_disconnect`	*(none)*
`halt_all()`	—	**ADMIN**	`risk.circuit_breaker_halt_all`	`risk.circuit_breaker_halt_all_ack.{`
`resume_all()`	—	**ADMIN**	`risk.circuit_breaker_resume_all`	`risk.circuit_breaker_resume_all_ack`
`kill_switch(target, symbol?)`	target GW ID, optional symbol	Any connected GW	`risk.kill_switch`	`risk.kill_switch_ack.{target}`

Command	Arguments	Auth required	Underlying message	Ack topic
`quote_cancel(target, symbol)`	target GW ID, symbol	Any connected GW	`quote.cancel`	`quote.cancel_ack.{target}`
`gateway_kick(target, reason?)`	target GW ID, optional reason	Any connected GW	`system.gateway_disconnect`	*(none)*
`book_depth(symbol)`	symbol	Any connected GW	`book.snapshot_request`	`book.{SYMBOL}`
`order_list(target)`	target GW ID	Any connected GW	`order.orders_request`	`order.orders.{target}`
`price_level_orders(sym, price?)`	symbol, optional price	**ADMIN**	`order.price_level_orders_request`	`order.price_level_orders.{GW}`
`symbol_list()`	—	Any connected GW	`system.symbols_request`	`system.symbols.{GW}`
`session_advance(state)`	target state string	Any connected GW	`session.transition`	`session.state`
`session_status()`	—	Any connected GW	`system.session_state_request`	`system.session_status.{GW}`
`session_schedule()`	—	Any connected GW	`system.session_schedule_request`	`system.session_schedule.{GW}`
`gateway_list()`	—	Any connected GW	`system.gateways_request`	`system.gateways.{GW}`
`volume()`	—	Any connected GW	`system.volume_request`	`system.volume.{GW}`
`position_snapshot(target)`	target GW ID	Any connected GW	`system.position_request`	`system.position_snapshot.{target}`
`symbol_halt(symbol)`	symbol	**ADMIN**	`risk.symbol_halt`	`risk.symbol_halt_ack.{GW}`
`symbol_resume(symbol)`	symbol	**ADMIN**	`risk.symbol_resume`	`risk.symbol_resume_ack.{GW}`
`cancel_symbol(symbol)`	symbol	**ADMIN**	`risk.cancel_symbol`	`risk.cancel_symbol_ack.{GW}`
`force_uncross(symbol)`	symbol	**ADMIN**	`risk.force_uncross`	`risk.force_uncross_ack.{GW}`
`kill_switch_gateway(target, note?, symbol?, quotes_only?)`	target GW ID; optional note, symbol, quotes-only	**ADMIN**	`risk.kill_switch_gateway`	`risk.kill_switch_gateway_ack.{GW}`
`kill_switch_global(note?)`	optional note	**ADMIN**	`risk.kill_switch_global`	`risk.kill_switch_global_ack.{GW}`
`reference_reload()`	—	**ADMIN**	`system.reference_reload`	`system.reference_reload_ack.{GW}`
`quote_bootstrap(target, symbol?)`	target GW ID, optional symbol	Any connected GW	`system.quote_bootstrap_request`	`system.quote_bootstrap.{target}`

Command	Arguments	Auth required	Underlying message	Ack topic
`index_history(index, from, to, types?)`	index ID, time range, optional type filter	— (no auth on `pm-index`'s socket)	`index.history_request`	`index.history.{GW}`
`index_corp_action(index, action, symbol, ...)`	index ID, action, symbol, action params	—	`index.corp_action`	`index.corp_action_ack.{GW}`
`index_delist(index, symbol)`	index ID, symbol	—	`index.constituent_change`	`index.constituent_change_ack.{GW}`
`index_add_constituent(index, symbol, shares, price)`	index ID, symbol, shares, price	—	`index.constituent_change`	`index.constituent_change_ack.{GW}`

`mass_cancel(target, symbol)` is a convenience alias for `kill_switch(target, symbol=symbol)` — same message and ack, not a separate command.

The four `index_*` methods talk to `pm-index`'s PUSH/PUB pair (ports 5559/5558), not the engine's — see Index Admin CLI — Why there is no connect() step for why they carry no auth.

PUSH socket has no authentication

The engine does not verify *who* placed a message on the PUSH socket — it only checks the `gateway_id` field inside the payload. ADMIN enforcement (halt/resume) is a role check inside the engine handler, not a transport- level control. This is appropriate for a learning system running on localhost; a production venue would add TLS mutual authentication and signing.

11.8 Command details

11.8.1 `halt_all` — Exchange-wide circuit-breaker halt

```
Frame 0:  b"risk.circuit_breaker_halt_all"
Frame 1:  {"gateway_id": "GW_ADMIN"}
```

```
client = ExchangeCommandClient("GW_ADMIN")
client.connect()

result = client.halt_all()
# result = {
#   "accepted": True,
#   "reason": "",
#   "halted_symbols": 4,
#   "halted_symbol_ids": ["AAPL", "MSFT", "GOOG", "TSLA"],
#   "cancelled_quotes": 12,
#   "cancelled_quote_order_ids": ["ORD-1", "ORD-2", "..."]
# }
print(f"Halted {result['halted_symbols']} symbols, cancelled {result['cancelled_quotes']} quote legs")
```

The engine sets every known symbol to `HALTED` with `halt_source = ADMIN`. No timer is set — the halt is permanent until `resume_all()` is called or the session transitions to `CLOSED`.

While halted:

- MARKET / FOK / IOC orders are rejected with `SYMBOL_HALTED`.
- LIMIT / ICEBERG orders are accepted and rest without matching.
- Quote submission is rejected.

11.8.2 `resume_all` — Lift the exchange-wide halt

```
Frame 0:  b"risk.circuit_breaker_resume_all"
Frame 1:  {"gateway_id": "GW_ADMIN"}
```

```
result = client.resume_all()
# result = {
#   "accepted": True,
#   "reason": "",
#   "resumed_symbols": 4,
#   "resumed_symbol_ids": ["AAPL", "MSFT", "GOOG", "TSLA"]
# }
print(f"Resumed {result['resumed_symbols']} symbols")
```

For each previously halted symbol the engine publishes `circuit_breaker.resume.<SYMBOL>` with `halt_source = "ADMIN"`. Normal order flow and MM quote obligations resume immediately after the ack is received.

11.8.3 `kill_switch` — Cancel all exposure for a gateway

```
Frame 0:  b"risk.kill_switch"
Frame 1:  {"gateway_id": "TRADER01", "symbol": ""}
```

```
# Cancel everything for TRADER01
result = client.kill_switch("TRADER01")
# result = {
#   "accepted": True,
#   "reason": "",
#   "cancelled_orders": 7,
#   "cancelled_order_ids": ["ORD-1", "ORD-2", "..."],
#   "cancelled_quotes": 0,
#   "cancelled_quote_order_ids": []
# }
```

Kill switch does not halt the gateway

Resting orders and quotes are cancelled but the gateway remains connected and can submit fresh orders immediately. There is **no resume message** because nothing is halted. To prevent the gateway from submitting new orders, follow up with `gateway_kick()`.

11.8.4 `mass_cancel` — Cancel exposure for one symbol

```
Frame 0:  b"risk.kill_switch"
Frame 1:  {"gateway_id": "TRADER01", "symbol": "AAPL"}
```

```
result = client.mass_cancel("TRADER01", "AAPL")
# result = {"accepted": True, "cancelled_orders": 3,
#           "cancelled_order_ids": ["ORD-1", "ORD-2", "ORD-3"],
#           "cancelled_quotes": 2,
#           "cancelled_quote_order_ids": ["ORD-4", "ORD-5"]}
```

Identical to `kill_switch` with a symbol argument. Only orders and quotes for `TRADER01` on `AAPL` are affected.

11.8.5 `quote_cancel` — Cancel a market-maker's quote for one symbol

```
Frame 0:  b"quote.cancel"
Frame 1:  {"gateway_id": "MM01", "symbol": "AAPL"}
```

```
result = client.quote_cancel("MM01", "AAPL")
# result = {"accepted": True, "quote_id": "MM01-AAPL-..."}
```

Cancels both the bid and ask legs of the active quote. Resting limit orders submitted outside the quote mechanism are unaffected. Use `mass_cancel` to also remove non-quote limit orders.

11.8.6 `gateway_kick` — Forcefully disconnect a gateway

```
Frame 0:  b"system.gateway_disconnect"
Frame 1:  {"gateway_id": "TRADER01", "reason": "Compliance hold"}
```

```
client.gateway_kick("TRADER01", reason="Compliance hold")
```

The engine applies the gateway's configured `disconnect_behaviour`:

Behaviour	Effect
`LEAVE_ALL`	Session marked disconnected, all orders and quotes left resting
`CANCEL_QUOTES_ONLY`	Quotes cancelled, limit orders left resting
`CANCEL_ALL`	All quotes and orders cancelled

No ack is published. Verify the effect with `order_list("TRADER01")`.

11.8.7 `book_depth` — L1 / L2 order-book snapshot

```
Frame 0:  b"book.snapshot_request"
Frame 1:  {"symbol": "AAPL"}
```

```
book = client.book_depth("AAPL")
# book = {
#   "symbol": "AAPL",
#   "tick_decimals": 2,
#   "ts_ns": 1784500215390000000,
#   "bids": [{"price": 149.50, "qty": 300, "count": 2}, ...],
#   "asks": [{"price": 150.00, "qty": 100, "count": 1}, ...],
#   "last_price": 149.75,
#   "last_qty": 200,
#   "last_buy_price": 149.75,
#   "last_sell_price": 149.50,
#   "recent_trades": [...]
# }

best_bid = book["bids"][0] if book["bids"] else None
best_ask = book["asks"][0] if book["asks"] else None
print(f"AAPL  {best_bid['price']} x {best_bid['qty']}  /  {best_ask['price']} x {best_ask['qty']}")
```

Returns the same payload as the live `book.{SYMBOL}` subscription but on demand rather than waiting for the next change. Useful for a one-shot dashboard or a pre-trade sanity check.

Real-time L1/L2 feed

For a continuous feed, subscribe directly to `book.AAPL` on the PUB socket (port 5556) rather than polling with `book_depth`. The engine publishes a new snapshot after every state-changing event.

11.8.8 `order_list` — Inspect a gateway's resting orders

```
Frame 0:  b"order.orders_request"
Frame 1:  {"gateway_id": "TRADER01"}
```

```
orders = client.order_list("TRADER01")
for o in orders:
    print(f"  {o['id'][:8]}  {o['symbol']}  {o['side']}  {o['order_type']}  "
          f"qty={o['remaining_qty']}/{o['quantity']}  price={o['price']}")
```

Returns all resting (unfilled, non-cancelled) orders across all symbols for the target gateway. Useful for confirming that a `kill_switch` or `mass_cancel` took effect.

11.8.9 `price_level_orders` — Per-order price-level composition

```
Frame 0:  b"order.price_level_orders_request"
Frame 1:  {"gateway_id": "GW_ADMIN", "symbol": "AAPL", "price": 149.50}
```

```
result = client.price_level_orders("AAPL", price=149.50)
# result = {
#   "symbol": "AAPL",
#   "price": 149.50,
#   "rejected": False,
#   "orders": [
#     {"id": "66d5940c-...", "gateway_id": "MM01", "symbol": "AAPL",
#      "side": "BUY", "order_type": "LIMIT", "remaining_qty": 200,
#      "price": 149.50, "arrival_seq": 4021, ...},
#     {"id": "9a2b1c04-...", "gateway_id": "TRADER01", "symbol": "AAPL",
#      "side": "BUY", "order_type": "LIMIT", "remaining_qty": 100,
#      "price": 149.50, "arrival_seq": 4088, ...},
#   ]
# }
for o in result["orders"]:
    print(f"  {o['id'][:8]}  {o['gateway_id']}  {o['side']}  qty={o['remaining_qty']}")
```

`book_depth` aggregates every order at a price into one `{price, qty, count}` row — it cannot say *whose* orders make up that quantity, or in what order they would fill. `price_level_orders` answers exactly that: it returns one record per resting order, across every gateway (not just the caller's own — this is the one query in the table above that is ADMIN-restricted for that reason), sorted by price and then by `arrival_seq` within a price level so time priority reads directly off the list.

Omit `price` to return every resting order for the symbol, across all price levels, in the same price-then-arrival order. A `price` that matches no resting order returns `{"rejected": False, "orders": []}` — an empty result is not the same as a rejection.

For a non-ADMIN caller or an unknown symbol, the engine replies with `rejected: True`, a `reason` string, and an empty `orders` list rather than timing out or raising:

```
result = client.price_level_orders("AAPL")   # called as a non-ADMIN gateway
# result = {"rejected": True, "reason": "...ADMIN...", "orders": []}
```

Where this comes from

This command exists to answer "what actually makes up this price level" when the aggregate `book.{SYMBOL}` view isn't enough to diagnose an anomaly — e.g. distinguishing a market maker's seeded quote leg from an ordinary trader's resting limit order sitting at the same price.

11.8.10 `symbol_list` — Discover configured instruments

```
Frame 0:  b"system.symbols_request"
Frame 1:  {"gateway_id": "GW_ADMIN"}
```

```
symbols = client.symbol_list()
print("Configured symbols:", symbols)
# ["AAPL", "MSFT", "TSLA", "GOOG"]
```

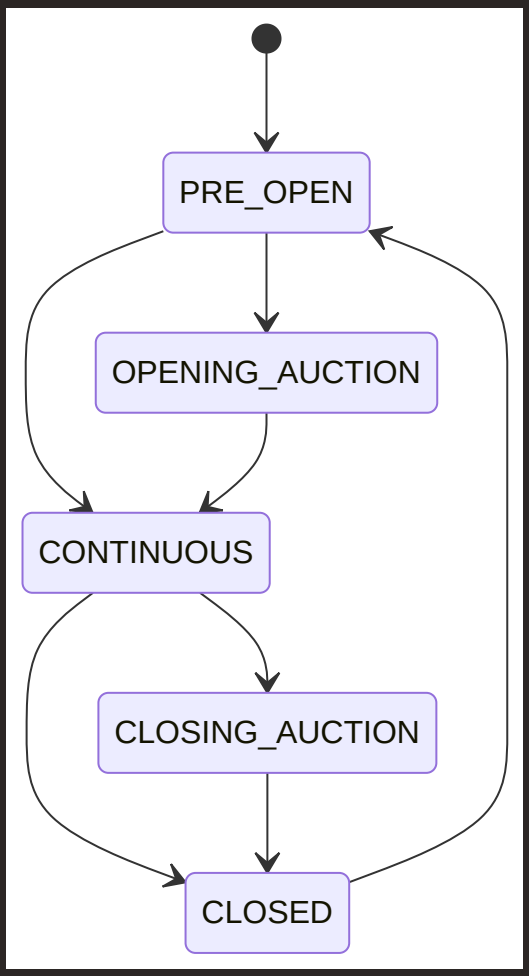
Returns the list of symbols the engine was started with. Useful at startup to drive iteration over all instruments (e.g. request a book snapshot for each).

11.8.11 `session_advance` — Manually drive the trading day

```
Frame 0:  b"session.transition"
Frame 1:  {"to_state": "CONTINUOUS"}
```

```
result = client.session_advance("CONTINUOUS")
# result = {"state": "CONTINUOUS", "prev_state": "OPENING_AUCTION"}
```

Valid state transitions:



Two transitions skip a phase on purpose: `PRE_OPEN → CONTINUOUS` (no opening auction) and `CONTINUOUS → CLOSED` (no closing auction). And `CLOSED` is not terminal — `CLOSED → PRE_OPEN` starts the next trading day, which is how a multi-day auto-scheduled run (and the daily volume-counter reset above) keeps going without restarting the engine.

An **accepted** transition is confirmed by the `session.state` broadcast, which carries the *actual* new state — check `result["state"]` to verify it. An **invalid** transition (one not allowed from the current state) publishes no `session.state` broadcast. The legacy `session_advance()` client does not request an addressed acknowledgement, so it blocks until `--timeout` and then raises `CommandTimeoutError`. By contrast, the REST `POST /api/v1/admin/session/transition` endpoint waits for the engine's addressed rejection and returns `409 TRANSITION_REJECTED` with its reason.

11.9 Full ADMIN operator workflow

The example below shows a complete emergency halt-and-resume sequence using the command client:

```
from edumatcher.commands import ExchangeCommandClient, CommandTimeoutError

client = ExchangeCommandClient("GW_ADMIN")

# 1. Connect
auth = client.connect()
assert auth["accepted"], f"Auth failed: {auth['reason']}"
print("Connected as ADMIN")

# 2. Inspect the market before halting
symbols = client.symbol_list()
for sym in symbols:
    book = client.book_depth(sym)
    bids = book.get("bids", [])
    asks = book.get("asks", [])
    bid_str = f"{bids[0]['price']} x {bids[0]['qty']}" if bids else "—"
    ask_str = f"{asks[0]['price']} x {asks[0]['qty']}" if asks else "—"
    print(f"  {sym:6}  bid={bid_str}  ask={ask_str}")

# 3. Halt all trading (e.g. technology incident detected)
result = client.halt_all()
print(f"HALT: {result['halted_symbols']} symbols halted, "
      f"{result['cancelled_quotes']} quotes cancelled")

# 4. Cancel all exposure for the affected participant
result = client.kill_switch("TRADER01")
print(f"Kill switch TRADER01: {result['cancelled_orders']} orders cancelled")

# 5. Kick the affected gateway
client.gateway_kick("TRADER01", reason="Compliance review")

# 6. Verify TRADER01 has no remaining orders
remaining = client.order_list("TRADER01")
assert not remaining, f"Unexpected orders: {remaining}"

# 7. Resume when the all-clear is given
result = client.resume_all()
print(f"RESUME: {result['resumed_symbols']} symbols resumed")
```

11.10 Extending the framework

To add a new command:

1. **Define the raw message** in `src/edumatcher/models/message.py` (a `make_<command>_msg` helper and an ack helper).
2. **Handle it in the engine** (`src/edumatcher/engine/main.py`) — add a `_handle_<command>` method and wire it into the dispatch `elif` chain.
3. **Add the ack topic prefix** to `_ACK_SUB_PREFIXES` at the top of `src/edumatcher/commands/client.py`.
4. **Add the method** that calls `self._send(frames)` and `self._recv(ack_prefix)`.
5. **Add tests** in `tests/test_commands.py`.
6. **Document it here** with a command-reference row and a detail section.

11.11 See also

In this book:

- Risk Controls — how halt state affects order matching
- Configuration — Role Privileges — the permissions matrix
- Auctions & Scheduling — valid session-state transitions

Participant Guide:

- The Trader Console (`pm-alf-console`) — participant-facing CLI commands (TRADER / MM role)

Protocols and Clients:

- Messages — raw frame format for every message

CHAPTER 12

Running the AI-Trader Swarm

Learning objectives

After reading this page you will be able to:

- Start, size and stop a swarm of AI traders for a class or a load test
- Read its logs and its daily summaries
- Recognise what happens when a worker, `pm-alf-gwy` or the engine goes down, and what (if anything) you have to do
- Run a compressed multi-day soak

Prerequisites: Running the Exchange, ALF Gateway, Session Scheduling.

12.1 What runs

`pm-ai-swarm` is a supervisor. It splits the agents into K contiguous ID blocks and runs each block in one worker process. A worker hosts its agents in one event loop:

- **Orders** go through `pm-alf-gwy`, one ALF session per agent, logged on as the agent's own participant ID (`AI001`, `AI002`, …). The gateway converts prices, heartbeats the engine for the session and applies the participant's `disconnect_behaviour` when the session ends.
- **Market data** comes straight from the engine's PUB socket, one subscription per worker shared by its agents.

Every agent is a *preset* — strategy × execution style × tempo × risk limits — and the swarm's *composition* says how many agents get which preset. `pm-ai-trader` runs one agent on its own.

```
pm-ai-swarm (supervisor)
 ├─ worker w1  AI001–AI167 ─┐ 167 ALF sessions ─┐
 ├─ worker w2  AI168–AI334 ─┤                   ├─> pm-alf-gwy -> pm-engine
 └─ worker w3  AI335–AI500 ─┘ engine PUB (market data) <──────────────┘
```

12.2 Before you start

1. The deployed configuration must list every agent as a participant with role `TRADER`. The `s150-*` examples register `AI001`–`AI020`, `s300-load` registers `AI001`–`AI500`, `s300-sim` `AI001`–`AI100`. The swarm checks this before it starts.
2. `pm-engine`, `pm-scheduler` (if sessions are enabled) and `pm-alf-gwy` are running. `pm-alf-gwy` accepts up to `max_connections` sessions (default 1024).

12.3 Starting it

Each example that has AI participants ships a `swarm.yaml` next to its `engine_config.yaml`:

```
pm-ai-swarm --swarm docs/examples/ref_data/s150-nominal-setup/swarm.yaml
pm-ai-swarm --swarm docs/examples/ref_data/s300-load-setup/swarm.yaml
```

Flags given on the command line win over the file, the file over the built-in defaults, so one file serves several runs:

```
pm-ai-swarm --swarm swarm.yaml --count 100 --budget 200 -v
```

Under `pm-opctl-cli`, the `ai-swarm` profile is `mm-demo` with its market maker anchored to the true values (`pm-mm-bot --anchor-sim 0.3`), plus `pm-market-sim` and `pm-ai-swarm --count 20 --budget 40 --symbols-per-agent 1000`. The `ai-sim` profile runs the `s300-sim` example's swarm (`AI001`–`AI100`) and is set up to run for weeks; it is what the `edumatcher-ai-sim` container image starts — see AI Traders and Market Simulation.

Anchor the market maker

A passive `pm-mm-bot` without `--anchor-sim` refills its quotes around its own mid. Informed AI traders cannot move such a price: in a test run the market drifted 2–4 % from the true values within minutes, against under 1 % with `--anchor-sim 0.3`.

12.3.1 Command-line options

Every `swarm.yaml` key has a flag; a flag wins over the file.

Flag	Default	`swarm.yaml` key	Meaning
`--swarm PATH`	none	—	The `swarm.yaml` to read
`--count N`	`10`	`agents.count`	Number of agents
`--prefix P`	`AI`	`agents.prefix`	Gateway ID prefix
`--start-index N`	`1`	`agents.start`	First ID number
`--presets a:3,b:1`	every built-in, equal weights	`composition`	Preset mix; `a,b` means equal weights
`--symbols S1,S2`	every deployed symbol	`symbols.include`	Universe
`--symbols-per-agent N`	just enough to cover the universe	`symbols.per_agent`	Symbols each agent trades
`--seed-base N`	`1000`	`seed`	Agent i gets `N + i`
`--duration SEC`	`0`	`duration`	Seconds to run; `0` runs until stopped
`--workers N`	`min(cpus − 1, ceil(count / 100))`	`workers`	Worker processes
`--budget N`	`0`	`budget`	Order actions per second for the whole swarm; `0` lets the presets decide
`--alf-host H`	`EDUMATCHER_ENGINE_HOST`, else `127.0.0.1`	`alf.host`	Where `pm-alf-gwy` listens
`--alf-port N`	deployed `alf_gateway.port`	`alf.port`	`pm-alf-gwy`'s port
`-v`, `-q`, `--log-level L`	`WARNING`	`logging.level`	Verbosity; applies to the workers too
`--log-target {server,stdout,file}`, `--log-file PATH`, `--log-failover-timeout SEC`	`server`, —, `30`	`logging.*`	Where operational log records go

12.3.2 `swarm.yaml`

Key	Default	Meaning
`version`	—	Must be `1`

Key	Default	Meaning
`agents.prefix` / `start` / `count`	`AI` / `1` / `10`	Agent IDs `AI001`… (three digits, at most 999)
`workers`	`0`	Worker processes; `0` = `min(cpus - 1, ceil(count / 100))`
`budget`	`0`	Order actions per second for the whole swarm (new orders and cancels); `0` lets the presets' own tempo decide
`seed`	`1000`	Agent i gets `seed + i`: the same file gives the same agents
`duration`	`0`	Seconds to run; `0` = until stopped
`composition`	every built-in preset, equal weights	`preset: weight`. A key is a built-in preset name or a preset YAML file relative to `swarm.yaml`. Weights can sum to anything; the counts are rounded to add up to `count` exactly and interleaved so every worker gets the same mix
`symbols.include`	every deployed symbol	The swarm's universe
`symbols.exclude`	none	Taken out of the universe
`symbols.per_agent`	just enough to cover the universe	Consecutive symbols each agent trades; raised if needed so every symbol is traded
`alf.host` / `alf.port`	`EDUMATCHER_ENGINE_HOST` / deployed `alf_gateway.port`	Where `pm-alf-gwy` listens
`logging.level` / `target` / `file` / `failover_timeout`	as other `pm-*` processes	Logging of supervisor and workers

Unknown keys and bad values are refused before anything starts, with the key named.

12.3.3 Sizing

- **Throughput is bounded by the presets' risk limits**, not only by the budget. A passive agent holds at most `max_live_orders_per_symbol` orders per symbol, so it needs several symbols to stay busy. The `s300-load` file gives every agent 10 symbols (`per_agent: 10`) and reaches its 1 000 actions per second about a minute after the open; with one symbol per agent the same swarm managed about 130.
- **Spread the kinds of trader over every symbol.** With the default, each symbol is traded by about one agent, so a symbol whose only agent is a noise trader never finds its value nor reacts to news. The `s150-*` files give all twenty agents all 150 symbols (`per_agent: 150`); the `s300-load` file's `per_agent: 10` gives each symbol about seventeen.
- The budget controller learns how many actions a decision produces and corrects every 10 seconds, so the first minute runs below budget.
- Measured with `s300-load` at 1 000 actions per second: each of the three workers (167 agents) about 13% of a core, `pm-alf-gwy` about 45%, the engine about 65%.

12.4 Logs and daily summaries

At `INFO` (`-v`) the swarm logs, per worker: one status line a minute, each session change, and one line per trading day:

```
[w1-AI001-AI167] t=120s session=CONTINUOUS actions=34737 (289.4/s) live_orders=2466 fills=22462 rejects=1
[w1-AI001-AI167] day 3 closed: 167 agents, 9120 orders, 5021 fills, 2 rejects, P&L 1532.40
```

Five seconds after each close (so the closing auction's fills are in), every worker appends one JSON line per agent to `<DATA_DIR>/ai_swarm/<date>-<worker>.jsonl`:

```
{"closed_at": "2026-12-08T16:05:05+00:00", "day": 2, "agent": "AI001",
 "preset": "noise-retail", "submitted": 41, "fills": 12, "filled_qty": 97,
 "cancels": 9, "rejected": 0, "killed": 0, "unseen_expiries": 0,
 "gtc_orders": 0, "gross_position": 37, "pnl_realized": -1.52, "pnl_mtm": 0.83}
```


The counters are for that day. `unseen_expiries` counts DAY/ATO/ATC orders the agent dropped at the close without having seen their expiry. P&L is the agent's position ledger since it started (or last re-read its positions after a reconnect), marked at the last trade. `day` counts the closes the worker has seen since it started.

Per-order rejects are logged at `DEBUG` only.

12.5 When something goes down

What fails	What you see	What happens	You do
A worker process	`worker exited (code …); restarting in N s`	Only that worker restarts (back-off 1 s doubling to 60 s); its agents log on again and re-read their positions	Nothing. After 10 restarts in an hour the supervisor gives up on that worker (`giving up on its N agents`) and, when stopped, exits 1
`pm-alf-gwy`	`ALF session lost` per agent, then `ALF session restored`	Agents retry with jittered back-off (at most 10 s apart). The engine keeps a dead gateway's IDs for about 15 s, so they come back 15–25 s after the gateway does	Restart `pm-alf-gwy`
The engine	`SESSION_LOST The exchange engine restarted` per agent	`pm-alf-gwy` closes every session; agents log on again, the worker re-reads the market. The engine restarts flat: positions start from zero, GTC orders are restored	Restart `pm-scheduler` if it happened during the trading day: a running scheduler does not notice the engine is back `CLOSED` and the market stays closed until the next day. A restarted scheduler catches up to the current phase

12.6 Stopping

`SIGTERM` or `Ctrl-C` stops every worker; each closes its ALF sessions, which applies the participants' `disconnect_behaviour` (`CANCEL_ALL` for the example AI participants). Workers get 10 seconds before they are killed.

12.7 Compressed soak

`pm-scheduler --daily --speed F` runs the trading calendar F times faster (see Compressed mode). At `--speed 1200` a trading day takes about a minute of wall time, so 30 trading days — weekends and holidays included — pass in under an hour:

```
pm-scheduler --daily --speed 1200 --start 2026-12-07T08:55 -v
pm-ai-swarm --swarm docs/examples/ref_data/s150-nominal-setup/swarm.yaml -v
```

Only the calendar is compressed. Order ages, heartbeats and decision rates run on the wall clock, so a compressed day is a short day, not a fast one.

CHAPTER 13

Listing a New Symbol (`pm-new-symbol`)

Learning objectives

After reading this page you will understand:

- What an IPO means on EduMatcher, and which part of it the exchange owns
- How `pm-new-symbol` (alias `pm-ipo`) chooses the file it edits, and when it redeploys
- Why the IPO price becomes the book's reference price
- How the opening seed quote is sized and priced against the market maker's obligation
- Every check that makes the command refuse, and what to do about each one
- The IPO-day steps the command deliberately leaves to you

13.1 What an IPO is, and what the exchange does

An initial public offering has two halves. In the **primary market**, the issuer sells new shares to investors at an agreed **offer price**. That happens before listing day, and no exchange order book is involved. In the **secondary market**, the shares start trading on an exchange. The listing is that exchange's part of the IPO:

- a new order book exists for the symbol,
- the offer price becomes the **reference price** that the price collar and the circuit breaker measure moves against (there is no previous close yet), and
- a designated market maker usually posts an opening quote, so the first investor who wants to sell has somebody to sell to.

`pm-new-symbol` does exactly this to an existing configuration. It does not touch any participant's position. EduMatcher has no primary-market allocation, so the shares investors "bought in the offering" only appear once they are traded.

13.2 Quick start

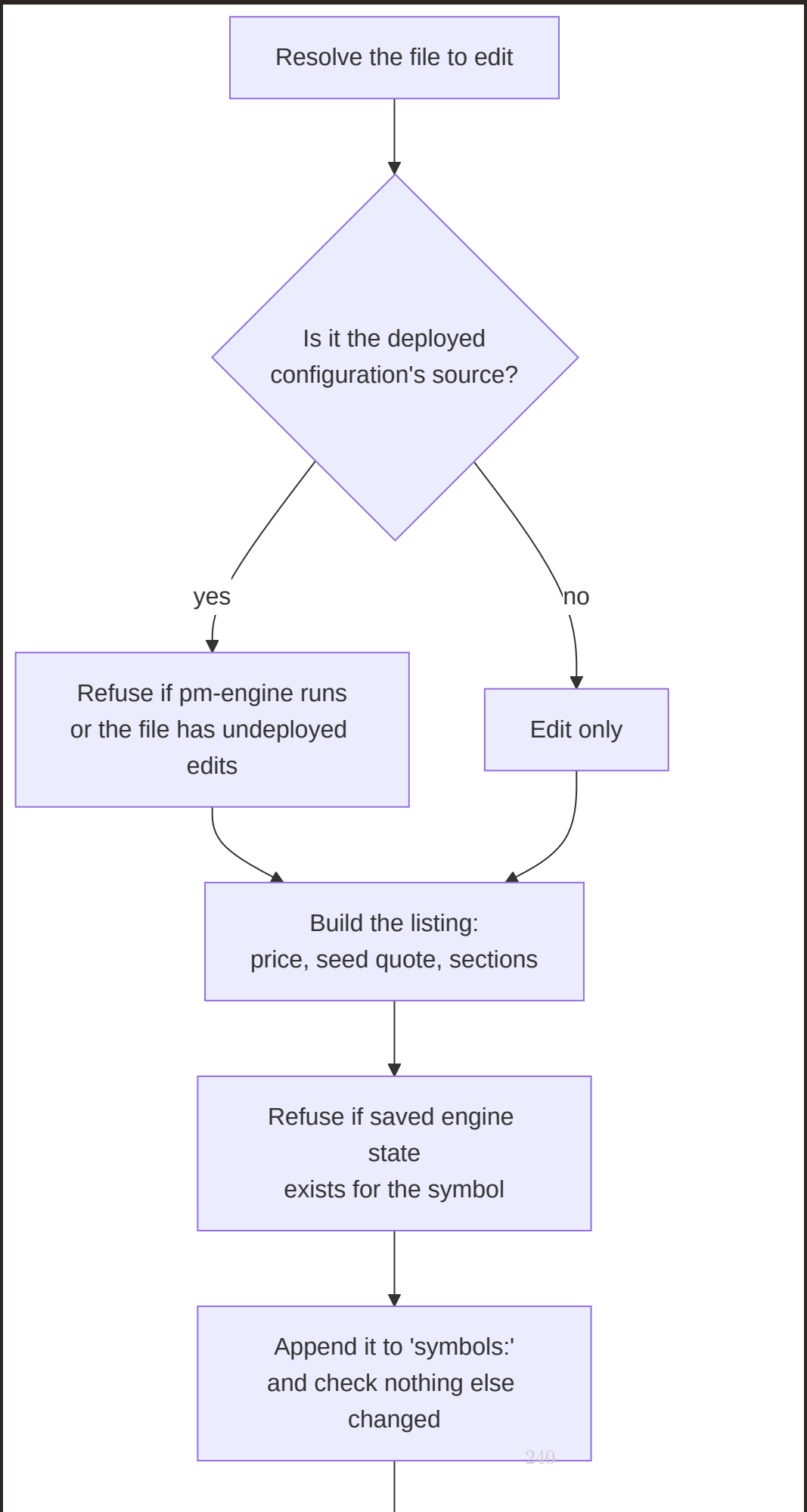
With the exchange stopped:

```
pm-opctl-cli stop
pm-new-symbol --symbol NEWCO --ipo-price 20.00 --outstanding-shares 50000000
pm-opctl-cli start
```

```
Listed NEWCO in /home/me/course/engine_config.yaml: IPO price 20.00, 50000000 shares outstanding
  seed quote MM01: 1000 @ 19.90 / 1000 @ 20.10 (DAY)
Deployed to /home/me/.local/share/edumatcher/ref_data/engine_config.json. Start the exchange to open trading.
```

`pm-ipo` is the same command under another name. Use whichever reads better in your course material.

13.3 What the command does



Nothing is written until the edited file has passed the same four verification layers as `pm-config-deploy` (see Config Verifier). A refusal at any step leaves both the YAML and the deployed artifact exactly as they were.

13.4 Which file is edited

The Configuration Workflow separates the YAML you **author** from the compiled artifact the exchange **runs**. Every artifact records the file it was compiled from in `meta.source_path`. `pm-new-symbol` edits that source, never the artifact.

Situation	What happens
No `--config`	Edits the deployed artifact's source and redeploys it
`--config` names the deployed source	Same as no `--config`
`--config` names any other file	Edits that file only. Run `pm-config-deploy` on it yourself
`--config` names `ref_data/engine_config.yaml`, which was copied from somewhere else	Refused: that file is only a copy, and the next deploy of the real source would drop the symbol
The source is a bundled example (`pm-config-deploy --example …`)	Refused: shipped examples are never edited. Copy it, deploy the copy, then list
Nothing deployed and no `--config`	Refused: there is nothing to list into

If you set up with `pm-setup`, the deployed source *is* `<DATA_DIR>/ref_data/engine_config.yaml`, so the default works without `--config`.

If the source uses `include:`, the symbol is written to the included file that supplies the last listed symbol and the main file is left as it is; the command reports which file it edited.

Undeployed edits block the listing

The deployed source, together with any files it includes, must be byte-for-byte what was last deployed (their combined SHA-256 must match `meta.source_sha256`). Otherwise the IPO deploy would also ship whatever else had been edited, without review. Deploy or revert those edits first.

13.5 The IPO price

`--ipo-price` is written as both `last_buy_price` and `last_sell_price` of the new symbol. When the engine starts, it uses them as:

- the book's last buy and last sell prices,
- the **collar reference**, so orders far from the offer price are rejected when collars are enforced (see Risk Controls), and
- the **circuit-breaker reference**, so the first minutes of trading are protected from the very first order.

The price must be positive and a whole number of ticks: with the default `--tick-decimals 2`, `20.005` is refused. There is no separate bid and ask because nothing has traded yet. The offer price is the only price the market knows.

13.6 The seed quote

13.6.1 When one is written

Configuration	Seed quote
At least one `MARKET_MAKER` gateway and `require_mm_seed_quotes: true` (the default)	Always. The engine would refuse to start without one
No `MARKET_MAKER` gateway, or `require_mm_seed_quotes: false` (the `-nomm` examples)	None, so the book opens empty. `--mm-gateway-id` asks for one anyway

With exactly one market-maker gateway, it posts the seed. With several, you must choose one with `--mm-gateway-id`. The command will not pick one for you.

13.6.2 How it is priced

The quote must meet the market maker's own obligation. The engine resolves that obligation with this precedence, and `pm-new-symbol` resolves it the same way:

1. `participants[…].mm_obligations.<SYMBOL>` for that gateway and symbol
2. the gateway's own `mm_max_spread_ticks` / `mm_min_qty` / `enforce_mm_obligation`
3. `mm_obligation_defaults`, which the gateway fields default to

(A global per-symbol entry under `mm_obligation_defaults.symbols` cannot name a symbol that is not listed yet, so it plays no part here.)

By default the quote is exactly the widest spread allowed, placed around the IPO price:

```
bid = ipo_price - (max_spread_ticks // 2) × tick
ask = bid + max_spread_ticks × tick
```

With `mm_max_spread_ticks: 20` and an IPO price of `20.00`, that gives `19.90 / 20.10`. With an odd width such as 7, the extra tick goes on the ask: `19.97 / 20.04`.

To quote tighter, pass both `--mm-bid-price` and `--mm-ask-price`. Explicit prices must be on the tick grid, must be ordered, and must **straddle the IPO price** (`bid ≤ ipo_price ≤ ask`). When the obligation is enforced, they must also not be wider than the maximum spread. The quantities (`--mm-bid-qty`, `--mm-ask-qty`, default 1000 each) must not be below `mm_min_qty` when the obligation is enforced.

`--mm-tif` is `DAY` (the default) or `GTC`. Auction-only TIFs make no sense for a resting opening quote. `--no-mm-seed-once` makes the engine re-inject the seed on every start, even when a quote from that market maker survived the restart. The default is to inject it only once. See Market Making for how seed quotes behave afterwards.

13.7 Optional sections: `--field`

`--field KEY=YAML_VALUE` sets one of the symbol's optional sections. The value is parsed as YAML, so nested mappings work. Only these keys are accepted:

Key	Example	Purpose
`level`	`--field level=L2`	Risk-control level from `risk_controls.levels`
`collar`	`--field 'collar={static_band_pct: 0.15}'`	Per-symbol collar bands
`order_limits`	`--field 'order_limits={max_order_qty: 10000}'`	Maximum order quantity / value
`circuit_breaker`	`--field 'circuit_breaker={reference_window_ns: 60000000000}'`	Per-symbol breaker overrides

Any other key is refused. The configuration loader ignores keys it does not know, so a misspelt `colar=` would otherwise vanish silently.

13.8 Symbol names

A symbol is 1–8 characters of `A-Z`, `0-9`, `.` and `_`. Lower case is upper-cased (`newco` becomes `NEWCO`). The limit comes from the wire: the BALF execution report carries the symbol as `char[8]` (Message Reference). A longer name would pass configuration validation and then fail at the first fill.

13.9 Why it refused

Every refusal leaves the YAML and the deployed artifact unchanged.

Message (abridged)	Cause	What to do
`something is listening on tcp://…:5556 / pm-engine is running (pid …)`	The exchange is running	`pm-opctl-cli stop`, then run again
`pgrep is unavailable …`	The running check cannot be done	Install `procps`. The command errs on the side of refusing
`has edits that were never deployed`	The deployed source differs from what was deployed	Deploy or revert those edits first
`is only the deployed copy of …`	`--config` points at `ref_data/` while the source lives elsewhere	Pass the real source with `--config`
`is a bundled example`	The exchange runs a shipped example	Copy it, `pm-config-deploy` the copy, then list
`already listed`	The symbol exists (in any letter case)	Choose another symbol
`must be 1-8 characters …`	Invalid symbol name	See *Symbol names*
`not on the … tick grid`	Price has too many decimals	Round it, or raise `--tick-decimals`
`several MARKET_MAKER gateways`	More than one market maker could seed it	Pass `--mm-gateway-id`
`exceeds …'s obligation / below …'s obligation`	The explicit quote breaks the market maker's obligation	Tighten the spread or raise the quantities
`… has a bid of …`	The default spread would push the bid to zero or below (penny stocks)	Give `--mm-bid-price` / `--mm-ask-price`, or more `--tick-decimals`
`holds saved state for …`	`book_stats.json`, `gtc_orders.json` or `gtc_combos.json` still holds data for this symbol name	See below
`would make … invalid`, followed by check codes	The edited file fails validation (for example a bad `--field` value)	Fix what the listed codes describe (see Config Verifier)
`rewrite 'symbols' in block style / cannot be appended …`	The YAML layout cannot be edited safely	Add the symbol by hand

Saved state wins over the listing

When the engine starts, a symbol's entry in `book_stats.json` takes precedence over its configured last prices, and resting orders in `gtc_orders.json` go back into its book. So a symbol name that traded before would open at the old price, with the old orders, not at the offer price. The same happens if you list a symbol, start the exchange, then remove it to correct a mistake before the engine has run again. `pm-new-symbol` refuses in all of these cases. Remove the symbol's entries from the files it names, or clear all engine state with `pm-opctl-cli clear`. Note that the second option discards the resting orders of *every* symbol. See Persistence.

13.10 The IPO-day checklist

The command handles the configuration. These steps are yours:

1. **Before**: agree the offer price, shares outstanding, tick size, and which market maker will support the stock. `pm-valuation` works out the first two for a fictive company (see IPO Valuation). Decide on day-one risk settings. A tighter collar and an `order_limits` cap are common for a new listing.
2. **Stop the exchange**: all of it, not just `pm-engine`. Every process reads the configuration only when it starts. A gateway restarted on its own would know a symbol the engine does not, or the other way round.

3. **List**: run `pm-new-symbol`. Read any `[WARN]` advice it prints about the new symbol.
4. **Check**: `pm-config-show` shows the deployed configuration, including the new symbol and its seed quote.
5. **Commit** the authored YAML to version control. The artifact is rebuilt from it, so the YAML is your record of the listing.
6. **Market-maker bot**: a `pm-mm-bot` started with `--all-symbols` (as the `mm-demo` profile does) quotes the new symbol automatically. A bot with an explicit `--symbols` list or a config-file `symbols:` block does not. Add the symbol there (see MM Bot).
7. **Obligations**: if the market maker needs obligations specific to this symbol, add `mm_obligations.<SYMBOL>` to its gateway *before* listing. The seed quote is then sized against it.
8. **Index membership** is a separate decision. The symbol does not join any index. Add it later as a constituent (see Market Index and Index Administration (`pm-index-admin-cli`)).
9. **Start** the exchange. With `sessions_enabled: true`, the new symbol takes part in the opening auction like every other symbol. That is where a real IPO's opening price is discovered (see Auctions & Scheduling). Without sessions, continuous trading starts at once against the seed quote.
10. **Watch the first trades**. The circuit breaker is already armed at the offer price.

Remote and container deployments

Run the command against the exchange's own data directory (the same `EDUMATCHER_DATA_DIR`). The running-engine check looks at the local machine only: the engine's publish port and the local process table. It cannot prove that an engine on another host, or in a container that does not publish port 5556, is stopped. Check that yourself.

13.11 Options

Option	Default	Meaning
`--config PATH`	the deployed source	Authored YAML to edit
`--symbol NAME`	required	New symbol
`--ipo-price PRICE`	required	Offer price, and the reference price
`--outstanding-shares N`	required	Shares outstanding (used for index market caps)
`--tick-decimals N`	`2`	Price grid, 0..8
`--mm-gateway-id ID`	the only MM gateway	Who posts the seed quote
`--mm-bid-price`, `--mm-ask-price`	max spread around the price	Explicit seed prices; both or neither
`--mm-bid-qty`, `--mm-ask-qty`	`1000`	Seed sizes
`--mm-tif`	`DAY`	`DAY` or `GTC`
`--[no-]mm-seed-once`	on	Skip the seed when a restored quote exists
`--field KEY=YAML_VALUE`	none	`level`, `collar`, `order_limits`, `circuit_breaker`; repeatable

13.12 Where to go next

- IPO Valuation - finding the offer price with `pm-valuation`
- The Configuration Workflow - authored YAML, the compiled artifact and `pm-config-deploy`
- Config Verifier - what the validation codes mean
- Running the Exchange - stopping and starting with `pm-opctl-cli`
- Market Making - seed quotes and obligations
- Risk Controls - collars and circuit breakers around the IPO price
- Market Index - adding the new symbol to an index
- Persistence - the saved state that can override a listing

CHAPTER 14

IPO Valuation (`pm-valuation`)

Learning objectives

After reading this page you will understand:

- What an initial public offering (IPO) is, who takes part, and the steps from the decision to go public to the first trade
- Why a sum of money today is worth more than the same sum later, and how interest rates, risk and discounting express that
- How a company is valued from its future cash flows (a DCF) and from what investors pay for similar companies (comparables)
- How a young company's forecast is built from its market (TAM, SAM), its customers and its costs
- How scenarios and a Monte Carlo simulation show what the company might be worth, not just one number
- What the offering document is (a prospectus in Sweden, an S-1 in the US), and how book-building turns a valuation into an offer price
- How a Swedish IPO and a US IPO differ, and how to switch `pm-valuation` between them
- Why the final price is part calculation and part judgement, and what goes wrong when it is too high or too low
- How to use `pm-valuation`: the interview, the report, the PDF, scenario files and the hand-off to `pm-new-symbol`

How to read this page

The page has two parts. **Part I** (up to Using pm-valuation) teaches the finance: no background in investment banking or accounting is assumed. Every idea is introduced with a small worked example you can check with a calculator. **Part II** is the reference for the tool. If you already know what a DCF and a book-build are, skip to Part II.
The numbers in the worked examples are chosen to be easy to follow. Where a section says *"in the Tornfalk case"*, the numbers come from the classroom case `pm-valuation --case tornfalk`, so you can find them in the tool's own report.
`pm-valuation` works in two **frameworks**: a Swedish IPO (the default: amounts in SEK, a prospectus approved by Finansinspektionen, a listing on Nasdaq Stockholm) and a US IPO (`--market us`: USD, an S-1 filed with the SEC). Part I explains both; its examples are Swedish. A *billion* is 1,000 million, a Swedish *miljard* (not a *biljon*), written "bn".

14.1 Going public: what an IPO is

A young company is usually owned by a small group: its founders, its employees (through shares and options) and venture capital funds that invested in return for part of the company. These **private** owners cannot easily sell their shares, and the company can raise new money only by negotiating with a few investors at a time.

An **initial public offering** (IPO) changes that. The company sells shares to the public for the first time, and the shares are then **listed** on a stock exchange, where anyone can buy and sell them. Companies go public to:

- **raise capital** for growth: new shares are sold and the money goes to the company (the *primary* offering);
- **give existing owners a way out**: some existing shares may be sold in the same offering (the *secondary* offering), and all shares become tradable after a waiting period;
- **gain a currency**: listed shares can pay for acquisitions and reward employees;
- **gain visibility and credibility** with customers, suppliers and lenders.

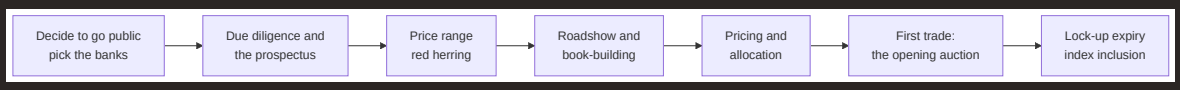
The price of this is permanent public scrutiny, heavy reporting duties, the cost of the offering itself, and the loss of some control.

14.1.1 Who takes part

Party	Role in the IPO
Issuer	The company selling shares. Its management and board decide to go public and must agree the price.
Existing shareholders	Founders, employees and venture investors. Every new share sold dilutes them; a secondary offering lets some of them sell.
Underwriters	Investment banks that organise the offering, write the prospectus with the lawyers, market the shares and, in a *firm commitment* underwriting, buy the shares from the company and resell them. They are paid a fee, the **gross spread** or commission: typically around 7% of the money raised in a mid-sized US IPO, and roughly half that in Europe.
Institutional investors	Pension funds, mutual funds, hedge funds and insurers. They buy most of an IPO and their orders decide the price.
Retail investors	Private individuals, usually offered a smaller tranche. In Sweden they apply through their bank or an online broker.
Cornerstone investors	Large investors who commit to buy a fixed amount before the offer opens. Common in Swedish IPOs, rare in US ones.
The regulator	In Sweden **Finansinspektionen** (FI), which must approve the prospectus; in the United States the Securities and Exchange Commission (SEC), which must declare the registration statement effective. No shares can be offered before that.
The exchange	Lists the shares and runs the first trading. On EduMatcher the opening auction finds the first price.

14.1.2 Primary and secondary markets

The IPO itself happens in the **primary market**: the company sells shares to investors at the **offer price**, and the money flows to the company. From the first trade onwards, investors trade with each other in the **secondary market**, the exchange, and the company receives nothing more. The difference between the offer price and the price at which trading ends on the first day is the **first-day return**, or **pop**.



Each step is explained in The IPO process below, after the finance that the price is built on.

14.2 The pricing dilemma: science and black magic

Everything in this page leads to one number: the **offer price**. It is the hardest number in the whole process, because both ways of getting it wrong are expensive.

	Price too high	Price too low
In the book	Investors refuse to order: the book is thin or undersubscribed	The book is many times oversubscribed

	Price too high	Price too low
On the first day	The stock falls below the offer price ("breaks issue"); buyers lose money at once	The stock jumps: a large pop
Who pays	The investors who bought, and the bankers' reputation with them	The company and the selling shareholders: they sold for less than the market would pay
Worst case	The IPO is pulled or postponed; the company may not get another chance for months	*Money left on the table*: the pop times the shares sold, a real loss to the company
Longer term	A "broken" IPO is remembered; later fund-raising is harder	A modest pop is good marketing; a huge one looks like a mistake

Too high, and nobody buys. Too low, and you give money away. The right price is somewhere between, and it is found by two very different kinds of work:

- **Hard finance.** A valuation built from the company's expected cash flows, a discount rate and the prices of comparable companies. It is arithmetic: given the assumptions, anyone gets the same answer. It gives a **range** of defensible values.
- **Judgement, negotiation and market mood** — the "black magic". What investors will actually pay on the day depends on sentiment, on which stocks are in fashion, on the story management tells, on how badly the founders want a higher number than their last private round, and on what the bankers think their investor clients will accept. None of this is in a spreadsheet, and all of it moves the price.

`pm-valuation` makes both halves visible. The valuation chapters of its report are the hard finance. The pricing chapter simulates the black magic with labelled heuristics (interest levels, a hype factor, a coverage target), so you can see *which way* each force pushes the price. The final judge is the opening auction on EduMatcher, where your fellow students' orders decide whether the price was right.

14.3 Money has a time value

The whole of valuation rests on one idea: **a sum of money today is worth more than the same sum in the future.** There are three reasons.

1. **Opportunity.** Money you have today can be invested and grow. If you wait a year for it, you lose a year of growth.
2. **Inflation.** Prices rise, so the same amount buys less later.
3. **Risk.** A promise of future money may not be kept. The further away and the less certain the payment, the less it is worth today.

The rest of this section turns these three reasons into arithmetic.

14.3.1 Interest and compounding

If you deposit 1,000 at an interest rate of 5% a year, after one year you have $1{,}000 \times 1.05 = 1{,}050$. In the second year you earn interest on the interest as well, $1{,}050 \times 1.05 = 1{,}102.50$, and after three years 1,157.63. This is **compounding**. In general, an amount PV invested at a rate r for n years grows to a **future value**

$$FV = PV \times (1 + r)^n$$

Compounding is slow at first and powerful later. A handy approximation, the *rule of 72*: money doubles in about 72/(rate in %) years. At 8% it doubles in about 9 years.

14.3.2 Present value and discounting

Turn the question around: **what is a payment in the future worth today?** If 1,000 arrives in three years and you could earn 5% a year meanwhile, it is worth the amount that would grow to 1,000 in three years:

$$PV = \frac{FV}{(1+r)^n} = \frac{1{,}000}{1.05^3} = 863.84$$

This is **discounting**, and r is the **discount rate**. The factor $1/(1+r)^n$ is the **discount factor**: the value today of 1 received in year n. Two things make a future payment worth less today: a higher rate, and a longer wait.

Discount factor	Year 1	Year 5	Year 10	Year 20	Year 30
at 4%	0.962	0.822	0.676	0.456	0.308
at 10%	0.909	0.621	0.386	0.149	0.057
at 20%	0.833	0.402	0.162	0.026	0.004

Read the table as prices: at a 10% discount rate, 1 promised in ten years is worth 0.386 today; at 20% it is worth 0.162. The same promise of 1,000 in three years is worth 863.84 at 5% but only 657.52 at 15%. **The discount rate is the single most powerful number in a valuation**, because it acts on every future cash flow at once, and hardest on the distant ones.

14.3.3 The risk-free rate

Which rate should you discount at? Start with the return you can get with **no risk** at all: lending to a government that can always repay in its own currency. The yield on a long-dated government bond (for a Swedish company the 10-year Swedish government bond, for a US company the 10-year US Treasury) is used as the **risk-free rate**, r_f. It pays for time and expected inflation, but not for risk. `pm-valuation` defaults to 3.0% in the Swedish framework (about the Swedish 10-year yield in August 2026) and 4.25% in the US one. Check today's value before you rely on it.

14.3.4 Risk and the required return

Nobody buys shares in a loss-making technology company to earn the risk-free rate. Investors demand a **risk premium** on top: the riskier the investment, the higher the return they require, and so the higher the rate at which they discount its future cash flows. The **required return** is the discount rate.

The standard way to set it for shares is the **capital asset pricing model** (CAPM):

$$r_e = r_f + \beta \times ERP$$

- The **equity risk premium** (ERP) is the extra return investors expect from the stock market as a whole over the risk-free rate, usually estimated at 4–6% a year. `pm-valuation` defaults to 5.6% in the Swedish framework (the median in PwC's 2026 survey of Swedish practitioners) and 5% in the US one.
- **Beta** (β) measures how strongly a stock moves with the market. A beta of 1 moves with the market; 1.5 moves 50% more, up and down. Young technology companies have high betas.

For a Swedish company with a beta of 1.5: $r_e = 3.0\% + 1.5 \times 5.6\% = 11.4\%$. Practitioners add further premia that CAPM leaves out, for small size and for the risk that a young company fails to deliver its plan. In the Tornfalk case the first five years are discounted at 15.9%: 11.4% plus a 1.5% size premium plus a 3% execution premium. From year 6, when the company is expected to be established, the rate falls to 9.66% ($3.0\% + 1.1 \times 5.6\% + 0.5\%$).

When a company also borrows, the lenders require a return too (the interest rate, reduced by the tax it saves). The average of the two required returns, weighted by how much of the company is financed each way, is the **weighted average cost of capital** (WACC). Most young companies have little debt, so their WACC is close to the cost of equity.

Risk lowers value

Higher risk means a higher discount rate, and a higher discount rate means a lower value today for the same expected cash flows. This is why the market pays less for a promise from a start-up than for the same promise from a government.

14.3.5 Streams of cash: NPV, perpetuities and growth

A company does not pay one amount once; it produces a **stream** of cash flows. Its value today is the sum of the present values of every flow. For an investment, subtracting what it costs today gives the **net present value** (NPV).

Example. A machine costs 1,000 today and earns 400 a year for three years. At a 10% discount rate:

Year	Cash flow	Discount factor	Present value
1	400	0.9091	363.64
2	400	0.8264	330.58
3	400	0.7513	300.53
		Total	**994.74**

The machine earns 1,200 in total but is worth only 994.74 today, so its NPV is $994.74 - 1{,}000 = -5.26$: at a 10% required return it is not quite worth buying.

Companies are expected to last indefinitely. A cash flow C that continues for ever (a **perpetuity**) is worth C/r; one that starts at C next year and then grows at a constant rate g for ever (a **growing perpetuity**) is worth

$$PV = \frac{C}{r - g}$$

At $r = 10\%$, a perpetual 100 a year is worth 1,000. If it grows at 2.5% a year it is worth $100/(0.10 - 0.025) = 1{,}333$; at 5% growth, 2,000; at 8%, 5,000. Two lessons follow. First, value is extremely sensitive to long-run growth when g is close to r. Second, g must stay below r, and in practice at or below the long-run growth of the economy (2–3%), or the formula claims the company will one day be larger than the world.

14.4 What a company is worth

A share is a claim on part of a company's future. So the value of a company is, in principle, the present value of all the cash it will ever be able to pay its owners. That principle is simple; applying it to a company that has never made a profit is where the work lies.

14.4.1 Three ways to value a company

Approach	Question it asks	Used for
Intrinsic value (discounted cash flow, DCF)	What are the company's expected future cash flows worth today?	The core of the valuation; forces every assumption into the open
Relative value (comparables, multiples)	What do investors pay for similar listed companies, per unit of revenue or profit?	A check on the DCF, and what investors in the book will actually compare
Asset value	What would the company's assets fetch if sold?	Banks, property and failing companies; rarely useful for a software company whose value is its customers and people

The DCF and the comparables rarely agree. The DCF reflects *your* forecast; comparables reflect the *market's* current mood about the sector. When the two differ widely, one of them is telling you something: either your forecast is too cautious, or the sector is priced for perfection. `pm-valuation` blends them (70% DCF, 30% comparables by default) and warns when they differ by more than 50%.

14.4.2 From profit to cash

An income statement starts with **revenue** and subtracts costs to reach profit. A simplified one, in the order `pm-valuation` reports it:

Line	Meaning
Revenue	What customers pay
− Cost of revenue	Running the service: infrastructure, support, operations staff
= **Gross profit**	What is left to pay for everything else; as a share of revenue, the *gross margin*
− Research and development (R&D)	Building the product
− Sales and marketing (S&M)	Winning customers
− General and administrative (G&A)	Finance, legal, management, the cost of being listed
− Stock-based compensation	Pay in shares; a real cost, because it dilutes every other owner
= **EBITDA**	Earnings before interest, taxes, depreciation and amortisation
− Depreciation and amortisation (D&A)	The cost of equipment spread over its useful life
= **EBIT**	Earnings before interest and taxes: the operating profit

Profit is not cash. Three adjustments turn operating profit into the cash the business actually generates:

- **Taxes** are paid on profit. Losses in early years are carried forward (a *net operating loss*, NOL) and reduce the tax on later profits.
- **Capital expenditure** (capex) is cash spent on equipment now; the income statement only shows it slowly, as depreciation. So add back D&A and subtract capex.
- **Working capital** is cash tied up in running the business: customers who have not yet paid, minus suppliers not yet paid and customers who paid in advance. Growth usually ties up more of it. Subscription companies that bill a year in advance are the happy exception: their growth *releases* cash.

The result is the **free cash flow to the firm** (FCFF): the cash available to everyone who financed the company, before any of it is paid to lenders or owners.

$$FCFF = EBIT - \text{taxes} + D\&A - \text{capex} - \Delta\text{working capital}$$

Example — Tornfalk, year 5 (SEK m): EBIT 1,101.9 − taxes 122.7 + D&A 100.2 − capex 248.1 − change in working capital (−94.4) = **925.7**. The change in working capital is negative because Tornfalk's customers pay in advance, so growth adds cash rather than using it.

14.4.3 Enterprise value and equity value

Think of buying a house with a mortgage. The house is worth 500,000; the mortgage is 300,000; the owner's share (the equity) is worth 200,000. If the house also contains a safe with 20,000 in it, the owner's equity is worth 220,000.

A company works the same way. Discounting FCFF gives the value of the business as a whole, the **enterprise value** (EV), which belongs to lenders and owners together. To reach the value of the shares, the **equity value**:

$$\text{Equity value} = EV + \text{cash} - \text{debt}$$

Dividing by the number of shares gives the **value per share**, the number the offer price is compared with. In the Tornfalk case: EV 10,330.4 m SEK, plus 900.0 m net cash, minus 180.0 m in offering fees, divided by 120 million pre-IPO shares, is a DCF value of 92.09 SEK per share.

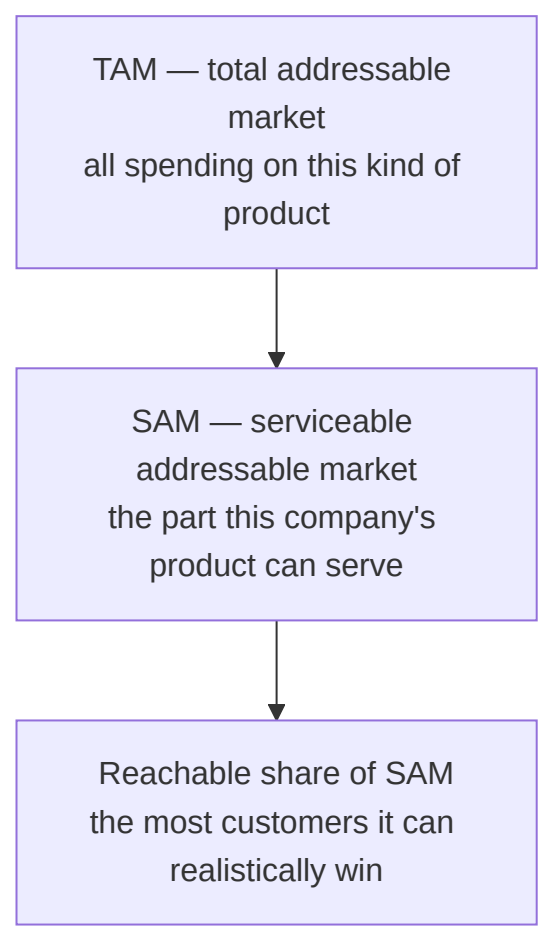
Why the money raised does not change the value per share

Selling new shares at a *fair* price brings in exactly as much cash as the new shares are worth, so existing owners neither gain nor lose, except for the fees. The model therefore values the company before the IPO and deducts only the costs of raising the money.

14.5 Forecasting a young company

A DCF needs cash flows for years to come. For a mature company, next year looks like last year plus a little. For a young company growing 50% a year, you have to build the forecast from its drivers. `pm-valuation` does this from the bottom up, and so should you when you answer its questions.

14.5.1 How big can it get? TAM, SAM and the reachable share



- The **total addressable market** (TAM) is everything customers spend on this kind of product worldwide: the whole pond.
- The **serviceable addressable market** (SAM) is the part the company's product, geography and price can actually serve.
- The **reachable share** (sometimes called the *serviceable obtainable market*, SOM) is the part of SAM the company can realistically win against competitors. It depends on the market structure: a fragmented, competitive market allows a smaller share than one where a few firms dominate.

In the Tornfalk case in year 1: a TAM of 448 mdr (miljarder kronor), of which 25% is serviceable (112 mdr), of which Tornfalk can reach at most 8%, about 9 mdr of revenue, or 14,222 customers at 630,000 SEK a year each. It has 2,600 today: plenty of room to grow, but not unlimited room.

14.5.2 Customers, churn and price

Revenue is built from customers:

- Each year the company **wins** new customers (*gross adds*), faster while it is small relative to its reachable market and slower as it approaches that ceiling. This S-shaped path is called **logistic** growth.
- It **loses** a share of its customers every year: the **churn** rate.
- Each customer pays the **average revenue per user** (ARPU), which grows with price rises and upselling.

Tornfalk, year 1: 2,600 customers at the start, 1,508 won, 208 lost (8% churn), 3,900 at the end. Revenue is ARPU times the

average number of customers during the year: 2,047.5 m SEK.

Unit economics ask whether a customer is worth what it costs to win. The *lifetime value* (LTV) is the gross profit a customer brings before leaving: with an ARPU of 60,000, a 75% gross margin and 8% churn, a customer stays on average $1/0.08 = 12.5$ years and is worth $60{,}000 \times 0.75/0.08 = 562{,}500$. Compare it with the *customer acquisition cost* (CAC), all sales and marketing spend divided by customers won. An LTV at least three times the CAC is a common mark of a healthy subscription business.

14.5.3 People and costs

Staff are the largest cost of a young technology company. Headcount grows with revenue, but less than proportionally (the model's *elasticity*, 0.6 by default: 10% more revenue needs about 6% more people). Revenue per employee therefore rises as the company grows, and the operating margin with it. This **operating leverage** is why investors accept years of losses: they are paying for the margin the company will have once it is large.

14.6 The discounted cash flow, step by step

A DCF applies the time value of money to the forecast. Here it is on a small company, with round numbers, before looking at what `pm-valuation` adds.

Step 1 — Forecast the free cash flows. Five years: −20, −5, 10, 25, 35 (millions). The early losses are typical of a growing company.

Step 2 — Choose the discount rate. 12%.

Step 3 — Discount each year.

Year	FCFF	Discount factor at 12%	Present value
1	−20	0.8929	−17.86
2	−5	0.7972	−3.99
3	10	0.7118	7.12
4	25	0.6355	15.89
5	35	0.5674	19.86
		Sum, years 1–5	**21.02**

Step 4 — Add a terminal value. The company does not stop after year 5. Assume its cash flow grows at 3% a year for ever from then on. With the growing perpetuity formula, the value at the end of year 5 of all later years is the **terminal value**:

$$TV_5 = \frac{FCFF_6}{r - g} = \frac{35 \times 1.03}{0.12 - 0.03} = 400.56$$

It is worth $400.56 \times 0.5674 = 227.29$ today.

Step 5 — Enterprise value. $21.02 + 227.29 = 248.31$ million.

Step 6 — Bridge to value per share. With 50 m cash, 20 m debt and 10 m shares: $(248.31 + 50 - 20)/10 = 27.83$ per share.

Most of the value is in the far future

In this example **91.5%** of the enterprise value is the terminal value: the years after the forecast ends. That is normal for a growth company, and it is the DCF's great weakness. Change the discount rate by one point and the value per share moves from 27.83 to 31.99 (at 11%) or 24.53 (at 13%). Change long-run growth by one point and it moves to 25.36 (2%) or 30.92 (4%). A DCF is only as good as its assumptions, which is why the report shows sensitivity grids and why the offer price is not simply the DCF value.

14.6.1 What pm-valuation does differently

The model follows the same steps with more care in three places:

- **Two stages.** Years 1–5 are discounted at a higher rate (risky growth) than years 6–10 (established business). Each year's discount factor is the product of every earlier year's rate, so later years carry the high early rate too.
- **Ten years, not five,** so that growth can slow before the terminal value takes over.
- **Growth must be paid for.** The simple formula above lets the company grow for ever for free. The model's terminal value instead asks how much the company must reinvest to grow at g, given the return it earns on new investment (RONIC).

$$TV_N = \frac{NOPAT_{N+1} \times \left(1 - \frac{g}{RONIC}\right)}{r_2 - g}$$

NOPAT is operating profit after tax. If new investment earns only the cost of capital, growth adds no value at all. In the Tornfalk case, the terminal value is 66.6% of an enterprise value of 10,330.4 m SEK.

14.7 Comparables: what the market pays

The second method asks what investors currently pay for similar listed companies, expressed as a **multiple**: enterprise value divided by revenue, by EBITDA, or by profit. Young companies without profits are compared on **EV / revenue**, usually next year's revenue.

In the Tornfalk case, listed business-software companies trade at 10 times next year's revenue. Tornfalk's next-year revenue of 2,047.5 m SEK therefore implies an enterprise value of 20,475 m SEK, and 176.62 SEK per share: almost double the DCF's 92.09. The blend, 70% DCF and 30% comparables, is the **fair value** of 117.45 SEK per share.

Multiples are quick and they are what investors in the book will look at. But they import the market's mood wholesale: when a sector is in fashion, every company in it looks cheap next to its peers.

14.8 Uncertainty: scenarios, sensitivity and Monte Carlo

Every input to the forecast is a guess. A single fair value hides how good or bad the guess might be. Three tools show the range.

Scenarios. Set every driver to a pessimistic value at once (the *bear* case) and then to an optimistic value (the *bull* case). In the Tornfalk case fair value ranges from 16.94 (bear) through 117.45 (base) to 307.41 SEK (bull): a reminder of how much is unknown.

Sensitivity and the tornado. Move one driver at a time to its bear and bull values, keeping everything else at base, and rank the drivers by how far they move fair value. The ranked bars look like a tornado. The drivers at the top are the ones worth arguing about; for Tornfalk it is how fast headcount grows with revenue, then how much of its market it can reach.

Monte Carlo simulation. Scenarios move all drivers together to their extremes, which is unlikely; the tornado moves one at a time, which ignores combinations. A **Monte Carlo** simulation does neither. It builds thousands of possible companies, each with every driver drawn at random from its range, and values each one exactly as the base case. The result is a **distribution** of fair values:

- The **median** (P50) is the middle company; P5 and P95 bound the central 90%.
- Drivers are **correlated**: in reality good news tends to arrive together (strong growth, low churn, a strong market). The model links the draws through a common factor with correlation ρ (0.5 by default).
- The most useful single number is the share of simulated companies worth **less than the offer price**: the probability that IPO buyers overpay.

In the Tornfalk case (10,000 companies): P5 58.53, median 110.48, P95 174.18 SEK; and in 43.8% of them fair value is below the 105.00 SEK offer price. The book was twelve times covered, yet for the fundamentals the price is close to a coin flip.

14.9 The IPO process

With the finance in place, here is how a valuation becomes an offer price. `pm-valuation` follows the Swedish process by default: a prospectus approved by Finansinspektionen, and a listing on Nasdaq Stockholm. With `--market us` it follows the American one: a Form S-1 filed with the SEC, and a listing on Nasdaq or NYSE. The two differ in names, legal detail and a few conventions, not in substance:

	Sweden (`--market se`, the default)	United States (`--market us`)
Company form	Public limited company, **AB (publ)**	Corporation, **Inc.**, usually incorporated in Delaware
Offering document	**Prospectus**, under the EU Prospectus Regulation	Registration statement, **Form S-1**, which contains the prospectus
Who checks it	**Finansinspektionen** (FI) approves it	The **SEC** reviews it and declares it effective
Exchange	Nasdaq Stockholm, Main Market	Nasdaq or NYSE
Price range	Stated in the prospectus; its top is the **maximum price**	Filed in an amended S-1; the final price may be up to about 20% outside it
Bank fees (model default)	3% of the money raised	7% of the money raised
Buyers	Institutions, often **cornerstone investors**, and an offer to the public	Mainly institutions
Currency	SEK	USD

14.9.1 Preparing: the banks and the prospectus

The company chooses its **underwriters**, one or more investment banks led by a *bookrunner*, in Swedish deals often called the *global coordinator*. Months of **due diligence** follow: bankers, lawyers and auditors check everything the company will say about itself, because everyone responsible for the offering document is liable if it is misleading. Only a public company may offer its shares to the public, so a Swedish private company (AB) first becomes a public one, **AB (publ)**. A few weeks before the offer, it usually announces its **intention to float**, which starts the public conversation about the IPO.

The result is the **prospectus**, the document investors read. In Sweden, Finansinspektionen must approve it before the offer opens. In the United States it is the main part of the **registration statement, Form S-1**, filed with the SEC. The headings differ; the content is much the same:

Part of the prospectus	What it tells an investor
Summary	The business and the offering in a few pages; the S-1 also has a cover page with the price, the shares and the banks' fee
Risk factors	Everything that could go wrong, from competition to key people leaving
Reasons for the offer and use of proceeds	Why the company lists, and what it will do with the money
Capitalisation and **dilution**	How the share count changes, and how much new investors pay over the book value per share
Operating and financial review (MD&A in an S-1)	Management's explanation of the numbers
Business and market	The company, its market and its strategy
Financial statements	Audited accounts
Terms of the offer, or Underwriting	The price range, how shares are allocated, the banks, their fees and the lock-up agreements

Smaller companies may publish lighter documents. In the EU, small and medium-sized companies, and companies listing on a growth market such as Nasdaq First North, may use the shorter **EU Growth prospectus**. In the United States, an

emerging growth company (EGC, annual revenue below about 1.235 billion dollars) may, for example, present fewer years of audited accounts, and a **smaller reporting company** (SRC) qualifies by its public float or revenue. The report's first section is the document's cover in brief: the **Prospectus cover** with the issuer, its ICB industry code, Finansinspektionen and the listing venue, or, with `--market us`, the **S-1 cover** with the SIC code and the EGC and SRC status, which `pm-valuation` works out from your answers.

14.9.2 The price range and the IPO discount

When the offer opens, the prospectus is published with a **price range**, for example 94.50–105.00 SEK per share. In the United States, the company files an amended S-1 with the range and prints the **preliminary prospectus**, nicknamed the *red herring* for the red warning on its cover that the registration is not yet effective.

The range is deliberately set **below** fair value. The gap is the **IPO discount**, typically 10–15%, and there are good reasons for it:

- New investors take a risk on a company with no trading history; the discount pays them for it.
- A stock that rises on its first day creates goodwill and attention, and makes the next offering easier.
- Bankers want a book of orders several times larger than the offering, so that the price holds when trading starts (see below).

In the Tornfalk case, fair value is 117.45 SEK, a 15% discount gives a midpoint of 99.83, and the range is rounded to 94.50–105.00 SEK. If management insists on a minimum valuation, for example no lower than the last private funding round, the range moves up to meet it, and the discount shrinks. When it shrinks too far, the IPO cannot be sold (the Halvard case).

14.9.3 The roadshow and book-building

Often before the offer is even launched, a few large investors commit to buy a fixed amount of money's worth of shares at whatever price is set. These **cornerstone investors** are common in Sweden. Their names in the prospectus tell other investors that professionals have already checked the company.

Then, for one to two weeks, management and the bankers present the company to institutional investors in meetings and presentations: the **roadshow**. The investors respond with **indications of interest**: how many shares they would buy, and at what maximum price. These orders are not binding, but reputations depend on them. In Sweden the offer is usually also open to the public: private investors apply for shares through their bank or online broker during the same period.

The bookrunner collects all indications in **the book**, and so learns how much demand there is at every price. The key number is **coverage**, or oversubscription: demand divided by the value of the shares offered.

Price (SEK)	Coverage (Tornfalk's book)
75.50 (20% below the range)	31.41×
94.50 (bottom of the range)	16.22×
105.00 (top of the range, the maximum price)	11.92×

Demand falls as the price rises, as it should. A book covered 12 times means investors asked for twelve times the shares on offer. Such a deal is called **hot**.

14.9.4 Pricing and allocation

When the book closes, the company and the bookrunner set the **offer price**. The usual rule: the highest price at which the book is still comfortably oversubscribed, typically at least three times.

How high that can go depends on the market. In Sweden, the prospectus states the top of the range as the **maximum price**. Private investors applied on that promise, so a higher price would need a supplement to the prospectus that lets everyone withdraw, and in practice the maximum price is a hard limit. `pm-valuation` uses it as one. In the United States a deal may price about 20% outside the filed range without re-filing, and with `--market us` that is the limit instead. Tornfalk prices at the maximum price, 105.00 SEK, and is still 11.92 times covered: an American deal would have raised the price above the range.

At 12 times coverage most investors get only a fraction of what they asked for. **Allocation** is at the bookrunner's discretion: cornerstones receive their shares first, then long-term investors are favoured, and so, it is often said, are the bank's best clients.

Tornfalk's institutions receive about 8% of their orders, its private investors about 22%. Unfilled investors who still want the stock must buy it on the exchange on the first day. That unfilled demand drives the first-day price up.

If the book is covered less than the target, the deal can still price at the bottom of the range on a **thin book**, with a real risk of trading below the offer price. If it is not covered even once, the IPO is **postponed**.

Real IPOs usually also include an **over-allotment option**, called the **greenshoe** in the United States: the banks may sell up to 15% more shares and buy them back in the market if the price falls, which supports the price. `pm-valuation` leaves it out.

14.9.5 The first day and after

Trading starts with an **opening auction**: buy and sell orders collected before the open are matched at the single price that trades the most shares (see Auctions & Scheduling). That price, not the offer price, is the market's first verdict.

The first-day return is the **pop**. IPOs in Sweden, the United States and most other countries have on average closed their first day above the offer price, in the United States somewhere in the high teens of percent over the past four decades, with enormous variation: some double, some fall. The pop is a gain for the investors who were allocated shares, and a cost to the company:

$$\text{Money left on the table} = \text{pop} \times \text{offer price} \times \text{shares sold}$$

Tornfalk's expected pop is 29.8%. On 38.1 million shares sold at 105.00 SEK, that is 1,193.1 million SEK, about 1.2 miljarder kronor, that the company could have raised but did not.

Two dates matter after the IPO:

- **Lock-up expiry.** Existing owners usually agree not to sell for 180 days, sometimes longer. When the lock-up ends, many more shares can be sold, and the price often weakens in anticipation. For Tornfalk, 120 million pre-IPO shares, 3.2 times the free float, are released.
- **Index inclusion.** Stock-market indices require a minimum size, a minimum free float (shares available to trade) and usually a *seasoning* period of trading before a new stock can join. Inclusion brings buying from index funds, so investors in the book value the prospect of it. See Market Index.

14.10 The black magic: what the formulas cannot tell you

The valuation gives a defensible range. What decides where in that range, or outside it, the price ends up is not arithmetic:

- **Market mood and the IPO window.** When stocks are rising and recent IPOs have done well, investors queue up; after a market fall, the same company cannot be sold at any sensible price. Companies time their IPOs for an open *window*.
- **Fashion.** Sectors go in and out of favour, and the comparables multiple moves with them. A company valued at 10 times revenue one year may be worth 4 times the next with no change in its business.
- **The story.** Investors buy a narrative about the future: a new market, a platform, a famous founder. A convincing story shows up as demand in the book, not as a change in the discount rate.
- **Anchoring.** Founders and venture investors anchor on the valuation of the last private funding round. Pricing below it (a *down round*) is painful for them, so they may set a floor that the market will not meet.
- **Conflicting incentives.** The bankers' fee is a percentage of the money raised, which favours a higher price. But their long-term business is with the institutional investors who buy IPOs, which favours a lower price and a pop. The company wants the most money; it also wants a successful first day.
- **The self-fulfilling pop.** Oversubscription creates unfilled demand, which creates the pop, which is reported as success, which creates demand for the next IPO.

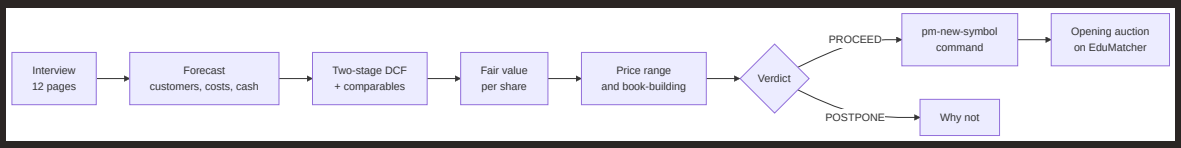
`pm-valuation` represents these forces with a few labelled **heuristics**, not finance theory. Investor interest levels and a hype factor move demand, a coverage target sets the pricing rule, and a formula links coverage to the expected pop. They show which way each force pushes, and how strongly compared with the others. They do not forecast the first day. On EduMatcher the opening auction does that, with real orders from real people.

Try it

Run `pm-valuation --case tornfalk`, press F5, then `b` to go back. Change only the investor interest and hype on page 9, press F5 again and `c` to compare. The valuation does not move at all, and the offer price stays at the maximum price, 105.00 SEK. But coverage falls from 11.92 to 5.01 times, and the expected pop from 29.8% to 16.9%. That is the black magic, isolated.

14.11 Using pm-valuation

Listing a New Symbol starts from an agreed offer price. `pm-valuation` is where that price comes from. It interviews you about a fictive company, values it with the methods of Part I, and simulates the IPO that sells its shares:



The tool is a teaching model, not a pricing engine. Every rate, multiple and sector preset is an assumption, and the demand and first-day-pop formulas are calibrated heuristics that show direction and causes. The real first-day price is found by the opening auction once the symbol is listed, which is the point of the exercise: the students' own orders decide whether their price was right. The formulas, presets and worked example are in the design document, `docs-design/EduMatcher-valuation.md`.

14.12 Quick start

Start from a classroom case, change what you like, and press F5:

```
pm-valuation --case tornfalk
```

Or print the report without the interview:

```
pm-valuation --case tornfalk --no-tui --mode deterministic
```

```
Tornfalk Security AB (TORN) — IPO valuation  PROCEED

1. Verdict
  PROCEED

   Fair value per share           117.45
   Price range              94.50–105.00
   Offer price                    105.00
   Market capitalisation      16,600.0 m
   Primary raise               4,000.0 m
   Coverage                       11.92×
   Expected first-day pop          29.8%
...
17. Next step
  pm-new-symbol --symbol TORN --ipo-price 105.00 --outstanding-shares 158095238 --tick-decimals 2
```

Without `--load` or `--case`, the interview starts empty: every answer has an automatic value, so pressing F5 straight away values an average Swedish B2B software company, in SEK.

14.13 Sweden or the United States

The market, page 1's last question, decides the IPO framework and every country-specific default. Sweden (`se`) is the default; `--market us` on the command line selects the United States and overrides the scenario file.

	se	us
Currency	SEK	USD

	se	us
Offering document and report section 2	Prospectus, approved by Finansinspektionen; **Prospectus cover** with the ICB industry code	Form S-1, filed with the SEC; **S-1 cover** with SIC code, EGC and SRC status
Default company name, incorporation, lead bank	Newco AB, Sweden, Fiktiva Banken AB	Newco Inc., Delaware, Fictive & Co.
Risk-free rate, equity risk premium	3.0%, 5.6%	4.25%, 5%
Tax rate, inflation, long-run growth	20.6%, 2%, 2%	25%, 2.5%, 2.5%
Bank fee (gross spread)	3%	7%
Maximum price above the range	0%: the top of the range is the maximum price	20%
Typical share price at the IPO	About 100 SEK	About 20 dollars

Money defaults, such as a sector's cost per employee or the index rulebook's minimum size, are converted into the market's currency at a fixed 10 SEK per dollar. Swedish salaries are set lower than American ones, and revenue per employee is scaled with them, so staff costs keep the same share of revenue in both markets.

The market is saved in the scenario file, and **the file's amounts are in that market's currency**. `--market us` on a Swedish case therefore reads 1.4bn of revenue as 1.4 billion dollars, not kronor, and values a company ten times larger. To compare the frameworks fairly, convert the amounts too.

Swedish and English count large numbers differently. A Swedish *miljard* is a thousand million, the English **billion**; a Swedish *biljon* is a million million. `pm-valuation` accepts both spellings for amounts (see Typing values). Field values are shown in the market's words: `400mdr` and `900mkr` in Sweden, `40bn` and `90m` in the United States, in the interview, in scenario files and in the report's list of assumptions. The report's own tables are in millions, `m`, which read the same in both languages.

14.14 Sector presets

The sector preset on page 1 is the most important single choice in the interview: it fills in most of the other answers with values that fit together for one kind of business. There are 18, grouped by industry the way Nasdaq Stockholm classifies listed companies. Nasdaq's Nordic exchanges have used FTSE Russell's **Industry Classification Benchmark** (ICB) since 2011: 11 industries, divided into about 170 subsectors, each with an eight-digit code. The US SEC files companies under four-digit **SIC codes** (Standard Industrial Classification). A preset carries both, and the cover of the report shows the one its market uses: *ICB 10101015 Software* on a Swedish prospectus, *SIC 7372* on an S-1.

Preset	ICB subsector	A customer is	Comparables multiple	Year-10 margin	What sets it apart
Technology					
`b2b_saas`	10101015 Software	business account	10×	20%–35%	Fast growth, few customers leave, high gross margin; losses now, wide margins later
`consumer_subscription`	10101020 Consumer Digital Services	paying subscriber	5×	15%–30%	Millions of small customers, a third of them leave each year
`online_marketplace`	10101020 Consumer Digital Services	active buyer	6×	15%–30%	Keeps a fee on each sale; heavy marketing to win buyers

Preset	ICB subsector	A customer is	Comparables multiple	Year-10 margin	What sets it apart
deep_tech_hardware	10102010 Semiconductors	enterprise account	3×	12%–25%	Hardware plus service; high capex, cash tied up in stock
consulting	10101010 Computer Services	client account	1.6×	8%–16%	Staff are nearly all the cost, so margins barely widen with size; little capex
Telecommunications					
telecom_operator	15102015 Telecommunications Services	subscriber	2.6×	15%–28%	Heavy network capex and fixed costs; stable subscribers; low beta
Health Care					
medtech	20102010 Medical Equipment	hospital or clinic account	3.9×	18%–30%	High gross margin and heavy R&D; approval risk raises the execution premium
healthcare_services	20101010 Health Care Facilities	care contract or clinic	1.2×	6%–12%	Staff-heavy and stable; paid by regions, insurers or patients; low beta
Consumer Discretionary					
retail_stores	40401030 Specialty Retailers	store	0.6×	4%–10%	Growth by opening stores; goods bought in and rent eat most of revenue; stock ties up cash
ecommerce_retail	40401010 Diversified Retailers	active buyer	0.4×	3%–8%	Thin margins on goods bought in; 40% of buyers stop each year
video_games	40203040 Electronic Entertainment	paying player	4×	15%–30%	Hit-driven: more than half of paying players leave each year; high execution premium

Preset	ICB subsector	A customer is	Comparables multiple	Year-10 margin	What sets it apart
media_publishing	40301030 Publishing	subscriber or reader	1.4×	8%–15%	A slowly shrinking market; print, distribution and royalty costs
Consumer Staples					
consumer_brands	45102020 Food Products	retail or distribution account	1.6×	8%–15%	Ingredients and production are half of revenue; brand marketing; low beta
Industrials					
fintech_payments	50205015 Transaction Processing Services	merchant	7×	15%–30%	Grows with merchants' sales; card-network fees; cash tied up in settlement
industrial_machinery	50204000 Machinery: Industrial	industrial customer	1.2×	8%–16%	Materials are the largest cost; factories need capex; stock and invoices tie up cash
construction_services	50101015 Engineering and Contracting Services	project client	0.8×	4%–9%	Low margins on large projects; subcontractors and materials
logistics_transport	50206060 Transportation Services	shipper account	0.8×	5%–12%	Vehicles and warehouses need capex; fuel and hired haulage; low margins
Utilities					
renewable_energy	65101010 Alternative Electricity	power-purchase contract	6×	30%–50%	Capital-heavy: parks are built up front (capex 35% of revenue) and depreciated over 25 years

The comparables multiple is the enterprise value investors pay per unit of next year's revenue, and the year-10 margin is the range of EBIT margins the model considers plausible for the sector (warning V008 outside it). Software is valued at many times its revenue because most of each extra sale is profit; a construction company earns a few percent on each sale and is valued at less than its revenue. Every preset's default company is tuned to be coherent in both markets, but the values are illustrative, not researched benchmarks.

Choosing a preset. Pick the business model that is closest, not the product: a company selling software to hospitals is b2b_saas, not medtech. Then change the answers that do not fit; the preset only supplies starting values. Look at what "a

customer is" for the preset: for `retail_stores` the model counts stores, so "customers" grow as the chain opens shops and "churn" means closing them.

Companies without a preset. Some businesses are valued in other ways, and the model would teach them wrongly:

- **Banks and insurers** borrow and lend as their business, so the cash flow to the whole firm means little. They are valued on their equity and its return, for example price to book value.
- **Property companies** are valued on the market value of their buildings less their debt, the *net asset value*.
- **Pre-revenue biotech** has no customers yet. Its drugs in development are valued one by one, weighted by the chance each passes its trials (*risk-adjusted NPV*).
- **Mining and oil exploration** companies are valued on the resources in the ground and the cost of extracting them.

14.15 The interview

The interview is a full-screen terminal form with twelve pages, laid out like the other EduMatcher terminal programs (`pm-viewer`, `pm-board`): the EduMatcher badge, program and version in the top bar, and a rounded box around each area.

```
 EduMatcher   pm-valuation x.yy.z   |   Tornfalk Security AB (TORN)   |   Page 3/11   |   Level Intermediate
┌─ Pages ──────────────────────┐┌─ 3 Customers & pricing ──────────────────────────────────────┐┌─ Live preview ──────┐
│  1 Company              ✎4   ││ Last FY revenue            1.4mdr       ✎ you           ^ ││ Fair value          │
│  2 Market                    ││ Customers now              2,600        ✎ you             ││       117.45        │
│▶ 3 Customers & pricing  ✎2   ││ ARPU per year                           600,000 · preset ││ DCF        92.09    │
│  4 People                    ││ Annual churn                            8% · preset      ││ Comps     176.62    │
│  5 Costs                     ││ Customer growth, year 1                 50% · preset     ││                     │
│  6 Capital & tax        ✎1   ││                                                          ││ Range 94.50–105.00  │
│  7 Discount rates            ││                                                          ││ Offer 105.00 at 11.9×│
│  8 Offering             ✎2   ││                                                          ││                     │
│  9 Investors & sentiment ✎4  ││                                                          ││ PROCEED             │
│ 11 Management                ││                                                          ││                     │
│ 12 Simulation                ││                                                          ││                     │
│                              ││                                                        v ││                     │
└──────────────────────────────┘└─ F3 → Advanced: 1 more field here · 1 answer hidden ──────┘└─ F4 explain ─────────┘
┌─ Field description ──────────────────────────────────────────────────────────────────────────────────────────────┐
│ Annual churn — The share of customers lost each year. At 8% the average customer stays 1 / 0.08 = 12.5 years; at 35% │
│ (typical for consumer apps) less than 3. Churn sets a customer's lifetime value, the profit it brings before        │
│ leaving: ARPU × gross margin (the share of revenue left after the direct cost of serving customers) / churn.        │
│ Business software often churns 5-15% a year.                                                                        │
│                                                                                                                     │
│                                                                                                                     │
└─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
 Tab next · PgDn page · Enter pick · Ctrl-D clear · F1 glossary · F2 review · F3 level · F5 calculate · F9 save · Esc qu
```

- **Top bar:** the company, the page and the level.
- **Pages:** the pages at the current level; counts your answers on each.
- **The form**, titled with the page: one field per row, your answer or the automatic value beside it. Its bottom border says what F3 adds on this page.
- **Live preview:** fair value, the price range and the verdict, recalculated as you type. F4 explains each number with its current value: for example that fair value is 70% of the DCF value plus 30% of the comparables value, and why the offer price stopped where it did. See The live preview for every line in full.
- **Field description:** what the field in focus means, or what is wrong with its value.

Page	What it asks
1 Company	Name, ticker, **sector preset**, cover details, **market** (`se` or `us`)
2 Market	Addressable market, its growth, your reachable share
3 Customers & pricing	Last year's revenue, customers, price per customer, churn, growth
4 People	Headcount, cost per employee, how hiring follows revenue

Page	What it asks
5 Costs	Infrastructure, acquisition cost, R&D, G&A, stock-based pay
6 Capital & tax	Capex, working capital, tax, loss carry-forwards, cash and debt
7 Discount rates	Risk-free rate, equity risk premium, betas and premia per stage
8 Offering	Shares before the IPO, raise, fees, IPO discount, lock-ups
9 Investors & sentiment	Institutional and retail interest, hype, comparable multiple
10 Index	The fictive index rulebook, and what the exchange may relax
11 Management	Last private round, minimum market cap, maximum dilution
12 Simulation	Scenarios, Monte Carlo draws, seed, correlation

14.15.1 Levels: how much the interview asks

Not every question matters equally, and a beginner should not face 129 of them. The interview has four **levels**. Each shows the fields of the levels before it and adds more; every field it does not show keeps its automatic value, so the valuation is always complete.

Level	Fields	What it adds
Beginner (the default)	16	The essentials: name, sector, market, market size, revenue and growth, cash and debt, the risk-free rate, the raise and IPO discount, investor interest and hype, the last private round
Intermediate	43	Customers, price and churn; headcount; tax; the equity risk premium, betas and terminal growth; share count and fees; the comparable multiple and DCF weight; management's limits
Advanced	88	Costs in detail, capital and working capital, the size and execution premia, lock-ups and cornerstones, the size of the book, the index rulebook, the Monte Carlo settings
Expert	129	Everything: staff splits, WACC and horizon settings, rate overrides, the bear and bull value of every scenario driver

Start at a level with `--level`, and change it at any time with F3, which steps Beginner → Intermediate → Advanced → Expert and back. The title bar shows the level. The bottom border of the form says how many fields the next level adds on this page, and warns when answers you typed or loaded sit in fields the current level hides: they are still used. Pages with no fields at the current level are left out of the page list; at Beginner, pages 4, 5, 10 and 12 are. The Field reference below explains every field and gives its level.

14.15.2 Automatic values and their source

Every question can be skipped. The hint beside a field shows what the model will use instead, and where it comes from:

Source	Meaning
you	Your answer. It always wins
preset	The sector preset chosen on page 1
derived	Computed from other answers, e.g. customers from revenue ÷ price
default	A fixed default, or the market's, e.g. the Swedish 20.6% tax rate

F2 opens the review page: every value in one table with its source, which is also section 3 of the report.

14.15.3 Typing values

Field	Accepts	Notes
Money and counts	`90m`, `1.4bn`, `2.5k`, `1_000`, `60,000`	Swedish suffixes too: `90mkr` (miljoner kronor), `1.4md` or `1.4mdr` (miljarder). In the Swedish market, amounts are shown with `mkr` and `mdr`
Percentages	`12`, `12%`, `0.5`	A plain number is **percentage points**: `0.5` is 0.5%, never 50%
Ratios	`10`, `10x`	
Yes / no	`yes`, `no`	
Choices	Enter opens a pick-list	Sector, market structure, interest levels, …
Optional fields	`none`	"Not given", e.g. no last private round

An empty field goes back to its automatic value (Ctrl-D does the same). A value out of range turns the field red and the error replaces the field description. F5 refuses to calculate while any field has a problem, and lists them.

14.15.4 Keys

Key	Action
Tab / Shift-Tab, ↓ / ↑	Next / previous field
PgDn / PgUp	Next / previous page
Enter	Open a pick-list
Ctrl-D	Clear the field back to its automatic value
F1	Glossary: type to filter it, ↑ / ↓ and PgUp / PgDn to scroll
F2	Review every value and its source
F3	Next level: Beginner → Intermediate → Advanced → Expert → Beginner. The form's bottom border says what it adds on this page
F4	Explain the live preview: what each number means and how it was reached, with today's values (↑ / ↓ scroll, Esc closes)
F5	Calculate and open the report
F9	Save the scenario
Esc / Ctrl-Q	Quit; asks first if there are unsaved changes

14.16 The live preview

The box on the right of the interview is the **live preview**. It values the company again after every keystroke, so you see at once what an answer does. It shows the deterministic base case: one forecast with every answer at its value, without the bear and bull scenarios or the Monte Carlo simulation (those need F5). Every number is per share, in the market's currency, and is the same number the report shows after F5. F4 opens the same explanations as below, with the values on your screen.

Line	What it is	Computed from
Fair value	What one share is worth after the IPO	DCF and Comps, blended
DCF	The value of the company's own forecast cash flows, per share	The whole forecast and the discount rates
Comps	The value at the price the market pays for similar companies, per share	Next year's revenue and the comparable multiple
Range	The price range published before investors order	Fair value and the IPO discount
Offer … at …×	The price IPO buyers pay, and how many times the offer is covered by orders	The book of orders at every price in the band
Verdict	PROCEED, THIN BOOK or POSTPONE	The book, and management's limits
notes	How many plausibility warnings the model raised	The checks behind the report's warnings

The numbers form a chain: the two methods give fair value, fair value gives the range, and the range is where the book is tested. An answer that moves fair value therefore moves everything below it; an answer about investors moves only the offer price, the coverage and the verdict. The examples below use the Tornfalk case (`pm-valuation --case tornfalk`): fair value 117.45, DCF 92.09, comparables 176.62, range 94.50–105.00, offer 105.00 at 11.9×, PROCEED.

14.16.1 Fair value

Meaning. The model's best estimate of what one share is worth once the IPO has settled: after the new money has come in and the fees have gone out. It is the anchor for everything else in the preview and in the IPO.

Calculation. A weighted average of the two valuation methods, with the *DCF weight* w on page 9 (70% by default):

$$\text{Fair value} = w \times \text{DCF} + (1 - w) \times \text{Comps}$$

Tornfalk: $0.7 \times 92.09 + 0.3 \times 176.62 = 64.46 + 52.99 = 117.45$.

Weight and importance. Fair value decides the price range, and with it where the offer price can land, so it is the most important number on the screen. Because the DCF counts 70%, the forecast and the discount rates dominate it. The report's tornado chart ranks the answers that move it most. For Tornfalk, moving each scenario driver alone from its pessimistic to its optimistic value changes fair value by:

Driver	Fair value, bear → bull	Swing
Headcount elasticity	73.27 → 154.48	81.22
Maximum share of SAM	83.13 → 152.09	68.96
Customer growth, year 1	93.02 → 123.84	30.82
Comparable EV / NTM revenue	102.09 → 132.80	30.71
Annual churn	104.47 → 125.99	21.52
Beta, year 6 on	107.68 → 126.37	18.70

Setting the DCF weight to 50% instead of 70% moves fair value from 117.45 to 134.36 without changing the company at all: the weight is a judgement about which method to trust, and it matters as much as many forecast inputs.

14.16.2 DCF

Meaning. The value per share of the cash the company itself is expected to generate, turned into today's money: the intrinsic value. It does not care what the stock market pays for similar companies today.

Calculation. In four steps (see The discounted cash flow, step by step):

1. Forecast the free cash flow to the firm (FCFF) for each of the ten years.
2. Add a terminal value for all the years after the forecast.
3. Discount each year at the stage-1 rate (years 1–5) and the stage-2 rate (year 6 on) and add them up: the enterprise value (EV).
4. Bridge to one share. The IPO money cancels out, because new investors pay for their shares exactly what they add; only the fees remain:

$$\text{DCF} = \frac{\text{EV} + \text{cash} - \text{debt} - f \times R - X}{S_{\text{pre}}}$$

where R is the primary raise, f the gross spread (the banks' fee), X the other offering expenses and S_{pre} the shares before the IPO. Tornfalk: EV 10,330.4 mkr at rates of 15.90% and 9.66%, plus 900 mkr of cash, minus 120 mkr of spread (3% of 4 mdr) and 60 mkr of other expenses, is 11,050.4 mkr; divided by 120 million shares, 92.09.

Weight and importance. 70% of fair value by default, and the most sensitive of the three values: 66.6% of Tornfalk's EV is terminal value, so anything that changes the long run changes it a lot. For Tornfalk:

Change	DCF	Fair value
None	92.09	117.45
Risk-free rate 3% → 4%	79.78	108.83
Terminal growth 2% → 2.5%	97.14	120.99
Annual churn 8% → 10%	82.56	110.78

14.16.3 Comps

Meaning. The value per share if the company were priced like comparable listed companies: the relative value, and what investors in the book will check first. It moves with market mood; the DCF does not.

Calculation. The comparable multiple (page 9; 10× for business software) times next year's revenue gives the enterprise value, which then crosses the same bridge as the DCF:

$$\text{Comps} = \frac{\text{multiple} \times \text{revenue}_1 + \text{cash} - \text{debt} - f \times R - X}{S_{\text{pre}}}$$

Tornfalk: $10 \times 2{,}047.5 = 20{,}475$ mkr, plus 900 cash, minus 180 of fees, is 21,195 mkr; divided by 120 million shares, 176.62.

Weight and importance. 30% of fair value by default. It depends on only two things: the multiple and next year's revenue. Each extra turn of the multiple adds next year's revenue divided by the shares, 17.06 per share for Tornfalk (11× gives 193.69), and 30% of that, 5.12, to fair value. Costs, margins and discount rates do not enter it at all, so a company with heavy costs can look much better on comparables than on its own cash flows. When the two methods differ by more than 50%, the report warns (V017). Tornfalk's DCF is 48% below its comparables value, just inside the limit, and the gap is itself worth explaining.

14.16.4 Range

Meaning. The price range printed in the prospectus before investors place their orders. It is set below fair value on purpose: the **IPO discount** pays investors for buying an untested stock and leaves room for the price to rise on the first day.

Calculation. The midpoint is fair value less the IPO discount d (page 8, 15% by default); the range is 5% either side, rounded outwards to "nice" prices:

$$\text{mid} = \text{Fair value} \times (1 - d), \qquad \text{low} = \lfloor 0.95 \times \text{mid} \rfloor_{\text{step}}, \qquad \text{high} = \lceil 1.05 \times \text{mid} \rceil_{\text{step}}$$

The step depends on the price: 0.01 below 2, 0.10 below 10, 0.50 below 100, 5.00 below 1,000, and so on. Tornfalk: $117.45 \times 0.85 = 99.83$; $0.95 \times 99.83 = 94.84$ rounds down to 94.50, and $1.05 \times 99.83 = 104.83$ rounds up to 105.00.

If management has a **minimum market cap** (page 11), its price, the floor, is (minimum market cap $- R$)$/S_{\text{pre}}$. A range below the floor is moved up to start there, keeping its width, and the preview adds "(moved by the floor)". If the moved range leaves the banks less than the minimum IPO discount (5%) below fair value, the IPO is postponed: the Halvard case.

Weight and importance. The range is fixed by fair value and the discount; nothing about investors enters it. It matters because it limits the offer price: in Sweden the top of the range is the maximum price. Note how the step widens at 100: a 10% discount gives Tornfalk a midpoint of 105.71 and a range of 100.00–115.00, 15 wide instead of 10.50.

14.16.5 Offer price and coverage

Meaning. "Offer 105.00 at 11.9×" is the price IPO buyers pay, and the **coverage** at that price: orders for 11.9 times the value of the shares on offer. Coverage above 1 means investors receive less than they asked for, and the unfilled ones buy on the first day.

Calculation. The model builds the book: the demand at every price on the step grid, from the bottom of the band, 80% of the range's low (or the management floor, if higher), to the maximum price (the top of the range in Sweden, 20% above it in the US). At each price P:

$$\text{coverage}(P) = \frac{\text{institutional}(P) + \text{retail}(P) + \text{cornerstone}}{R + \text{secondary shares} \times P}$$

Institutional demand is the number of institutions × their average order × the interest level (0.3× to 2.2×) × small factors for the lock-up, the index prospects, dual-class shares and hype, all × (fair value/P)^3, so it falls quickly as the price rises above fair value. Retail demand is the applicants × their average application × their interest × $(1 + 0.25 \times \text{hype})$, and falls more slowly with the price. The deal is priced at the **highest price whose coverage is at least the target** (3× by default, page 9).

Tornfalk at 105.00: 60 institutions × 200 mkr × 2.2 (very high interest) × 1.08 (index prospects) × 1.15 (hype 5) × $(117.45/105)^3 = 1.40$ gives 45,888 mkr; 20,000 retail applicants × 25,000 SEK × 1.5 × 2.25 × $(117.45/105)^{0.6}$ gives 1,805 mkr. Together 47,693 mkr against a 4 mdr offer: 11.92×. Every price in the band is covered more than 3 times, so the deal is priced at the highest, the maximum price 105.00.

Weight and importance. The offer price is what the company actually receives per share, so it decides the money raised, the dilution and the market capitalisation at listing. It depends on fair value through the range, and on the investor answers through the book. Investor interest and hype move coverage a lot but the offer price little, because the price cannot leave the band. Coverage is not value: a hot book (12×) says that demand would have supported a higher price, and the money left on the table is the cost of the discount.

14.16.6 Verdict

Meaning and calculation.

- **PROCEED**: some price in the band is covered at least the target number of times; the deal is priced as above.
- **THIN BOOK**: no price reaches the target, but the bottom of the band is covered at least once; the deal is priced there, with a real risk of trading below its offer price.
- **POSTPONE**: the book is not covered even once at the bottom of the band, or management's floor leaves the banks less than the minimum discount. The preview shows the reason under F4.

Weight and importance. The only line about *whether* the IPO happens; every other line is about *how much*. A POSTPONE has no offer price, and `--list` refuses to list it.

14.16.7 Notes

Meaning and calculation. " 2 notes" counts the warnings and observations from the model's plausibility checks (V001–V021): for example a year-10 EBIT margin outside the sector's range, most of the value lying beyond the horizon, or the DCF

and comparables differing by more than 50%. F2 lists them under the values; the report explains each one. The line is absent when there are none.

Weight and importance. Notes do not change any number. They tell you which numbers to trust less, and they make good questions for a write-up: why is the margin so high, why do the two methods disagree?

14.17 The report

F5 opens the report in a scrollable viewer with a section index on the left.

Key	Action
↑ / ↓, PgUp / PgDn, Space, Home / End	Scroll
Tab / Shift-Tab	Next / previous section
`b`	**Back to the interview, with every answer kept**
`c`	Compare with the previous calculation: headline numbers, changes, and the inputs that changed
`e`	Export the report as Markdown
`p`	Write the report as a printable PDF (see below)
`q`	Quit

`b`, change one answer, F5, `c` is the what-if loop the exercise is built on.

The sections, in order:

Section	Content
Verdict	PROCEED, PROCEED (THIN BOOK) or POSTPONE, the headline numbers and why
Prospectus cover, or S-1 cover	Issuer, ticker, industry code (ICB in Sweden, SIC in the US), the authority and the listing venue; for the US also emerging-growth and smaller-reporting status
Assumptions	Every input with its source
Market and customers; Unit economics; Headcount	The ten-year operating forecast
Income statement; Taxes, reinvestment and FCFF	From revenue to free cash flow
Discount rates; DCF	The rate build-up per stage, discount factors, terminal value
Bridge and fair value	From enterprise value to value per share, blended with comparables
Scenarios, tornado and sensitivity	Bear / base / bull, the drivers that matter most, two grids
Monte Carlo	The distribution of fair value, and the chance the offer price is too high
Pricing	Range, management floor, the book at every price, the chosen price, allocation, first-day pop
Capitalisation and dilution	Shares before and after, dilution to new investors
Lock-ups, free float and index	The index rulebook at the offer price, and the lock-up overhang
Risk factors	The model's warnings, phrased as a prospectus would
Next step	The `pm-new-symbol` command, and the index command for later
What this model leaves out	Its limits

Scenarios appear with `--mode deterministic` or `both`, Monte Carlo with `montecarlo` or `both`; the section numbers close up when one is left out.

14.17.1 How the offer price is chosen

The range is fair value less the IPO discount (15% by default), rounded to "nice" prices. Management's minimum market cap can move it up. The book is then built at every price from 20% below the range up to the **maximum price**: the top of the range in Sweden, 20% above it with `--market us`. The deal is priced at the **highest price that is covered at least 3×**. If no price reaches 3×, it is priced at the bottom of the band on a thin book. If even that is less than 1× covered, or the management floor leaves the bankers less than the minimum discount to fair value, the IPO is **postponed**.

14.17.2 The PDF report

`--pdf FILE`, or `p` in the report viewer, writes the report as a document for print: A4 by default, `--paper letter` for US Letter.

```
pm-valuation --case tornfalk --no-tui --pdf tornfalk.pdf
```

It holds the same numbers as the terminal report, arranged as a document:

Part	Content
Cover	Company, verdict and headline figures
Contents	Every chapter and section, with page numbers
Executive summary	The verdict, the headline figures, why, what the model flags, the principal risks and the next step
Chapters 1–6	The company and its offering; operating forecast; valuation; uncertainty; the offering; risks and next steps. Each chapter and section opens with prose on what it shows and how to read it
Figures	Revenue and free cash flow over the forecast; the tornado; the Monte Carlo distribution against the offer price; the book's coverage at each price
Appendix A	Every assumption with its source
Appendix B	The glossary

Every page after the cover carries the company in its header, and the page number and the disclaimer in its footer. The document has PDF bookmarks for each chapter and section. The comparison with a previous run (`c`) is not part of it.

The PDF uses the Vera font that ships with ReportLab, embedded in the file. Vera has no Greek letters or check marks, so the PDF spells those few symbols out: "Beta" for , "correlation" for , "met" and "not met" for and .

14.18 Scenario files

A scenario file holds only what you typed. Defaults are recomputed when it is loaded, so a change to a sector preset shows up.

```
pm_valuation: 1
company:   {name: Tornfalk Security AB, ticker: TORN, sector: b2b_saas, market: se}
customers: {last_fy_revenue: 1.4bn, now: 2600}
capital:   {cash: 900m}
offering:  {shares_pre: 120m, raise: 4bn}
investors: {inst_interest: very_high, retail_interest: high, hype: 5, n_institutions: 60}
```

Amounts are in the currency of the file's market: here SEK.

Values are written the way you type them in the interview. Loading is strict: an unknown field or format version is an error, so a typo cannot silently fall back to a default.

Command	Effect
F9, or `--save FILE`	Write your answers
`--save FILE --with-defaults`	Write every value, each commented with its source: a fully specified case to hand out
`--load FILE`	Start from a file
`--case NAME`	Start from a classroom case shipped with the tool

The classroom cases are `tornfalk` (Tornfalk Security AB, a hot deal) and `halvard` (Halvard Robotics AB, a hard one), both Swedish. The IPO Valuation training chapter uses them.

14.19 Listing the result

`--list` passes the priced IPO to `pm-new-symbol`, in the same process and with the same arguments the Next step section prints:

```
pm-opctl-cli stop
pm-valuation --load tornfalk.yaml --no-tui --list
pm-opctl-cli start
```

- `--list` needs `--no-tui`. Explore in the interview, save with F9, then list from the saved file, so the listed price is always reproducible.
- `--config PATH` is passed on as `pm-new-symbol --config`: the authored YAML to edit. Without it, the deployed configuration's source is edited and redeployed.
- Every `pm-new-symbol` guard applies: the exchange must be stopped, and no saved state may exist for the symbol (see Why it refused).
- A postponed IPO has nothing to list: the report is printed, and the command exits with status 1.
- The seed quote, collar and other listing options are `pm-new-symbol`'s defaults. To choose them, run the printed command yourself with the extra options.

When the index verdict is positive, Next step also prints the `pm-index-admin-cli` command for the first trading day on which the stock can join the index. That is a later step, not part of listing.

14.20 Exit status

Status	Meaning
0	Success, including a POSTPONE verdict
1	Invalid answers in `--no-tui` mode, an unreadable scenario file, or a failed `--list`
2	Usage error, e.g. `--list` without `--no-tui`

14.21 Options

Option	Default	Meaning
`--load FILE`	none	Scenario file to start from
`--case NAME`	none	Classroom case to start from (`tornfalk`, `halvard`)
`--market M`	the scenario's, else `se`	IPO framework and defaults: `se` (Sweden) or `us` (United States)
`--no-tui`	off	Print the report instead of interviewing

Option	Default	Meaning
`--level LEVEL`	`beginner`	Interview detail: `beginner`, `intermediate`, `advanced` or `expert` (see Levels). TUI only — combining it with `--no-tui` is a usage error (exit 2), since there is no interview to choose fields for
`--mode MODE`	the scenario's, else `both`	`deterministic`, `montecarlo` or `both`
`--draws N`	`10000`	Monte Carlo draws
`--seed S`	`42`	Monte Carlo seed
`--save FILE`	none	Write the scenario
`--with-defaults`	off	With `--save`: write every value with its source
`--export FILE`	none	Write the report as Markdown
`--pdf FILE`	none	Write the report as a printable PDF
`--paper SIZE`	`a4`	PDF page size: `a4` or `letter`
`--presets FILE`	bundled	Alternative sector presets
`--list`	off	With `--no-tui`: list the priced IPO with `pm-new-symbol`
`--config PATH`	the deployed source	With `--list`: the engine YAML to edit

Terminal width

The interview is laid out for a terminal at least 120 columns wide; in a narrower one the automatic-value hints are cut off first. Reports printed with `--no-tui` are always 100 columns wide, so they read the same in a terminal, a pipe or a file; a narrower terminal wraps their lines. A table wider than 100 columns is split into parts that each repeat its first column. `--export` writes every table in one piece.

14.22 Field reference

This part explains every field in the interview, page by page, for readers who are new to finance. Each page opens with what it is about and why it matters to the valuation, followed by a table of all its fields.

- **Field** is the name shown in the interview.
- **Level** is the first level that shows the field (see Levels).
- **Default** is the automatic value used when you leave the field empty, and where it comes from (see Automatic values and their source). Values follow the default Swedish market (`se`). Where the American market (`us`) differs, its value follows the slash. "Preset" values are shown for the default `b2b_saas` sector; the other sectors are in the design document.
- **Explanation** says what the field means, how the model uses it, and what happens when you change it.

Money amounts are in the market's currency: SEK for `se`, USD for `us`. Swedish amounts are written as the interview shows them: `mkr` for millions of kronor and `mdr` for *miljarder*, thousands of millions. The sector presets are written in dollars and converted at 10 SEK per dollar; salaries and revenue per employee are then set 30% lower for Sweden.

14.22.1 Page 1 — Company

The first page is the **cover of the offering document**: who is going public, where, and under what name. Almost everything here is for display, as it would appear on the first page of a prospectus or an S-1. Three fields do change the valuation: the **sector preset**, which supplies most of the other defaults; the **market**, which supplies the country's rates, tax, fees and currency; and **dual-class shares**, which lowers institutional demand. Choose the market and the sector first, because the other pages fill in from them.

Field	Level	Default	Explanation
Company name	Beginner	Market: Newco AB / Newco Inc.	The legal name of the company going public, as it will stand on the cover of the prospectus (or the S-1). A Swedish limited company ends in **AB**, *aktiebolag* ("share company"), the counterpart of *Inc.*; a listed one is an **AB (publ)**, a public company that may offer shares to anyone. The name is used in the report's title and headers and in the proposed ticker. It does not affect the value.
Proposed ticker	Intermediate	Derived from the name: NEWC	The short symbol the stock will trade under on the exchange, and the name `pm-new-symbol` lists it with. It must be 1–8 characters of A–Z, 0–9, . or _. The default drops suffixes such as AB or Inc., then takes the first four letters of a one-word name, or three letters of the first word and one of the second. Traders see only this symbol all day, so real companies choose it with care.
Sector preset	Beginner	`b2b_saas`	The company's business model. A **preset** is a coherent set of starting values for one kind of business: market size, price per customer, how many customers leave each year, costs, capital needs, risk (beta and the execution premium) and how the stock market values similar companies. There are 18, from software and consulting to retail, heavy industry and power production; Enter opens the list, grouped by industry with a line on each. See Sector presets for what each one is and which companies have none. Every value marked "preset" changes when you change the sector, unless you typed it yourself.

Field	Level	Default	Explanation
Industry code	Advanced	Derived from the sector and market: ICB 10101015 Software / SIC 7372	The official industry classification on the cover of the offering document. In Sweden it is the **ICB** subsector (Industry Classification Benchmark), the classification Nasdaq Stockholm uses: an eight-digit code whose first two digits are the industry (10 is Technology). In the US it is the four-digit **SIC** code (Standard Industrial Classification) the SEC files companies under (7372 is prepackaged software). Display only.
Incorporated in	Advanced	Market: Sweden / Delaware	The country (or US state) where the company is registered as a legal person, and whose company law governs it. Swedish IPOs are usually of Swedish ABs; most US companies are incorporated in Delaware for its well-tested company law. Display only.
Currency	Advanced	Market: SEK / USD	The currency of every amount you type and every amount in the report. It follows the market. Large numbers can be typed with suffixes: `m` for million, `bn` for billion, and the Swedish `mkr` (miljoner kronor), `md` or `mdr` (miljarder). A billion is a thousand million: a Swedish *miljard*, not a *biljon*. The model's arithmetic does not depend on the currency.
Fiscal year end	Advanced	December 31	The last day of the company's financial year, on which its annual accounts are closed. "Year 0" in the forecast is the last completed financial year; year 1 is the next. Most Swedish and US companies use the calendar year. Display only.

Field	Level	Default	Explanation
Dual-class shares	Intermediate	no	Whether the company has two classes of shares with different voting power, for example A shares with ten votes each and B shares with one, so that founders keep control while selling most of the capital. This is common in Sweden. Institutional investors (pension, insurance and investment funds) see a governance risk, because they cannot outvote the founders, so the model lowers their orders by 5%. Many stock-market indices also refuse such companies unless the exchange agrees (page 10).
Lead underwriter	Advanced	Market: Fiktiva Banken AB / Fictive & Co.	The investment bank that leads the IPO, the *bookrunner* or *global coordinator*. It advises on the price, writes the prospectus with the company's lawyers, markets the shares to investors, collects their orders and allocates the shares. Display only.
Use of proceeds	Advanced	"General corporate purposes, including working capital, R&D and sales expansion"	What the company will do with the money it raises, a section every prospectus must have. Investors read it to judge whether the money will create value (new products, expansion) or merely replace existing owners' money. Display only.

Market	Beginner	`se`	The IPO framework, Swedish (`se`) or American (`us`). It decides the offering document (prospectus approved by Finansinspektionen, or an S-1 filed with the SEC), the currency, the country defaults for interest rates, tax, inflation and bank fees, and whether the deal may be priced above its price range. See Sweden or the United States. `--market` on the command line overrides it.

14.22.2 Page 2 — Market

A company can only grow as large as its market allows. This page describes that market in three steps, from the whole industry down to what this company can realistically win. The **total addressable market** (TAM) is all spending on this kind of product. The **serviceable addressable market** (SAM) is the part this product can serve. The **maximum share** of the SAM is the most the company can win against its rivals. Together they set the ceiling that the company's customer growth slows toward, so they have a large effect on the value: they decide how big the company can ever get.

Field	Level	Default	Explanation
Total addressable market	Beginner	Preset: 400 mdr / 40 bn USD	The total yearly spending on this kind of product by every possible customer, worldwide, today. It is the size of the whole industry, not the company's revenue. It can be estimated *top-down*, from industry reports, or *bottom-up*, as the number of possible customers × the yearly price. Start-ups are tempted to quote a huge TAM; it matters only through the SAM and the share below.
TAM growth, year 1	Intermediate	Preset: 12%	How fast the whole market grows next year. The rate then falls in a straight line to the terminal growth rate (page 7, the growth assumed for ever after the forecast) by the last forecast year, because no market can outgrow the economy for ever. A fast-growing market lifts the ceiling on the company's size year by year.

Field	Level	Default	Explanation
SAM share of TAM	Intermediate	Preset: 25%	The share of the total market that this product can actually serve, given its countries, languages, customer segments and price. A product sold only in Europe to mid-sized companies might serve 20–30% of its TAM. With the defaults, the SAM is 25% of 400 mdr = 100 mdr.
Market structure	Beginner	competitive	How contested the market is, which sets the largest share of the SAM the company can realistically win: `fragmented` 5% (many small rivals), `competitive` 8%, `oligopoly` 15% (a few large players) or `dominant` 30% (close to a monopoly). With the defaults, the most revenue the company can reach today is 8% of 100 mdr = 8 mdr, against 900 mkr of revenue last year.
Maximum share of SAM	Expert	Derived from the market structure: 8%	The same ceiling, typed directly instead of through the market structure. Customer growth follows an **S-curve**: fast while the company is small compared with its ceiling, then slowing as it gets closer, like a new product spreading through a population. It is one of the ten scenario drivers (page 12), and usually one of the inputs with the largest effect on value.

14.22.3 Page 3 — Customers & pricing

Revenue is the number of customers × the average revenue per customer. This page gives the starting point, last year's revenue and customers, and the three forces that move them: how fast new customers are won, how many are lost each year, and how the price per customer changes. These inputs drive the whole operating forecast on the pages that follow. Revenue and customers are linked: if you give only one of them, the model derives the other from the price per customer.

Field	Level	Default	Explanation
Last FY revenue	Beginner	Preset: 900 mkr / 90 m USD	Revenue (sales) in the last completed financial year, *year 0*: the starting point of the forecast. It is the most important fact about a young company, and the one investors check first. If you give customers but not revenue, it is derived as 0.85 × customers × ARPU (see the next field for why 0.85).
Customers now	Intermediate	Derived: 1,765	Paying customers at the end of year 0. What a "customer" is depends on the sector: a business account, a subscriber, a buyer, a shop. The default is revenue ÷ (0.85 × ARPU): 900 mkr ÷ (0.85 × 600,000) 1,765. The 0.85 allows for growth during the year: customers who joined during the year paid for only part of it, so the year earned less than the year-end customer count × a full year's price. Compared with the ceiling from page 2, the customer count shows how far along its S-curve the company already is.
ARPU per year	Intermediate	Preset: 600,000 SEK / 60,000 USD	**Average revenue per user**: what one customer pays per year on average. For business software it is the yearly contract value; for a marketplace it is only the fee the platform keeps (the *take rate*), not the value of what was bought. Revenue = customers × ARPU. A high ARPU with few customers (enterprise software) and a low ARPU with millions of customers (a consumer app) can give the same revenue with very different costs.

Field	Level	Default	Explanation
ARPU growth, year 1	Advanced	5%	How much the average customer pays more next year, from price increases and from selling more to existing customers (*upselling*). It fades to general inflation (page 5) by the end of the forecast. Because the market (page 2) is measured in money, a faster-rising price also means the market holds fewer customers. A scenario driver: bear 3%, bull 7%.
Annual churn	Intermediate	Preset: 8%	The share of customers lost each year. At 8% the average customer stays $1 \div 0.08 = 12.5$ years; at 35%, typical for consumer apps, less than three. Churn decides a customer's **lifetime value**, the gross profit it brings before it leaves: ARPU × gross margin ÷ churn (the gross margin is the share of revenue left after the direct cost of serving customers). Halving churn roughly doubles what each customer is worth. Business software often loses 5–15% of customers a year. A scenario driver.
Customer growth, year 1	Beginner	Preset: 50%	Net growth in the number of customers next year: customers won minus customers lost, as a share of today's customers. From year 2, growth slows by itself as the company approaches its ceiling (the S-curve). It is a **scenario driver**: one of the ten inputs the report varies between a pessimistic (*bear*) and an optimistic (*bull*) value, here 30% and 65%, to show how uncertain the valuation is.

14.22.4 Page 4 — People

For a young technology company, staff are the largest cost by far, so how many people it needs as it grows decides most of its future profit. The key idea is **operating leverage**: if revenue grows faster than the number of employees, each employee brings in more revenue every year, and the profit margin widens. This page sets today's headcount, what each employee costs,

and how hiring follows growth.

Field	Level	Default	Explanation
Headcount	Intermediate	Derived: 643 / 450	The number of employees today. The default divides last year's revenue by the sector's typical revenue per employee: 900 mkr ÷ 1.4 mkr = 643. A company with many more staff than that for its revenue is investing heavily ahead of growth, or is inefficient.
Loaded cost per employee	Intermediate	Preset: 1.05 mkr / 150,000 USD	The full yearly cost of one employee: salary plus social charges and pension (in Sweden, employer contributions of about a third of the salary), benefits, office space and equipment. It is often 1.3–1.5 times the salary itself. Headcount × loaded cost is the staff cost, which the model splits into the four departments below.
Wage inflation	Advanced	3.5%	How much the loaded cost per employee rises each year. It is usually a little above general inflation, because wages also rise with productivity and because technology staff are in demand.
Headcount growth	Advanced	`follow_revenue`	How the number of employees grows. `follow_revenue` ties hiring to revenue growth through the elasticity below, never slower than the floor: when revenue grows 50%, staff grow 30% at the default elasticity. `explicit` ignores revenue and moves in a straight line from the year-1 hiring rate to the floor in the last forecast year; use it when you have a hiring plan.

Field	Level	Default	Explanation
Headcount elasticity	Intermediate	0.6	In `follow_revenue` mode: the percentage growth in staff for each percentage of revenue growth. At 0.6, revenue up 50% needs 30% more staff. Below 1, revenue per employee rises as the company grows, which is operating leverage; at 1, margins stay where they are; above 1, the company gets less efficient as it grows. Because staff are most of the cost, this input often has the largest single effect on value in the report's **tornado** chart, which ranks inputs by how far changing each one alone moves the value. A scenario driver: bear 0.7, bull 0.5.
Headcount growth, year 1	Advanced	25%	In `explicit` mode only: the hiring rate next year, as a share of today's staff.
Headcount growth floor	Advanced	4%	The slowest staff growth in any year in `follow_revenue` mode, and the rate reached in the last forecast year in `explicit` mode. Even a mature company keeps hiring a little, to replace leavers and support new products.
Staff share: R&D	Expert	Preset: 38%	The share of employees in **research and development**: engineers, designers and product managers who build the product. Their cost is an operating expense. The four shares must add up to 100%.
Staff share: sales & marketing	Expert	Preset: 30%	The share of employees who win customers: sales people, marketing, partnerships. Their cost is part of the total cost of acquiring customers, together with the paid acquisition cost on page 5.

Field	Level	Default	Explanation
Staff share: G&A	Expert	Preset: 14%	The share in **general and administrative** work: finance, legal, HR and management. G&A usually shrinks as a share of revenue as the company grows, one more source of operating leverage.
Staff share: operations	Expert	Preset: 18%	The share in support and service operations: customer support, onboarding, running the platform. Their cost is part of the **cost of revenue**, the direct cost of delivering the product, so it lowers the gross margin rather than the operating expenses.

14.22.5 Page 5 — Costs

Apart from staff, a company spends money on delivering its product, on winning customers, on development tools and offices, and, once listed, on being a public company. This page sets those costs. Some are fixed amounts that rise with inflation, some grow with the number of customers, and some are a share of revenue. Together with page 4 they turn the revenue forecast into operating profit: **EBIT**, earnings before interest and taxes.

Field	Level	Default	Explanation
Infrastructure, fixed	Advanced	Preset: 60 mkr / 6 m USD	Yearly hosting and platform costs that do not depend on the number of customers: the base cost of running the service at all. It rises with inflation. It is part of the cost of revenue. Because it is fixed, it weighs heavily while the company is small and becomes a small share of revenue as it grows.
Infrastructure per customer	Advanced	Preset: 40,000 SEK / 4,000 USD	The yearly cost of serving one more customer: computing, storage, support tools, and for hardware the cost of the delivered equipment. It rises with inflation. Compared with the ARPU (600,000 SEK) it shows how much of each customer's payment is left as gross profit.

Field	Level	Default	Explanation
Other cost of revenue	Advanced	Preset: 6%	Other direct costs that grow with revenue, as a share of it: card and payment fees, licences for third-party software built into the product, app-store fees. Revenue minus all costs of revenue is the **gross profit**; as a share of revenue, the gross margin. Software companies often have gross margins of 70–80%.
Paid acquisition cost per customer	Intermediate	Derived from the preset: 300,000 SEK / 30,000 USD	The marketing money spent to win one new customer: advertising, campaigns, events, referral fees. Sales staff are counted in headcount, not here. The default is the sector's multiple of the ARPU (0.5 × 600,000 for `b2b_saas`). A customer is worth winning only if its lifetime value (page 3) is several times what it cost to win; a common rule of thumb is at least 3 times. A scenario driver: bear 390,000, bull 240,000.
Acquisition cost growth	Advanced	3%	How fast the cost of winning a customer rises each year. The easiest customers are won first; later ones need more persuasion, and rivals bid up advertising prices. This is one reason growth becomes more expensive as a company matures.
R&D, non-staff	Advanced	4%	Development spending other than salaries, as a share of revenue: developer tools, software licences, test equipment, cloud capacity for development.
G&A, non-staff	Advanced	3%	Administrative spending other than salaries, as a share of revenue: rent, insurance, accountants and lawyers.

Field	Level	Default	Explanation
Public-company cost	Advanced	30 mkr / 3 m USD	What it costs each year to be listed: the audit, insurance for directors, investor relations, exchange fees, quarterly reports and compliance with market rules. A private company does not pay these, so they are a real cost of going public. They grow with inflation and are part of G&A.
Stock-based compensation	Intermediate	12%	Shares and share options given to employees as pay, as a share of staff cost. An **option** is the right to buy a share later at a fixed price, which is valuable if the share price rises. No cash is paid, but it is a real cost: new shares **dilute** the existing owners, who then own a smaller slice of the company. The model counts it as an expense, like salaries, and does not add it back.
General inflation	Advanced	Market: 2% / 2.5%	The general rise in prices each year. Fixed costs rise with it, and ARPU growth fades to it by the end of the forecast. The Riksbank (Sweden's central bank) and the US Federal Reserve both aim for inflation of about 2% a year.

14.22.6 Page 6 — Capital & tax

Profit is not the same as cash. A company must also buy equipment, fund the money tied up in running its business, and pay tax, and the valuation is built on the cash that is left, the **free cash flow**. This page sets those items, and the cash and debt the company has today, which are added to or subtracted from the value of the business to find the value of its shares.

Field	Level	Default	Explanation
Capex	Advanced	Preset: 3%	**Capital expenditure**: cash spent on long-lived equipment such as servers, hardware and office fit-outs, as a share of revenue. The income statement spreads this cost over the equipment's life as depreciation, but the cash leaves the company at once, so capex lowers free cash flow in the year it is spent. Hardware companies need much more capex than software companies.
Useful life	Advanced	Preset: 4 years	The number of years over which capex is depreciated, in equal amounts (*straight line*): equipment bought for 4 m with a four-year life costs 1 m a year in the income statement. The preset's life fits its assets: 4 years for servers and computers, 10 for factory machinery, 25 for wind turbines. Depreciation is not a payment; it spreads cash already spent. A longer life raises reported profit in the early years but does not change the cash, so it hardly changes the value.
Opening PP&E	Advanced	Derived: 54 mkr / 5.4 m USD	**Property, plant and equipment**: the book value of the company's equipment today, what it cost less the depreciation so far. Each year capex adds to it and depreciation (PP&E ÷ useful life) reduces it. The default is the level the equipment settles at: capex × useful life ÷ 2 = 3% × 900 mkr × 4 ÷ 2 = 54 mkr.

Field	Level	Default	Explanation
Net working capital	Advanced	Preset: −5%	The money tied up in day-to-day business, as a share of revenue: what customers owe plus inventory, minus what the company owes suppliers and what customers have paid in advance. When it is positive, as for a hardware maker with stock and slow-paying customers, growth absorbs cash. When it is negative, as for software paid a year in advance, growth releases cash, because customers pay before the company delivers.
Tax rate	Intermediate	Market: 20.6% / 25%	Corporate income tax on operating profit (EBIT). Sweden's rate is 20.6%; the US rate of about 25% combines federal and state taxes. A young company pays no tax while it makes losses, and its losses then shelter its first profits (next field). After the forecast the company pays the full rate, which matters most for the terminal value.
Tax losses carried forward	Intermediate	0	Losses from past years that can be set against future profits, so that no tax is paid until they are used up. Young companies often have large ones, and ignoring them undervalues the company. The forecast's own early losses are added to this amount automatically.
Cash	Beginner	0	Cash in the bank before the IPO. The discounted cash flow values the business itself, its **enterprise value**. Cash is not needed to run the business, so it is added on top to reach the value of the shares: equity value = enterprise value + cash − debt. The money raised in the IPO is handled separately (page 8).

Field	Level	Default	Explanation
Debt	Beginner	0	Loans and bonds before the IPO. Lenders are paid before shareholders, so debt is subtracted from the enterprise value to reach the value of the shares. Like a house bought with a mortgage: the house is the enterprise value, the owner's share of it is the equity value.

14.22.7 Page 7 — Discount rates

Money received in the future is worth less than money today, both because today's money could earn interest and because the future is uncertain. The **discount rate** is the yearly return investors require, and it turns each future cash flow into today's money: a cash flow in year t is divided by $(1 + \text{rate})\hat{}t$. This page builds that rate from its parts. It starts from the **risk-free rate**, what a government bond pays. It adds a premium for the risk of shares, scaled by the company's **beta**, and premia for its small size and for the risk that its plan fails. The model uses two rates: a higher one for the risky first five years (stage 1) and a lower one once the company has become established (stage 2). With the Swedish defaults, stage 1 is $3.0 + 1.5 \times 5.6 + 1.5 + 3.0 = 15.9\%$ and stage 2 is $3.0 + 1.1 \times 5.6 + 0.5 = 9.66\%$. See Risk and the required return for the ideas behind them.

Field	Level	Default	Explanation
Risk-free rate	Beginner	Market: 3.0% / 4.25%	The return on an investment with no risk of loss, measured by the yield on a long-dated government bond: the Swedish or US 10-year bond. Every discount rate starts here, so a higher risk-free rate lowers every value. The default is a recent round figure, not a quote: check today's yield and type it in.
Equity risk premium	Intermediate	Market: 5.6% / 5.0%	The extra yearly return investors demand for owning shares in general rather than government bonds, because shares can fall and bonds (held to the end) cannot. It cannot be observed directly; surveys of professionals put it at about 5–6% (5.6% for Sweden in PwC's 2026 study). It is multiplied by the beta, so it matters most for high-beta companies.

Field	Level	Default	Explanation
Beta, years 1-5	Intermediate	Preset: 1.5	**Beta** measures how strongly a stock moves with the stock market as a whole: at 1 it moves with the market, at 1.5 it rises and falls 50% more. Investors cannot avoid this kind of risk by owning many shares, so they demand to be paid for it: cost of equity = risk-free rate + beta × equity risk premium, the **capital asset pricing model** (CAPM). Young growth companies have high betas. This is the beta for stage 1.
Beta, year 6 on	Intermediate	Preset: 1.1	The beta once the company is established. Mature companies behave more like the market as a whole, so their beta drifts toward 1. It also applies to the terminal value, the value of every year after the forecast, so small changes here move the value a lot. A scenario driver: bear 1.4, bull 0.9.
Size premium, years 1-5	Advanced	1.5%	Extra yearly return investors demand from small companies, on top of the CAPM. Small companies fail more often, and their shares are harder to sell quickly without moving the price, so investors want to be paid more for holding them. 1–3% is a common range. Added to the stage-1 rate.
Size premium, year 6 on	Advanced	0.5%	The size premium from year 6 onwards. By then the company is larger and its shares easier to trade, so the premium is smaller. It is part of the stage-2 rate, which also discounts the terminal value, so small changes here move value noticeably.

Field	Level	Default	Explanation
Execution premium	Advanced	Preset: 3.0%	Extra yearly return in stage 1 only, for the risk that the plan simply does not happen: products are late, customers do not come, key people leave. It is the main reason the first five years have their own, higher rate. Once the company is established, it falls away. The preset sets it: 2% for established businesses such as construction or retail, 5% for medical devices awaiting approval, 6% for a game studio living on hits. A scenario driver: bear + 3 points, bull − 2 points (6% and 1% at 3%).
Target debt ratio D/V	Expert	0%	The share of the company's capital (debt plus equity, $D + E = V$) that it plans to finance with debt. With debt, the discount rate becomes a **weighted average cost of capital** (WACC): the cost of equity and the after-tax cost of debt, weighted by their shares. Debt is cheaper than equity, so some debt lowers the rate. Most growth companies have none.
Cost of debt	Expert	Derived: risk-free + 3% = 6.0% / 7.25%	The interest rate the company pays on its debt, before tax. Interest is deducted before tax is charged, so the model uses it after tax: cost × (1 − tax rate). Used only when the target debt ratio is above 0.
Stage-1 years	Expert	5	How many years are discounted at the higher stage-1 rate; the following years use the stage-2 rate. Each year's discount factor multiplies all the earlier years' rates together, so later years still carry the stage-1 discount of the early years.

Field	Level	Default	Explanation
Forecast horizon	Expert	10	How many years are forecast one by one before the terminal value takes over. It should be long enough for growth to settle close to the terminal growth rate; the model warns if revenue is still growing much faster than that in the last year.
Terminal growth	Intermediate	Market: 2% / 2.5%	The yearly growth of the cash flows for ever after the forecast. The **terminal value** is the value, at the end of the forecast, of all those later years, and for a young company it is often more than half of the total value. Terminal growth must stay well below the stage-2 rate, and at or below the growth of the whole economy: no company can outgrow the economy for ever. A scenario driver: bear 1.5%, bull 2.5% (2% / 3% for `us`).
RONIC spread	Expert	2%	**RONIC** is the return on new invested capital: the yearly profit on money the company reinvests to keep growing after the forecast. The spread is how far it is above the stage-2 rate. Growth needs reinvestment; if new money earns only what investors require (a spread of 0), growth adds no value at all. A larger spread means growth needs less reinvestment, which raises the terminal value.
Override rate, years 1-5	Expert	none (built up)	Type the stage-1 discount rate directly instead of building it up from the fields above. Useful to test a rate you have from elsewhere, for example from a bank's research report. The build-up is then shown as not used.

Field	Level	Default	Explanation
Override rate, year 6 on	Expert	none (built up)	Type the stage-2 discount rate directly. The terminal value is also discounted at this rate, so it must be well above the terminal growth rate; otherwise the terminal value explodes, and the model reports that it cannot value the company.
Mid-year convention	Expert	no	Discount each year's cash flow as if it arrived in the middle of the year rather than at its end, because cash comes in all year round. It raises the value by about half a year of discounting. Off by default, so that the report's tables can be checked by hand.

14.22.8 Page 8 — Offering

The offering is the sale itself: how many shares exist before the IPO, how much new money the company raises, what the banks and advisers are paid, and how far below fair value the shares are offered. It also sets the lock-ups that keep existing owners from selling right after the listing. None of these change what the business is worth, but they decide the price per share, how much of the company the new investors get, and how much it costs to raise the money.

Field	Level	Default	Explanation
Pre-IPO shares (fully diluted)	Intermediate	Derived: 139,000,000 / 69,000,000	The number of shares before the IPO, **fully diluted**: employee options and convertible securities are counted as if they had already become shares, because they will. Before listing, companies *split* their shares (turn each one into several) so that the price per share lands at a usual level; the default does the same, aiming at about 100 SEK (20 USD) per share. The share count changes the price of each share, not the value of the company: a pizza cut into more slices is not a bigger pizza.

Field	Level	Default	Explanation
Primary raise (gross)	Beginner	Derived: 2.75 mdr / 275 m USD	The new money the company raises by issuing new shares, before fees. At a fair price, raising money neither creates nor destroys value for the existing owners: the company gets cash worth exactly what the new shares are worth. Only the fees cost them something. The default is 20% of the company's value at the comparables' multiple, rounded to 250 mkr (25 m USD). The money raised, divided by the price, gives the number of new shares, and so the dilution (page 11).
Secondary shares	Intermediate	0	Existing shares sold in the offering by current owners, such as founders or venture funds. The money goes to them, not to the company. They make the offering larger, so more demand is needed to cover it, and they add to the free float (the shares anyone can trade), but they change neither the company's value nor the number of shares. Investors watch them: owners selling heavily at the IPO can be a warning sign.
Gross spread	Intermediate	Market: 3% / 7%	The banks' fee, as a share of the money raised. The banks (**underwriters**) organise the offering, market it and guarantee that the shares are sold. About 7% is customary for mid-sized US IPOs; European IPOs, Swedish ones included, typically pay roughly half. It is a cost to the company and lowers the value per share.

Field	Level	Default	Explanation
Other offering expenses	Advanced	Derived: 47.5 mkr / 4.75 m USD	Lawyers, auditors, printing, the regulator's and the exchange's fees: paid whatever the price. The default is 20 mkr (2 m USD) plus 1% of the raise. Like the gross spread, they reduce the value per share.
IPO discount	Beginner	15%	How far below fair value the **price range** is set, the range of prices published in the prospectus before investors place their orders. The discount rewards investors for buying a stock with no trading history, and leaves room for the price to rise on the first day. 10–15% is typical. The final **offer price** is then chosen from the orders (page 9).
Minimum IPO discount	Expert	5%	The smallest discount to fair value at which the banks will still launch the offering. Below it, investors get no reward for the risk of an untested stock, and the book would not fill. If management's minimum valuation (page 11) pushes the price range that high, the IPO is **postponed**.
Maximum price above the range	Expert	Market: 0% / 20%	How far above the top of the price range the deal may be priced. In Sweden and the rest of the EU, the top of the range is the **maximum price** stated in the prospectus; pricing above it would need a supplement that lets investors withdraw, so the default is 0%. In the US a deal may be priced about 20% above the filed range without filing again (SEC Rule 430A).

Field	Level	Default	Explanation
Lock-up	Advanced	180 days	The number of days after the IPO during which existing owners (founders, staff, venture investors) promise not to sell. It tells new investors that insiders are staying, which lifts their demand a little for longer lock-ups. When it ends, many shares may be offered at once, and the price often weakens in anticipation: the *lock-up overhang.*
Lock-up coverage	Advanced	100%	The share of the pre-IPO shares that the lock-up covers. Locked shares cannot be sold, so they are not part of the free float until the lock-up ends. Shares that are not locked are tradable from the first day: they make the stock easier to trade and to include in an index, but can also be sold into the first day's market.
Cornerstone commitment	Advanced	0	Money that **cornerstone investors**, large investors such as pension funds, promise to invest before the offering opens, at whatever price is set. Their names in the prospectus give other investors confidence, and their orders make the book safer from the start. Common in Sweden and the rest of the Nordics, and in Hong Kong. Cornerstones receive their shares first.
Cornerstone lock-in	Expert	180 days	How long the cornerstone investors promise to keep their shares. Until then those shares are not part of the free float.

Field	Level	Default	Explanation
Retail tranche	Advanced	10%	The share (*tranche*, French for slice) of the offering reserved for private investors, **retail** investors; institutions receive the rest. If retail investors order less, institutions take the difference, and the other way round. A larger retail tranche spreads ownership widely; institutions are seen as more stable long-term owners.

14.22.9 Page 9 — Investors & sentiment

This is where valuation meets **demand**. During the book-building, the banks collect orders from institutions and private investors, and learn how many shares they would buy at each price. The fields on this page describe who those investors are and how keen they are. They move the order book, and so the offer price and the expected first-day rise, but not the fair value. The exceptions are the comparable multiple and the DCF weight, which say how fair value itself is calculated. With the defaults, 40 institutions each ordering 137.5 mkr bring 5.5 mdr of orders at fair value, twice the 2.75 mdr on offer, before interest, hype and index prospects are applied.

Field	Level	Default	Explanation
Institutional interest	Beginner	medium	How warm the professional investors are: pension, insurance and investment funds. Before launching, the banks sound them out in meetings (*testing the waters*). The five levels scale institutional demand: `very_low` 0.3×, `low` 0.6×, `medium` 1×, `high` 1.5×, `very_high` 2.2×. It is the biggest single lever on the book, and it is judgement, not calculation.
Institutions in the book	Advanced	40	How many institutions place orders. Together with the average order, it sets the size of the institutional book at fair value.
Average institutional order	Advanced	Derived: 5% of the raise = 137.5 mkr	The typical institutional order, in money, at a price equal to fair value. At higher prices institutions order less (see the elasticity below).

Field	Level	Default	Explanation
Retail interest	Beginner	medium	How keen private investors are, on the same five levels as institutional interest (0.3× to 2.2×). In Sweden, private investors apply for IPO shares through their bank or online broker. Retail money is a small part of most books, but it reacts strongly to hype.
Retail applicants	Advanced	20,000	How many private investors apply for shares.
Average retail application	Advanced	25,000 SEK / 2,500 USD	The typical private investor's order. With the defaults, retail demand at fair value is 20,000 × 25,000 = 500 mkr before interest and hype.
Hype factor	Beginner	3	Press and social-media excitement around the listing, on a scale of 0 to 10. It swells retail demand (3.5 times at 10), nudges institutions a little, and makes private investors less sensitive to the price. It also adds to the expected **first-day pop**, the rise from the offer price during the first day of trading. Hype is not value: it moves the book, not the fair value.
Target coverage	Intermediate	3×	How many times over the banks want the offering to be ordered: 3× means orders for three times the shares on offer. The deal is priced at the highest price that still meets this target. An oversubscribed book leaves investors with less than they asked for, and their buying on the first day supports the price.

Field	Level	Default	Explanation
Comparable EV / NTM revenue	Intermediate	Preset: 10×	How the stock market values similar listed companies: their **enterprise value** (the value of the whole business) divided by their revenue over the **next twelve months** (NTM). Companies without profits are usually valued on revenue. The model multiplies the company's own next-year revenue by this number to get the *comparables valuation*. It moves with market fashion: the same software company may be worth 10 times revenue one year and 5 times the next. A scenario driver: bear 7×, bull 13×.
DCF weight	Intermediate	70%	How much the discounted cash flow valuation counts in fair value; the comparables valuation gets the rest. The DCF reflects your own forecast of the company's cash; comparables reflect what the market pays for similar companies today, mood included. Bankers look at both. Set 100% to see the pure value of the company's own cash flows, or 0% to price it like the market.
Institutional price elasticity	Expert	3	How sharply institutions cut their orders as the price rises above fair value: demand at price P is proportional to (fair value ÷ P) to the power of this number. At 3, a price 10% above fair value cuts orders by about a quarter. A higher number makes demand fall off faster, so the book supports a lower offer price.

14.22.10 Page 10 — Index

A **stock-market index**, such as the OMX Stockholm 30, is a list of shares whose combined price is followed as a measure of the market. **Index funds** simply buy every share in their index, so a company that joins one gets forced buying from them, and investors in the IPO value that prospect. This page is the rulebook of a fictive index: how large and how freely traded a company must be, and how long it must have traded, before it can join. The report checks the IPO against it at the offer price, and a likely inclusion raises institutional demand in the book by up to 10%. See Market Index for the index on EduMatcher.

Field	Level	Default	Explanation
Minimum market cap	Intermediate	5 mdr / 500 m USD	Index rule: the company's **market capitalisation**, the price per share × all shares, must be at least this. Indices want companies large enough to matter to the funds that track them. Checked at the offer price; size rules are never relaxed.
Minimum free float	Advanced	15%	Index rule: the share of all shares that is freely tradable, the **free float**. Shares that are locked up, or held by cornerstone investors, do not count. Index funds must be able to buy the stock without pushing its price up. The exchange may relax this rule to 10%.
Minimum free-float cap	Advanced	1.5 mdr / 150 m USD	Index rule: the value of the freely tradable shares (free-float shares × price) must be at least this. Index funds weight each member by its free-float value, so a company with a small float would have too small a weight to matter. Never relaxed.
Seasoning	Advanced	63 trading days	Index rule: how long a new stock must have traded before it can join, so that it has a price history and the swings of its first days have settled. 63 trading days is about three months. The exchange may waive most of it, bringing inclusion forward to day 10.
Fast-entry market cap	Expert	50 mdr / 5 bn USD	Index rule: an IPO at least this large skips the seasoning period and joins after a short delay, because an index that left out such a large company would no longer represent the market. Fast entry gives the strongest boost to demand in the book.
Fast-entry delay	Expert	5 trading days	How many trading days after listing a fast-entry company joins the index.

Field	Level	Default	Explanation
Multiple share classes allowed	Advanced	no	Index rule: whether companies with several share classes (dual-class shares, page 1) may join. Some indices exclude them because outside shareholders cannot outvote the founders. The exchange may agree to admit them.
Exchange may relax	Advanced	none	Which index rules the exchange is willing to loosen for this listing: `seasoning`, `float`, `multi_class`, `all`, or `none`. The report shows the index verdict both with and without the relaxation, so you can see what it is worth.
Passive assets tracking the index	Expert	none	Optional: the money in index funds that track this index. Together with the next field, it lets the report estimate how many shares the index funds must buy when the stock joins.
Index free-float market cap	Expert	none	Optional: the combined free-float value of all the index's current members. The new stock's **weight** in the index is its own free-float value ÷ (this total + its own). Index funds must then buy that weight × the passive assets above.

14.22.11 Page 11 — Management

The company's own leaders, the CEO, the CFO and the board, have limits of their own. They may refuse to sell the company below the value it had in its last funding round, give away more than a certain share of it, or raise less than it needs. This page states those limits. The model does not change the valuation to meet them; it checks them, moves the price range up when it must, and postpones the IPO when no price can satisfy both management and investors. The Halvard case shows how.

Field	Level	Default	Explanation
Last private round, post-money	Beginner	none	The value of the company at its last private (venture capital) funding round: the price paid per share then × all shares after the round, the **post-money valuation**. Founders and venture investors hate listing below it, a **down round**, because it shows that the company lost value and can trigger clauses that give some investors extra shares.
Minimum market cap	Intermediate	Derived: the last private round	The lowest market capitalisation after the IPO that management will accept. The price that meets it is the *management floor*: (minimum market cap − raise) ÷ pre-IPO shares. If the price range is below the floor, it is moved up to it. If that leaves the banks less than the minimum IPO discount to fair value (page 8), the IPO is postponed.
Maximum dilution	Intermediate	25%	The largest share of the company, after the IPO, that management will give to new investors: new shares ÷ all shares after the IPO. **Dilution** is the price of new money: existing owners keep a smaller slice of a company with more cash. The report warns if the offering exceeds it.
Minimum net proceeds	Advanced	none	Optional: the least money the company must receive, after the banks' fee and other expenses, for the IPO to be worth doing, for example to fund the next years of the plan. The report warns if the offering falls short.

14.22.12 Page 12 — Simulation

A valuation is only as good as its assumptions, and every one of them is uncertain. This page controls how the report shows that uncertainty. The **deterministic** part computes fixed cases: a pessimistic (*bear*), a base and an optimistic (*bull*) scenario, a **tornado** chart that moves one input at a time to its bear and bull value and ranks them by their effect on value, and sensitivity grids. The **Monte Carlo** part simulates thousands of possible companies, each with its inputs drawn at random between their bear and bull values, and shows the spread of fair values that results. The ten inputs that are varied are the

scenario drivers; their bear and bull values are Expert fields. See Uncertainty: scenarios, sensitivity and Monte Carlo.

Field	Level	Default	Explanation
Mode	Intermediate	both	Which analyses the report runs: `deterministic` (scenarios, tornado and sensitivity grids), `montecarlo` (the simulation) or `both`, which also compares the two. `--mode` on the command line overrides it.
Monte Carlo draws	Advanced	10,000	How many companies the simulation draws. More draws give smoother, steadier results but take longer; 10,000 takes a few seconds.
Random seed	Advanced	42	The starting point of the random number generator. The same seed gives exactly the same simulation, so a class can compare results. Change it to see how much the results move by chance alone.
Execution correlation	Advanced	0.5	How strongly the six operating drivers (customer growth, churn, price growth, market share ceiling, acquisition cost, headcount elasticity) move together in the simulation, from 0 to 1. Good and bad news tend to come together: a company that grows fast usually also keeps its customers. At 0 each is drawn independently; at 1 they move in step, which widens the spread of values. The comparable multiple and the rate drivers are always drawn independently.

Field	Level	Default	Explanation
Customer growth, year 1: bear / bull	Expert	Derived: base × 0.6 / × 1.3 (30% / 65%)	The pessimistic and optimistic ends of the plausible range for next year's customer growth (page 3). The **bear scenario** sets every driver to its bear value at once, and the bull scenario to its bull value; the tornado moves this driver alone. The Monte Carlo draws it between the two, with your base value as the most likely. The same holds for every pair below.
Annual churn: bear / bull	Expert	Derived: base × 1.5 / × 0.7 (12% / 5.6%)	The range for customers lost each year. Bear is higher churn.
ARPU growth, year 1: bear / bull	Expert	Derived: base − 2 / + 2 points (3% / 7%)	The range for next year's rise in revenue per customer.
Maximum share of SAM: bear / bull	Expert	Derived: base × 0.7 / × 1.3 (5.6% / 10.4%)	The range for the ceiling on market share (page 2).
Paid acquisition cost per customer: bear / bull	Expert	Derived: base × 1.3 / × 0.8 (390,000 / 240,000 SEK)	The range for the marketing cost of winning a customer. Bear is more expensive.
Headcount elasticity: bear / bull	Expert	Derived: base + 0.1 / − 0.1 (0.7 / 0.5)	The range for how fast staff grow with revenue. Bear is more staff per unit of growth.
Comparable EV / NTM revenue: bear / bull	Expert	Derived: base × 0.7 / × 1.3 (7× / 13×)	The range for the multiple the market pays for similar companies (page 9).
Execution premium: bear / bull	Expert	Derived: base + 3 / − 2 points (6% / 1%)	The range for the stage-1 premium for the risk that the plan fails (page 7). Bear is a higher premium, so a lower value.
Beta, year 6 on: bear / bull	Expert	Derived: base + 0.3 / − 0.2 (1.4 / 0.9)	The range for the beta of the established company (page 7).
Terminal growth: bear / bull	Expert	Derived: base − 0.5 / + 0.5 points (1.5% / 2.5%)	The range for growth after the forecast (page 7).

14.23 Where to go next

- Listing a New Symbol - what `--list` does to the configuration
- Auctions & Scheduling - the opening auction that tests the price
- Market Index - the index the report's inclusion verdict refers to
- Index Administration (`pm-index-admin-cli`) - adding the stock to an index after seasoning
- IPO Valuation training chapter - the classroom exercise

CHAPTER 15

Session Scheduling and Auctions

Learning objectives

After reading this page you will understand:

- Why exchanges use auctions rather than continuous matching at the open and close
- How the equilibrium (uncross) price is calculated
- Which order types are valid in each session phase
- What happens to resting orders at each phase transition
- How to drive phase transitions in EduMatcher

Prerequisite: A Full Trading Day gives a visual overview of the session phases before diving into the mechanics here.

15.1 What are auctions?

In real exchanges, a trading day is not a single uninterrupted period of continuous matching. Instead it is divided into **session phases**, two of which are dedicated **auction periods** — one at the open and one at the close.

An auction collects orders over a fixed time window without executing any of them. When the window ends the exchange computes a single **equilibrium price** and fills every **crossable** order at that price in one atomic event called the **uncross**. ("Crossable" means orders that overlap in price: a buy order priced at or above the equilibrium and a sell order priced at or below it — they can trade because the buyer is willing to pay at least what the seller demands.)

15.1.1 Why auctions exist

Problem	How auctions help
Opening price discovery	Overnight news creates uncertainty. An opening auction lets all participants express interest simultaneously, producing a fair opening price that reflects the balance of supply and demand — rather than letting the first few aggressive orders set the tone.
Closing price quality	The official closing price is used by fund managers to value portfolios (NAV = Net Asset Value), by index providers to rebalance index compositions, and by clearinghouses for end-of-day settlement. A closing auction concentrates end-of-day liquidity into a single price, reducing the impact of last-second volatility.

Problem	How auctions help
Reduced information asymmetry	In continuous trading a fast participant can exploit outdated ("stale") resting orders before the slower participant has time to update them — for example, news breaks and a high-frequency trader buys shares from a resting sell order whose owner hasn't yet cancelled it. Because auctions execute everything at once, speed advantages are neutralised during these windows.
Maximised fill quantity	The equilibrium algorithm explicitly maximises the number of shares that can trade, giving more participants a fill than a sequence of bilateral matches would.

Day one: the IPO reference stands in for price discovery

A freshly listed symbol has no trade history, so there is nothing to discover a price *from* yet. On day one the symbol's seeded opening reference (`last_buy_price` / `last_sell_price`, set when the symbol is listed) acts as the starting point until the first opening auction or trade establishes a real market price. See Configuration - Adding or Removing Symbols and Risk Controls - Day one (IPO) behaviour.

15.1.2 EduMatcher's auction model

EduMatcher implements a **standard two-sided call auction** used by most equity exchanges (Euronext, LSE, Nasdaq Nordic, etc.). Each symbol book participates in a global schedule of session phases:

```
PRE_OPEN → OPENING_AUCTION → CONTINUOUS → CLOSING_AUCTION → CLOSED
```

The scheduler process (`pm-scheduler`) drives these transitions by sending `session.transition` messages to the engine over ZeroMQ.

15.2 The trading date

A **trading date** is the calendar date, in the exchange's local wall clock, on which one `PRE_OPEN → OPENING_AUCTION → CONTINUOUS → CLOSING_AUCTION → CLOSED` cycle runs. It is the unit every daily figure in EduMatcher is bucketed by:

Where	Column	Meaning
`stats.db`	`daily_stats.date`, `index_daily_stats.date`	Trading date
`clearing.db`	`trade_events.trade_date`, `gateway_daily_summary.trade_date`	Trading date

The trading date is deliberately **not** the UTC date, and **not** the date on the clock of whichever machine happens to be running a process. An exchange's daily rollup has to describe the session its participants actually traded. A venue whose session runs into the evening — or any venue whose session straddles 00:00 UTC — would otherwise have a single trading session split across two dates, and every daily summary, P&L statement and index close would describe half a day.

All timestamps (`ts` columns, log lines, message payloads) remain **UTC instants**. Only the daily rollups use the trading date. Converting between the two is exactly the job of the session timezone.

15.2.1 Setting the session timezone

The scheduler and the two **recorders**, `pm-stats` and `pm-clearing`, read `locale.session_timezone` from the engine config, an IANA timezone name defaulting to `"Europe/Stockholm"`:

```
locale:
  session_timezone: Europe/Stockholm
```

Each recorder writes the value into its own database. Every **reader** then picks it up from there and needs no configuration:

```
pm-stats-cli daily --date 2026-06-14   # resolves in Europe/Stockholm
pm-ticker                              # same
```

`pm-stats-cli --timezone` and `pm-ticker --timezone` remain available as explicit overrides on the reader side, and warn when they contradict what the database records.

The two recorders must agree with each other

`pm-stats` and `pm-clearing` write separate databases, and nothing cross-checks them while running. If they are started against different `locale.session_timezone` values, `daily_stats.volume` will not reconcile against `gateway_daily_summary.traded_qty`, and neither process will report an error.

Within a single database the value is enforced: restarting a recorder against an existing file with a different `locale.session_timezone` is refused outright, since its `date` column would otherwise mean two different things.

Since both processes read the same compiled config, this can only happen by redeploying the config with a changed `session_timezone` between restarts — set it once and leave it alone. If your exchange runs in UTC, set `session_timezone: UTC`.

15.2.2 Relationship to `locale.country`

`locale.country` (used by `pm-scheduler` for the bank-holiday calendar, see Bank holidays and weekends) and `locale.session_timezone` are **separate settings and are not derived from one another**, even though they are grouped under the same `locale:` block and share a Swedish default. `locale.country` decides *which days* the schedule runs on; `locale.session_timezone` determines the scheduler's clock and *which calendar day* a timestamp is filed under. A country can span several timezones, so one cannot be inferred from the other.

Keep them consistent: an exchange configured with `locale.country: Sweden` should also be recording with `locale.session_timezone: Europe/Stockholm` (the default for both).

15.2.3 Trading days are not always 24 hours

On a daylight-saving transition the trading date spans 23 or 25 hours, and the query tooling accounts for this — a `--date` filter resolves to the instant range between local midnight and the *next* local midnight, whatever its length. Nothing needs configuring for this to work.

The scheduler follows `locale.session_timezone`, independently of the host's timezone. Its waits and published UTC countdown targets account for offset changes automatically. On the autumn rollback, a transition inside the repeated hour runs **only on the second occurrence** (for example, 02:30 CET, not 02:30 CEST, in Stockholm). Startup catch-up uses that same instant. During the spring jump, a nonexistent time is shifted forward by the gap; prefer scheduling transitions outside that missing hour to preserve the intended spacing between phases.

Upgrading from host-local scheduling

Older versions interpreted schedule times in the host's timezone. If that differs from `locale.session_timezone`, check the deployed timezone before restarting the scheduler: the same `HH:MM` now means exchange-local time, including for holiday selection and daily rollover.

15.3 Session phases

Phase	Orders accepted?	Matching?	Description
`PRE_OPEN`	Yes	No	Orders rest on the book, no execution. Participants can position themselves before the auction starts.
`OPENING_AUCTION`	Yes	No	Orders continue to accumulate. **ATO** (At-The-Open) orders are accepted only during this phase.
`CONTINUOUS`	Yes	Yes	Normal price-time-priority matching. Every incoming order is immediately swept against resting liquidity.

Phase	Orders accepted?	Matching?	Description
`CLOSING_AUCTION`	Yes	No	Orders accumulate for the closing uncross. **ATC** (At-The-Close) orders are accepted only during this phase.
`CLOSED`	No	No	Market is closed. All new orders are rejected.

15.3.1 Order acceptance rules by phase

Order characteristic	PRE_OPEN	OPENING_AUCTION	CONTINUOUS	CLOSING_AUCTION	CLOSED
LIMIT / ICEBERG	Rest	Rest	Match	Rest	Reject
MARKET	Reject	Reject	Match	Reject	Reject
FOK	Reject	Reject	Match	Reject	Reject
IOC	Reject	Reject	Match	Reject	Reject
STOP / STOP_LIMIT	Rest (no trigger)	Rest (no trigger)	Normal trigger logic	Rest (no trigger)	Reject
TRAILING_STOP	Rest (no trigger[1])	Rest (no trigger[1])	Normal trigger logic	Rest (no trigger[1])	Reject
TIF = ATO	Reject	**Accept**	Reject	Reject	Reject
TIF = ATC	Reject	Reject	Reject	**Accept**	Reject

[1] This check is not actually phase-specific: a TRAILING_STOP order is rejected in **any** session phase (including CONTINUOUS) if no `STOP=` is given and no prior trade has established a `last_trade_price` for the symbol — the engine needs one of the two to compute an initial stop price. In practice this almost always bites during PRE_OPEN/auction phases, since that is before the symbol's first trade of the day.

MARKET, FOK, and IOC orders are always rejected outside CONTINUOUS because they cannot rest on the book — they require immediate execution.

15.3.2 Transition side-effects

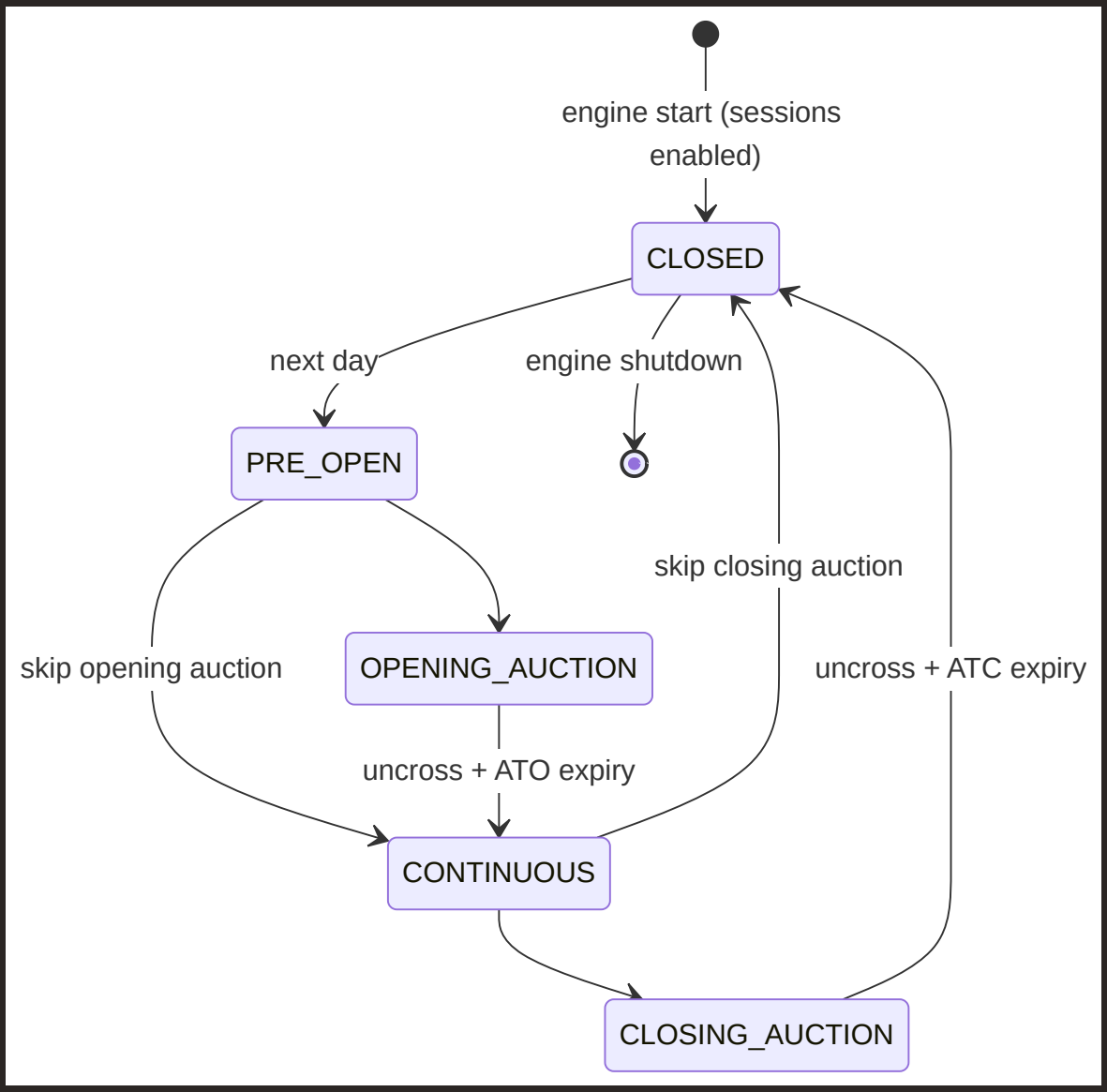
When the engine transitions **into a matching phase, or into `CLOSED` from `CLOSING_AUCTION`** (i.e. when the new state has matching enabled and the old one did not, or the transition is the end-of-day close — including the `PRE_OPEN → CONTINUOUS` shortcut):

1. **Uncross** — the equilibrium price algorithm runs on every symbol book and executes all crossable interest at the equilibrium price.
2. **TIF expiry** — ATO orders are expired when leaving `OPENING_AUCTION`; ATC orders are expired when leaving `CLOSING_AUCTION`. Expired orders receive an `order.expired` event on the PUB socket.

Note

The uncross also fires on `PRE_OPEN → CONTINUOUS` (skipping the opening auction). Any orders that accumulated during pre-open are crossed at the equilibrium price before continuous matching begins.

15.3.3 Valid state transitions



Invalid transitions are silently rejected by the engine and logged to stdout (the engine's `logging` output stream, not stderr).

15.4 The session scheduler (`pm-scheduler`)

The scheduler is a standalone process that sends `session.transition` messages to the engine at configured wall-clock times over a ZeroMQ PUSH socket. It only does so on days that resolve to a schedule — see Bank holidays and weekends below.

15.4.1 Starting the scheduler

```
# Use times from engine_config.yaml (or built-in defaults); run today's
# schedule once (catching up on any times already passed) and then exit
poetry run pm-scheduler

# Repeat the schedule every calendar day instead of exiting after today
poetry run pm-scheduler --daily

# Rapid-fire all transitions immediately (for testing / demos)
poetry run pm-scheduler --now

# Custom delay between transitions in --now mode (default 3 s)
poetry run pm-scheduler --now --delay 5

# Compressed mode: whole days (weekends and holidays included) in minutes
poetry run pm-scheduler --daily --speed 600 --start 2026-11-02T08:55

# pm-scheduler always reads the deployed compiled artifact (like pm-engine) --
# there is no flag to point it at a different config file; see
# [Configuration -- Missing-file Behavior](../part-2-configure/010-the-configuration-workflow.md) for why.
poetry run pm-scheduler
```

15.4.2 CLI flag reference

Flag	Default	Description
`--now`	off	Rapid-fire all five transitions immediately (for testing / demos)
`--delay SECONDS`	`3.0`	Seconds between transitions in `--now` mode; ignored (with a warning) otherwise
`--daily`	off	Run continuously, repeating the schedule every calendar day instead of exiting after today
`--speed F`	off	Compressed mode (needs `--daily`): simulated time runs F times faster than the wall clock; see Compressed mode
`--start YYYY-MM-DDTHH:MM`	now	Compressed mode: local time the simulated clock starts at
`--no-confirm`	off	Skip querying/confirming session state via the engine's `session.state` broadcast
`--on-close COMMAND`	off	Start `COMMAND` after each day's `CLOSED` transition and once at start-up; detached, one at a time, output in the scheduler log. Typically `"pm-eod run --pending"` — see Closing the Day
`--on-close-delay SECONDS`	`30`	Wait this long after `CLOSED` before starting `--on-close`, so `pm-clearing` has saved its close snapshot
`--log-level {CRITICAL,ERROR,WARNING,INFO,DEBUG}`	`WARNING`	Explicit logging level override
`--verbose` / `-v`	off (WARNING)	Increase log verbosity; repeatable (`-v` = INFO, `-vv` = DEBUG)
`--quiet` / `-q`	off	Reduce log output to warnings/errors (this is already the default level)

15.4.3 After the close: `--on-close`

When the engine reaches `CLOSED` the trading day is over, and there is work to do: settle the traders' gains and losses, write their statements, back things up. The scheduler can start that work for you. Give it a command with `--on-close`; it starts the command a short time after every confirmed `CLOSED` (and once when the scheduler itself starts, to catch a day that closed while it was down):

```
# checkdocs: ignore  (the quoted command has its own flags)
pm-scheduler --daily --on-close "pm-eod run --pending"
```

The command runs in a separate process — the scheduler does not wait for it, so a slow or failing command can never delay tomorrow's timetable — and only one runs at a time. Its output and exit code appear in the scheduler's log. The 30-second default pause (`--on-close-delay`) lets `pm-clearing` save its close snapshot first. What `pm-eod` does is explained, from scratch, in Closing the Day.

15.4.4 Configuring the schedule

A `schedule` section resolves to one schedule per day of the week — Monday through Sunday — plus an optional override for bank holidays. Each day (or the holidays entry) is either a block of all five transition times, or unconfigured, which means **CLOSED** that day: no transitions are sent and the market never leaves `CLOSED`.

Two shortcuts avoid repeating the same five times for every day:

- `weekdays:` — applies to Monday-Friday, for any of the five that isn't individually overridden by a `mon:..fri:` block.
- `weekend:` — applies to Saturday and Sunday, for either that isn't individually overridden by `sat:/sun:`. `weekend:` and `sat:/sun:` are **mutually exclusive** — specify one or the other, not both.

Every block that *is* present — `weekdays`, `weekend`, an individual `mon..sun`, or `holidays` — MUST define all five transition times (`pre_open`, `opening_auction_start`, `continuous_start`, `closing_auction_start`, `closing_auction_end`). There is no per-key fallback: a partial block is a config error, not a partially-applied schedule.

Typical exchange: same hours Monday-Friday, closed on weekends

```
schedule:
  weekdays:
    pre_open: "09:00"
    opening_auction_start: "09:25"
    continuous_start: "09:30"
    closing_auction_start: "16:00"
    closing_auction_end: "16:05"
```

Saturday, Sunday and bank holidays are all unconfigured here, so they default to CLOSED.

Short Friday, a shared weekend schedule, and a holiday half-day

```
schedule:
  weekdays:
    pre_open: "09:00"
    opening_auction_start: "09:25"
    continuous_start: "09:30"
    closing_auction_start: "16:00"
    closing_auction_end: "16:05"
  fri:                      # overrides weekdays: for Friday only
    pre_open: "09:00"
    opening_auction_start: "09:25"
    continuous_start: "09:30"
    closing_auction_start: "13:00"
    closing_auction_end: "13:05"
  weekend:                  # same hours for both Saturday and Sunday
    pre_open: "10:00"
    opening_auction_start: "10:25"
    continuous_start: "10:30"
    closing_auction_start: "14:00"
    closing_auction_end: "14:05"
  holidays:                 # bank holidays get a reduced session too
    pre_open: "10:00"
    opening_auction_start: "10:25"
    continuous_start: "10:30"
    closing_auction_start: "13:00"
    closing_auction_end: "13:05"
```

An individually-specified day always wins over `weekdays`/`weekend` for that day: `fri:` here overrides the `weekdays:` hours for Friday only, while Monday through Thursday still run on the `weekdays:` block.

Weekend-only exchange

A schedule doesn't have to run Monday-Friday at all. Omitting `weekdays` (and every individual `mon`..`fri` key) leaves those five days CLOSED:

```
schedule:
  weekend:
    pre_open: "10:00"
    opening_auction_start: "10:25"
    continuous_start: "10:30"
    closing_auction_start: "14:00"
    closing_auction_end: "14:05"
```

Invalid: `weekend` and `sat`/`sun` combined

```
schedule:
  weekend:
    pre_open: "10:00"
    opening_auction_start: "10:25"
    continuous_start: "10:30"
    closing_auction_start: "14:00"
    closing_auction_end: "14:05"
  sat:
    pre_open: "11:00"
    opening_auction_start: "11:25"
    continuous_start: "11:30"
    closing_auction_start: "13:00"
    closing_auction_end: "13:05"
```

Rejected at load: `weekend` and an individual `sat`/`sun` key say two different things about the same day, so the config MUST pick one.

Invalid: an incomplete day block

```
schedule:
  weekdays:
    pre_open: "09:00"
    opening_auction_start: "09:25"
    continuous_start: "09:30"
    closing_auction_start: "16:00"
    # closing_auction_end missing
```

Rejected at load: `weekdays` is present but doesn't define all five transition times.

Times are `HH:MM` in `locale.session_timezone` (default `Europe/Stockholm`). Naive `--start` values use this timezone too; an explicit UTC offset is converted to the session timezone. Any day left with no resolved schedule — not covered by `weekdays`/`weekend` and not individually specified — defaults to **CLOSED**, exactly like an omitted `holidays:` block. See Bank holidays and weekends below for how the `holidays:` entry and the `locale.country` key interact.

15.4.5 Session handling toggle

Session gating in the engine is controlled by `sessions_enabled`:

```
sessions_enabled: true
```

- `true` (the default when a config file is present but omits the field): session transitions are enforced, the engine starts `CLOSED`, and new orders are rejected outside order-accepting session states.
- `false`: session handling is disabled, `session.transition` messages are ignored by the engine, and the engine starts (and remains) in `CONTINUOUS` state — all order types are accepted and matched immediately. This is also the effective value when **no** config file is present at all (unrestricted mode).

Use `sessions_enabled: false` when you want an always-open simulation without time-based session control.

Default value

With a config file present, omitting `sessions_enabled` enables sessions (`true`). It is only `false` when you set it explicitly or run with no config file. See Configuration → `sessions_enabled` for the full scenario table.

15.4.6 Bank holidays and weekends

Which block of the resolved weekly table applies to a given calendar day is decided in two steps, using the `python-holidays` package:

1. If the day is a bank holiday for the configured `locale.country`, the `holidays:` entry applies — or CLOSED, if no `holidays:` block is configured.
2. Otherwise, the day's own entry from the weekly table applies (`mon..sun`, already resolved from `weekdays`/`weekend`/individual overrides as described above) — or CLOSED, if that day has no resolved entry.

```
locale:
  country: Sweden
```

- `locale.country` accepts either a country name (`"Sweden"`) or an ISO 3166-1 alpha-2 code (`"SE"`).
- `locale.country` selects the **holiday calendar** only. It does not set the session timezone used to bucket daily statistics — that is `locale.session_timezone`, described under The trading date.
- If omitted, or if the value is not a country `python-holidays` recognizes, it **falls back to `"Sweden"`** and logs a warning.
- Weekends are no longer forced CLOSED — a `weekend:` or `sat:`/`sun:` block gives Saturday and/or Sunday their own schedule, the same as any weekday. Only a day with no resolved entry at all (weekday, weekend day, or holiday) is CLOSED.

Behavior when today resolves to CLOSED depends on the run mode:

Mode	Behavior on a CLOSED day
Single-shot (default)	Sends no transitions and exits immediately — today's schedule is skipped entirely.
`--daily`	Skips today, then sleeps through to the **next day that resolves to a schedule** (not just the next calendar day) before running the schedule again.
`--now`	Unaffected — `--now` ignores wall-clock scheduling (and therefore schedule resolution) entirely, since it's intended for testing/demos.

```
locale:
  # Germany's holiday calendar instead of the default (Sweden)
  country: DE
```

```
2026-12-25 is CLOSED today (bank holiday in Sweden); skipping today's schedule
```

(shown with `-v`/`--verbose`, since it's an `INFO`-level log line)

Why this matters for –daily

Before per-day schedules existed, a scheduler left running continuously with `--daily` would happily drive `CLOSED → PRE_OPEN → … → CLOSED` on a Sunday or on Christmas Day. A day with no resolved schedule — a weekend with no `weekend:`/`sat:`/`sun:` block, or a bank holiday with no `holidays:` block — is skipped, and the engine simply stays `CLOSED` until the next day that does resolve to a schedule.

15.4.7 Wall-clock re-checking

Long waits — whether counting down to a scheduled transition time or sleeping overnight between trading days under `--daily` — are never slept out in a single call. Instead, `pm-scheduler` wakes at least every **30 seconds** and re-derives the remaining wait from the current local wall-clock time. This means a server time adjustment (NTP sync, manual clock change, DST) made while the scheduler is waiting is picked up within 30 seconds, rather than only being noticed after a long sleep computed under now-stale assumptions finally elapses.

15.4.8 Countdown to the next transition

Each transition the scheduler sends also names the one after it, and the engine publishes that as the countdown target the GUIs and CALF clients show (`Closed → Pre-Open in 15:42:10`). There are two further cases:

- **The close.** Under `--daily` the close names the next trading day's pre-open, skipping weekends and bank holidays, so a closed exchange counts down to its reopening. A single-shot run's close names nothing, because the scheduler will have exited before then.
- **Startup.** A scheduler that starts with nothing to catch up announces the transition it is about to wait for (`session.next`). That covers a start before the open, between phases, after the close, or on a closed day (`--daily` only, for the last two).

The engine keeps the target and returns it from its session-state query, so a `pm-md-gwy` that restarts while the market is closed shows the countdown again within a couple of seconds. A manual transition clears the target until the scheduler sends its next transition or is restarted.

15.4.9 Built-in default schedule

If no config file provides a `schedule` section, `pm-scheduler` uses these times as `weekdays:` (Saturday, Sunday and bank holidays default to CLOSED):

Time	Transition target
09:00	`PRE_OPEN`
09:25	`OPENING_AUCTION`
09:30	`CONTINUOUS`
16:00	`CLOSING_AUCTION`
16:05	`CLOSED`

15.4.10 `--now` mode

The `--now` flag skips wall-clock waiting and sends every transition in the default sequence with a short configurable delay between each. This is useful for integration testing and demos where you want to exercise the full auction lifecycle in seconds rather than hours.

```
poetry run pm-scheduler --now --delay 2 -v
```

`-v` is needed to see the transition log below — like every other `pm-` process, `pm-scheduler` is quiet by default (`WARNING` and above only); `-v` raises the level to `INFO`.

Output:

```
2026-07-23 09:30:00,000 INFO edumatcher.scheduler.main - starting pm-scheduler with log level INFO
2026-07-23 09:30:00,001 INFO edumatcher.scheduler.main - --now mode: sending all transitions with 2.0s delays
2026-07-23 09:30:00,001 INFO edumatcher.scheduler.main - -> PRE_OPEN
2026-07-23 09:30:00,001 INFO edumatcher.scheduler.main - Waiting 2s for OPENING_AUCTION
2026-07-23 09:30:02,003 INFO edumatcher.scheduler.main - -> OPENING_AUCTION
2026-07-23 09:30:02,003 INFO edumatcher.scheduler.main - Waiting 2s for CONTINUOUS
2026-07-23 09:30:04,005 INFO edumatcher.scheduler.main - -> CONTINUOUS
2026-07-23 09:30:04,005 INFO edumatcher.scheduler.main - Waiting 2s for CLOSING_AUCTION
2026-07-23 09:30:06,007 INFO edumatcher.scheduler.main - -> CLOSING_AUCTION
2026-07-23 09:30:06,007 INFO edumatcher.scheduler.main - Waiting 2s for CLOSED
2026-07-23 09:30:08,009 INFO edumatcher.scheduler.main - -> CLOSED
2026-07-23 09:30:08,009 INFO edumatcher.scheduler.main - Done.
```

15.4.11 Compressed mode

`--daily --speed F` runs the scheduler on a simulated local clock that starts at `--start` (default: now) and runs F times faster than the wall clock. The schedule, weekends and bank holidays are resolved in simulated time, so a month of trading days passes in minutes — this is how the AI-trader swarm is soak-tested. With `--speed 600`, a 09:00–16:05 day takes about 43 seconds and a 5-minute auction half a second.

The countdown the engine publishes with each transition (`next.at`) is the *wall-clock* instant the next transition will happen, so terminal countdowns stay correct. Everything else in the exchange runs on the wall clock: order ages, heartbeats and the bots' decision rates are not compressed.

15.5 Equilibrium price

The **equilibrium price** (also called the **auction price** or **uncross price**) is the single price at which the auction executes all crossable interest. It is the price that maximises the number of shares traded while minimising the leftover imbalance.

15.5.1 Intuition

Imagine plotting two curves:

- **Cumulative demand** — for each candidate price P, how many shares are buyers willing to buy at P or higher? (Sum of all bid quantities where bid price $\geq P$.)
- **Cumulative supply** — for each candidate price P, how many shares are sellers willing to sell at P or lower? (Sum of all ask quantities where ask price $\leq P$.)

The equilibrium price sits where these two curves cross — the point where the most shares can change hands.

15.5.2 Algorithm

The engine computes the equilibrium price in $O(p)$ time, where p is the number of distinct price levels on the book.

Step 1 — Build cumulative quantity arrays

Let $B = \{b_1, b_2, \ldots\}$ be the set of distinct bid prices sorted **descending** (highest first), and $A = \{a_1, a_2, \ldots\}$ the set of distinct ask prices sorted **ascending** (lowest first).

Compute the running cumulative quantities:

$$\text{cum_buy}[b_i] = \sum_{j=1}^{i} \text{bid_qty}(b_j)$$

$$\text{cum_sell}[a_i] = \sum_{j=1}^{i} \text{ask_qty}(a_j)$$

Because bids are sorted highest-first, $\text{cum_buy}[b_i]$ gives the total bid quantity at prices $\geq b_i$. Symmetrically, $\text{cum_sell}[a_i]$ gives total ask quantity at prices $\leq a_i$.

Step 2 — Evaluate every candidate price

The set of candidate prices is $C = B \cup A$ (every distinct price level on either side), sorted ascending.

For each candidate price $P \in C$:

$$\text{buy_qty}(P) = \text{cum_buy}[\min\{b \in B \mid b \geq P\}]$$

$$\text{sell_qty}(P) = \text{cum_sell}[\max\{a \in A \mid a \leq P\}]$$

$$\text{exec_qty}(P) = \min\big(\text{buy_qty}(P),\ \text{sell_qty}(P)\big)$$

$$\text{surplus}(P) = \big|\text{buy_qty}(P) - \text{sell_qty}(P)\big|$$

Step 3 — Select the best price

$$P^* = \arg\max_{P \in C} \text{exec_qty}(P)$$

If multiple prices yield the same maximum exec_qty, break ties by:

1. Pick the P with the smallest surplus (least imbalance).
2. If still tied, pick the **lowest** price among the candidates. (This is an arbitrary but deterministic convention; some real exchanges pick the price nearest the last traded price instead.)

If $\text{exec_qty}(P^*) = 0$, there is no crossable interest and no uncross takes place.

Step 4 — Determine imbalance

At the chosen P^*:

- If buy_qty > sell_qty → imbalance side is **BUY**
- If sell_qty > buy_qty → imbalance side is **SELL**
- If equal → **balanced** (no imbalance)

The surplus quantity is the number of shares that could not be matched.

15.5.3 Worked example

Consider this order book accumulated during an opening auction:

Side	Price	Quantity
BUY	105	10
BUY	103	10
BUY	100	10
SELL	102	15
SELL	104	10

Cumulative buy (highest first): 105→10, 103→20, 100→30

Cumulative sell (lowest first): 102→15, 104→25

Evaluate candidates (ascending):

P	buy_qty	sell_qty	exec_qty	surplus
100	30	0	0	30
102	20	15	15	5
103	20	15	15	5
104	10	25	10	15
105	10	25	10	15

Maximum exec_qty = **15** at prices 102 and 103 (both with surplus 5). Tie-broken by lowest price: $P^* = 102$.

Result: 15 shares execute at **102.00**, with a BUY-side imbalance of 5 shares.

15.5.4 Uncross execution

Once the equilibrium price P^* is determined, the engine sweeps the book:

1. Take the best bid (highest price, earliest time) and the best ask (lowest price, earliest time). This is **price-time priority**: orders at better prices go first; among equal prices, the one that arrived earlier goes first.
2. If the bid price $\geq P^*$ and the ask price $\leq P^*$, fill $\min(\text{bid_remaining}, \text{ask_remaining})$ shares at P^*.

3. Repeat until no more crossable interest remains.
4. Any remaining orders whose prices do NOT cross P^* (surplus orders) stay on the book and participate in the next phase (typically continuous trading). They are not expired or cancelled — they simply did not get matched in the auction.

All fills occur at the single equilibrium price — there is no price improvement or slippage during the uncross.

15.5.5 Published events

When an uncross completes, the engine publishes:

Topic	Content
`order.fill.{gateway_id}`	One per order that received a fill (partial or complete)
`trade.executed`	One per matched pair
`auction.result.{symbol}`	Summary: equilibrium price, quantity, surplus, imbalance side
`order.expired.{gateway_id}`	One per ATO/ATC order that did not fill and was expired
`session.state`	Confirms the new session state after the transition

15.6 Hands-on: driving an opening auction manually

This short walkthrough uses two terminals with `sessions_enabled: true`. Terminal 1 runs `pm-admin --id OPS01` (a gateway configured with `role: ADMIN`) — the `SESSION|STATE=` command that advances the session is only available there, not in `pm-alf-console`.

```
# Terminal 1 — ADMIN gateway (pm-admin --id OPS01)
[OPS01|ADMIN]> SESSION|STATE=OPENING_AUCTION
SESSION  PRE_OPEN → OPENING_AUCTION

# Terminal 2 — trader submits orders during auction
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=ATO
[TRADER01]> NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=149.50|TIF=ATO

# Terminal 1 — trigger the uncross
[OPS01|ADMIN]> SESSION|STATE=CONTINUOUS
SESSION  OPENING_AUCTION → CONTINUOUS
# Engine logs: UNCROSS AAPL: 1 trade(s) @ 149.5, qty=100, surplus=0 (...)
# and publishes an auction.result.AAPL message with the same fields.
```

Both ATO orders fill at the equilibrium price. The `TIF=ATO` orders are expired if they did *not* fill.

Why 149.50 and not the midpoint

With only one bid (150.00) and one ask (149.50), both candidate prices (149.50 and 150.00) achieve the same maximum `exec_qty` (100) with the same surplus (0). Per the tie-break rule (Step 3 above), the algorithm picks the **lowest** of the tied candidate prices — 149.50 — not the midpoint between the bid and ask. EduMatcher's uncross always settles at one of the resting order prices, never an interpolated value.

15.7 See also

In this book:

- The Configuration Workflow — `schedule:` YAML keys, `locale.country`, and `sessions_enabled`
- Running the Engine — the `--now` shortcut for rapid session cycling
- Risk Controls — circuit-breaker resumption modes that re-use the uncross algorithm
- Closing the Day — settlement, statements and housekeeping after `CLOSED`

Participant Guide:

- Order Types — ATO and ATC time-in-force explained

Reference Manual:

- Processes, Environment and Ports — how to start and configure `pm-scheduler`

Protocols and Clients:

- Message Reference — `session.state`, `auction.result`, and `order.expired` message formats

CHAPTER 16

Risk Controls

Learning objectives

After reading this page you will understand:

- How an instrument halt state prevents trading while a symbol is suspended
- How price collars reject orders that stray too far from a reference or last-traded price
- How order-size and notional limits cap how *big* a single order may be, independently of its price
- How circuit breakers detect violent price moves and automatically halt and resume a symbol
- How the kill switch lets any gateway instantly cancel all its own resting orders
- How Self-Match Prevention (SMP) stops a gateway from trading against its own resting orders, and the four actions (`NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`) it can take
- Why SMP is resolved per order, not fixed per gateway, and how a gateway-level default fills in when an order or quote doesn't specify one
- How all six mechanisms interact with the order types described in Order Types
- How to configure each feature in `engine_config.yaml`

Prerequisite: Concepts — Order Book explains the order lifecycle that underpins these controls.

16.1 Overview

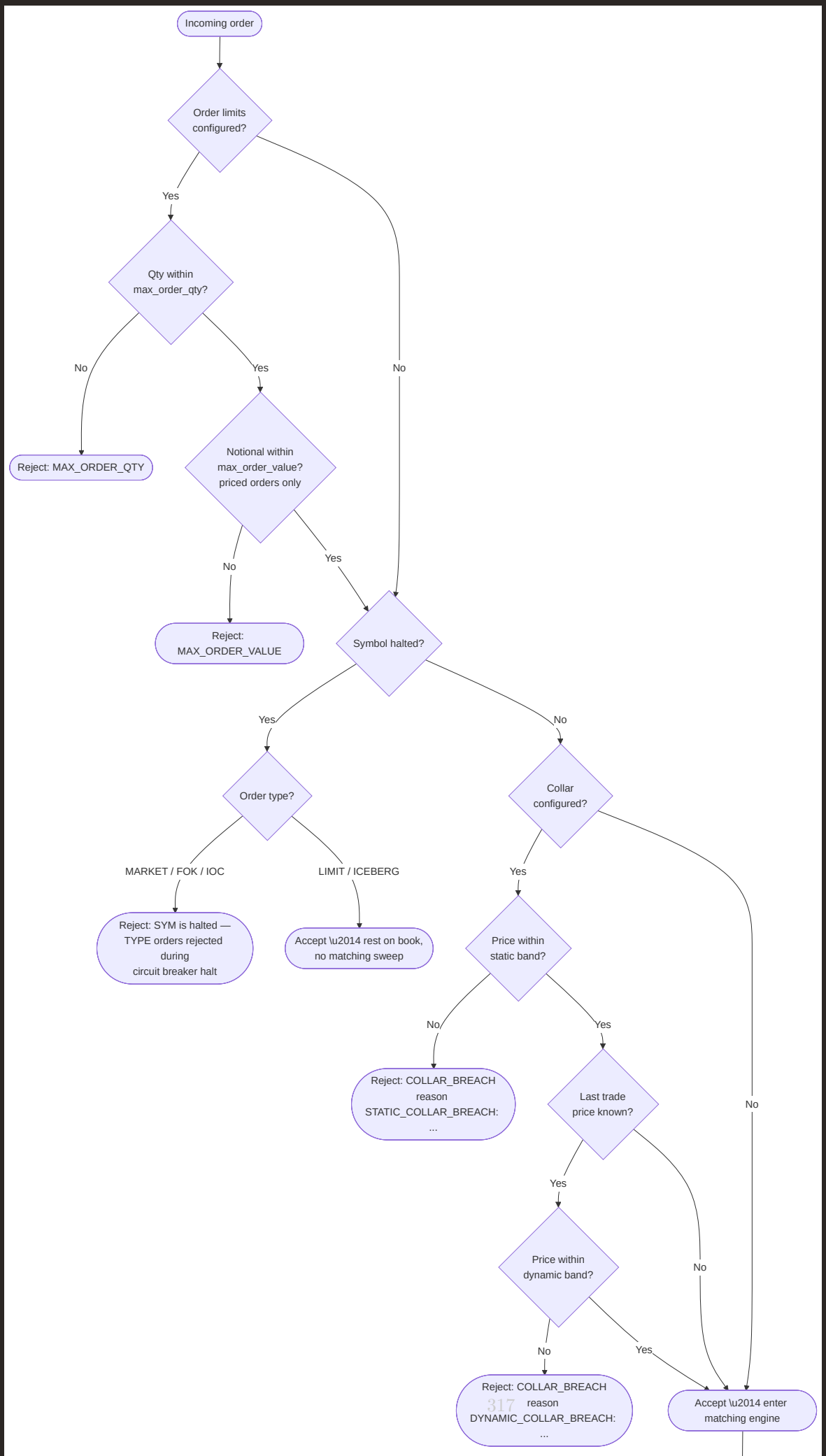
Real exchanges operate several layers of protection against runaway prices and disorderly markets. EduMatcher implements six complementary mechanisms:

Mechanism	Who sets it	What it checks	How it resolves
Instrument halt	Operator (external message)	Symbol is marked `HALTED`	Operator sends resume
Price collar	Config per symbol	Incoming order price vs. reference band	Order is rejected
Order size / notional limits	Config per symbol	Incoming order quantity, and quantity × price	Order is rejected
Circuit breaker	Config per symbol	Last trade price vs. rolling reference	Automatic halt + scheduled resume
Kill switch	Any authenticated gateway	—	Cancels all resting orders for that gateway
Self-Match Prevention (SMP)	Per order/quote, or a gateway-level default	Incoming order would trade against a resting order from the *same* gateway	Cancel aggressor, resting order, both, or allow the trade

The first four — instrument halt, price collar, order size / notional limits, and circuit breaker — act on the **order admission path**, *before* an order enters the book. The kill switch is different: it acts on orders that are **already resting**, cancelling a

gateway's own exposure on demand. SMP is different again: it acts **during matching**, at the instant a specific incoming order would otherwise cross against a specific resting order — see Self-Match Prevention (SMP) below.

The following diagram shows the order admission path through those four admission-path controls:


Incoming order
Order limits configured?
Yes
No
Qty within max_order_qty?
No
Yes
Reject: MAX_ORDER_QTY
Notional within max_order_value? priced orders only
No
Yes
Reject: MAX_ORDER_VALUE
Symbol halted?
Yes
No
Order type?
MARKET / FOK / IOC
LIMIT / ICEBERG
Reject: SYM is halted — TYPE orders rejected during circuit breaker halt
Accept \u2014 rest on book, no matching sweep
Collar configured?
Yes
No
Price within static band?
No
Yes
Reject: COLLAR_BREACH reason STATIC_COLLAR_BREACH: ...
Last trade price known?
Yes
No
Price within dynamic band?
No
Yes
Reject: COLLAR_BREACH reason DYNAMIC_COLLAR_BREACH: ...
Accept \u2014 enter matching engine

Order limits have no global switch

Unlike collars and circuit breakers, order limits are turned off by simply not setting them on a symbol: an absent cap is not enforced. There is deliberately no `enforce_order_limits` flag, no risk-level tier and no built-in fallback number — a limit that exists only in code and not in the configuration file is not auditable as a requirement.

Global on/off switches

Collar and circuit-breaker enforcement can each be disabled engine-wide with the top-level `enforce_collars` / `enforce_circuit_breakers` boolean config fields (both default to `true`). These are blunt, engine-startup switches intended for tests, not per-symbol controls — see Configuration - Engine Behavior Flags.

16.2 Instrument halt state

16.2.1 What it does

Each symbol in EduMatcher can be in one of two states:

State	Orders accepted?	Quotes accepted?
`ACTIVE`	Yes	Yes
`HALTED`	LIMIT and ICEBERG rest (do not match); MARKET, FOK, IOC are rejected	Rejected immediately

A halt is an operator-initiated pause. It is broader than a circuit breaker pause: it can be applied manually at any time, for any reason (e.g., a pending news announcement, a technology incident at a venue, a regulatory instruction).

16.2.2 State model

The two states are represented as:

```
class InstrumentState(str, Enum):
    ACTIVE = "ACTIVE"
    HALTED = "HALTED"
```

but the halt/resume request path does not read this enum at all. What actually gates order and quote acceptance is `self._halted_symbols: dict[str, bool]` on the engine — a plain boolean flag per symbol, checked with `.get(symbol)`. A symbol can appear in the dictionary with value `False` (an operator resumed it, but the entry was never removed) as well as be absent entirely; both read as `ACTIVE`. Only the boolean value is ever consulted, so the distinction is invisible to a client — but a reader inspecting engine state directly should not expect "key present" to mean "halted."

16.2.3 Halt behaviour by order type

When the engine receives a new order for a halted symbol:

- **MARKET / FOK / IOC** — rejected immediately. These order types require immediate execution and cannot be held on the book.

  The `reject_code` says **which kind of halt** you hit, because the two are not equivalent to a client: an automatic volatility halt clears itself and is worth waiting out, while a discretionary one does not and is not.

Halt source	`reject_code`	Reason text
Circuit breaker tripped on a price move	`CIRCUIT_BREAKER_ACTIVE`	`"<SYMBOL> is halted — <TYPE> orders rejected during circuit breaker halt"`
Administrator halted the symbol, or halted everything	`INSTRUMENT_HALTED`	`"<SYMBOL> is halted — <TYPE> orders rejected during trading halt"`

A symbol with no circuit breaker configured can only be halted by an administrator, so it always answers `INSTRUMENT_HALTED`.

- **LIMIT / ICEBERG** — accepted and placed on the book but the engine suppresses the continuous-matching sweep. The order will rest until the symbol is resumed, at which point it participates in the reopening.

16.2.4 Resuming

A halt is a call phase, so the resting orders may cross: a buy at 101 and a sell at 99 can both rest. Resuming therefore always starts with a reopening auction (`auction.result` with `REOPEN`, prints with match type `REOPEN_AUCTION`) that uncrosses the book at its equilibrium price, and only then publishes `circuit_breaker.resume` and returns to continuous trading. An administrator's `RESUME_SYM` / `resume-sym`, or `resume` for every halted symbol, reopens at once, without the circuit breaker's reopening corridor: the operator has decided the symbol reopens. If the session is not in a matching phase at that moment (pre-open, an auction call), the resume only lifts the halt, and the session's own auction uncrosses the book.

When the engine receives a new quote for a halted symbol, the entire quote is rejected (both sides) with reason `"<SYMBOL> is halted — quotes rejected during circuit breaker halt"`.

Quote eligibility by participant role (for example, `MARKET_MAKER` vs `TRADER`) is described in Configuration - Role Privileges.

16.2.5 Interaction with auctions

If a symbol is halted during an auction phase the engine will still accept LIMIT orders so they can participate in the uncross when the halt is lifted and trading resumes. Market-maker quotes are rejected because a quote always implies willingness to trade immediately and a halted market does not offer that guarantee.

16.3 Price collars

16.3.1 Motivation

Price collars are a pre-trade filter. They prevent an erroneous ("fat-finger") order from moving the market far from its fair value. Without collars a mistyped limit price could instantly sweep the entire book.

16.3.2 Band definitions

EduMatcher supports two independent collar bands, both configured per symbol:

Band	Measured from	Purpose
Static band	Reference price (typically last closing price, set in config)	Wide safety net; catches gross errors
Dynamic band	Last traded price in this session	Tighter rolling band; tracks intra-day moves

Both bands are expressed as a *percentage of the relevant reference price*. The boundary tick values are computed with **truncation toward zero** (`int()`) so that the protected range is always at least as tight as the nominal percentage:

Band	Reference price used
Static	Resolved opening price at startup — persisted last price if available, else `last_buy_price`, else `last_sell_price` (converted to ticks)
Dynamic	Last executed trade price in the current session

The dynamic band is skipped entirely when no trade has occurred yet in the session.

```
static_upper  = int(reference_price × (1 + static_band_pct))
static_lower  = int(reference_price × (1 - static_band_pct))

dynamic_upper = int(last_trade_price × (1 + dynamic_band_pct))
dynamic_lower = int(last_trade_price × (1 - dynamic_band_pct))
```

An incoming order price is rejected if it falls outside *either* band. The static band is checked first.

Tick-based prices

All prices in EduMatcher are stored as integer tick counts. The collar boundaries are in ticks, not display prices. See The Configuration Workflow for the relationship between ticks and display prices.

16.3.3 Validation logic

```
validate_collar(price, collar, last_trade_price) → CollarResult
```

1. If `price < static_lower` or `price > static_upper` → rejected with reject code `COLLAR_BREACH` and a reason starting `STATIC_COLLAR_BREACH:`
2. If `last_trade_price` is known and `price < dynamic_lower` or `price > dynamic_upper` → rejected with reject code `COLLAR_BREACH` and a reason starting `DYNAMIC_COLLAR_BREACH:`
3. Otherwise → accepted

The dynamic check is skipped when `last_trade_price` is `None` (no trade has occurred yet in this session).

16.3.4 Which orders are checked

Collars apply to **LIMIT and ICEBERG** orders (any order that carries an explicit price). MARKET orders do not carry a price and are therefore not subject to collar validation.

The check covers every way a priced order enters: a new order, an amend's new price, and each leg of an OCO or combo. A leg outside the collar rejects the whole OCO (`oco.ack`) or combo (`combo.ack`), with the reason prefixed `Leg <n>:`, and no leg reaches the book. A market maker's two-sided quote, and combos seeded from `market_maker_combos`, are not collared.

16.3.5 Configuration

Add a `collar` sub-section to any symbol in `engine_config.yaml`:

```yaml
symbols:
  MSFT:
    tick_decimals: 2
    last_buy_price: 420.00
    collar:
      static_band_pct: 0.20   # ±20% from reference price (default)
      dynamic_band_pct: 0.02  # ±2% from last traded price (default)
```

Both fields are optional. When the `collar` key is absent entirely, no collar is applied to that symbol. When the `collar` key is present but empty (`{}`), the defaults above are used.

16.3.6 Global level profiles (L1/L2/L3 style)

Collar values can be sourced from reusable global risk levels:

```
risk_controls:
  default_level: L2
  levels:
    L1:
      collar:
        static_band_pct: 0.30
        dynamic_band_pct: 0.05
    L2:
      collar:
        static_band_pct: 0.20
        dynamic_band_pct: 0.02

symbols:
  AAPL:
    # inherits L2
    tick_decimals: 2
  TSLA:
    level: L1
    # override dynamic band only
    collar:
      dynamic_band_pct: 0.06
```

Resolution is **per field**, not per whole block: a symbol's `collar` fields are merged on top of its level's `collar` fields, so a symbol can override just one of the two percentages and still inherit the other from its level. For a single field, the precedence is:

1. `symbols.<symbol>.collar.<field>` (symbol override)
2. `symbols.<symbol>.level` profile from `risk_controls.levels`
3. `risk_controls.default_level` profile
4. built-in defaults (`static_band_pct=0.20`, `dynamic_band_pct=0.02`)

Parameter	Type	Default	Meaning
`static_band_pct`	float	`0.20`	Maximum distance from reference price, as a fraction ($0 < x < 1$)
`dynamic_band_pct`	float	`0.02`	Maximum distance from last trade price, as a fraction ($0 < x < 1$)

The reference price for the static band is resolved at engine startup, preferring a persisted last price from `book_stats.json`, then the configured `last_buy_price`, then `last_sell_price` (converted to ticks). The circuit breaker seeds its reference from the same resolved value, so both controls agree at the open — see Day one (IPO) behaviour.

16.4 Order size and notional limits

16.4.1 Motivation

A collar catches an order whose *price* is wrong. It cannot catch an order whose price is perfectly reasonable and whose *size* is not: "buy 1,000,000 shares at 150.00" passes every price band, because 150.00 is exactly where the stock trades. Two independent caps close that gap.

`max_order_qty` The largest quantity a single order may carry. Catches the fat-finger that adds a zero to the size field.

`max_order_value` The largest notional — `quantity × price`, in display money — a single order may carry. Catches the same error expressed as a plausible-looking size on an expensive instrument.

Both are **pre-trade** checks on the admission path, evaluated before the order reaches the book, and both reject rather than truncate: an order over a cap is never silently reduced to fit.

16.4.2 Which orders are checked

`max_order_qty` applies to **every** order type. `max_order_value` needs a price, so it is skipped for **MARKET** and **IOC** orders, which carry none on the wire — the same orders the collar's price bands already skip, for the same reason.

Both caps are re-checked on `order.amend`, against the amended quantity and price. Without that, the control would be trivially bypassed by entering a small order and amending it up.

The quantity cap is evaluated first, so an order that breaches both is rejected with `MAX_ORDER_QTY`.

16.4.3 Reject codes

Condition	`reject_code`
`quantity > max_order_qty`	`MAX_ORDER_QTY`
`quantity × price > max_order_value`	`MAX_ORDER_VALUE`

16.4.4 Configuration

Add an `order_limits` sub-section to any symbol:

```
symbols:
  AAPL:
    tick_decimals: 2
    order_limits:
      max_order_qty: 50000
      max_order_value: 2500000
  TSLA:
    tick_decimals: 2
    order_limits:
      # a quantity cap only; notional is left unlimited
      max_order_qty: 5000
  MSFT:
    tick_decimals: 2
    # no order_limits at all — neither cap is enforced for MSFT
```

Parameter	Type	Default	Meaning
`max_order_qty`	int	*none*	Maximum quantity on a single order (`> 0`)
`max_order_value`	float	*none*	Maximum notional on a single priced order, in display money (`> 0`)

`pm-config-gen` can write both, either with the explicit per-symbol flags or as `--symbol-opts` keys:

```
pm-config-gen --symbols AAPL MSFT --participants TRADER01 \
  --symbol-max-order-qty AAPL:50000 \
  --symbol-max-order-value AAPL:2500000
```

Order limits are configured per symbol and nowhere else. There is no risk-level tier and no global default, so the rule is simply:

1. `symbols.<symbol>.order_limits.<cap>` — the cap applies
2. absent — the cap is **not enforced**

Note how this differs from collars. A collar can be defined once on a `risk_controls` level and shared by every symbol that references it, because a percentage band means the same thing on a $10 stock and a $1,000 one. A size or notional cap does not: the right number depends on the individual instrument's price and typical trade size, so there is nothing useful for a shared profile to say. `risk_controls.levels.<L>.order_limits` is therefore **rejected** — by the engine loader at startup and by `pm-cverifier` as check `S117` — rather than silently accepted and ignored.

Each cap is independent: `TSLA` above caps quantity while leaving notional unlimited.

16.4.5 Querying the effective limits

`GET /api/v1/reference/symbols` reports each symbol's caps alongside its collar:

```
{
  "symbol": "TSLA",
  "tick_decimals": 2,
  "level": "L1",
  "order_limits": { "max_order_qty": 5000 }
}
```

A symbol that sets neither cap omits `order_limits` entirely — see REST API reference.

16.5 Circuit breakers

16.5.1 Motivation

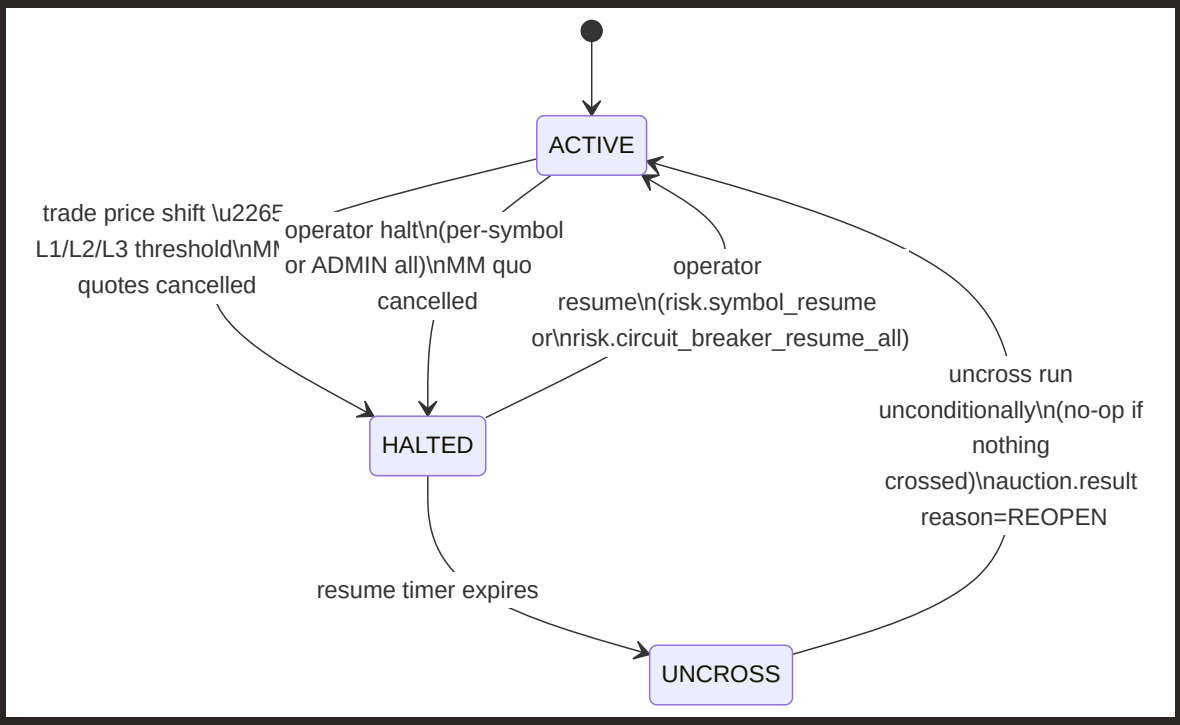
A circuit breaker is an *automatic* mechanism that pauses trading when prices move too fast. Unlike a collar (which rejects individual orders), a circuit breaker monitors executed trades and halts the *entire symbol* when a trade deviates too sharply from recent history. It then automatically resumes after a configurable pause, giving participants time to update their orders.

Real-world examples: the US market-wide circuit breakers (Level 1/2/3), the London Stock Exchange's Automated Auction Call mechanism, and per-stock volatility interruptions on Euronext.

In many production exchanges, the primary circuit-breaker trigger is linked to an index (market-wide reference) with per-symbol volatility controls layered on top. EduMatcher does not yet implement index-level circuit-breaker triggers, so its current circuit-breaker logic is symbol-linked: each symbol evaluates its own rolling trade reference and can halt independently.

16.5.2 How it works — step by step

1. **Every trade is recorded.** When the engine publishes a fill, it calls `record_trade(price, now)` on the circuit breaker for that symbol.
2. **Reference price is computed from history.** The engine looks back over a rolling time window (`reference_window_ns`) and averages the prices of trades that occurred within that window — *excluding the new trade being evaluated.* This ensures the check is "does this trade deviate from where the market has been?" rather than including the potentially erroneous trade in its own reference. On day one, before any real trade exists, this history is pre-seeded with a single baseline point derived from the symbol's opening reference price, so even the very first order can be evaluated (see Day one (IPO) behaviour).
3. **Deviation is tested against levels.** The absolute shift from reference is compared against configured levels (for example `L1=7%`, `L2=13%`, `L3=20%`). The highest crossed level fires.
4. **Symbol is halted.** The engine sets `_halted_symbols[symbol] = True`, cancels all outstanding market-maker quotes for that symbol, and broadcasts a `circuit_breaker.halt.{symbol}` message over the pub socket.
5. **Resume is scheduled by level.** The fired level defines the halt length. Example: L1 might halt for 5 minutes, L2 for 15 minutes, and L3 for the rest of the trading day.
6. **Engine polls for resumption.** Each iteration of the main event loop calls `_flush_circuit_breakers()`, which checks `should_resume(now)` for every active circuit breaker. When the pause expires:
   - The symbol is un-halted.
   - The engine runs an uncross (`_run_uncross()`, the same equilibrium-price algorithm used for scheduled auctions) for that symbol before continuous matching resumes, so interest that crossed while resting during the halt is matched at a fair equilibrium price rather than starting continuous trading in a crossed state. If nothing crossed, the uncross is a no-op.
   - An `auction.result.{symbol}` with `reason: "REOPEN"` is broadcast.
   - A `circuit_breaker.resume.{symbol}` message is broadcast, carrying `halt_source`.



16.5.3 Rolling reference window

The reference window is a sliding time window. Any trade older than `now - reference_window_ns` is discarded before computing the average. This means the reference tracks recent price behaviour — a slow steady trend will not accumulate stale data that masks a sudden move.

```
          reference_window_ns
        ←──────────────────→
──●──●──●──●──●──●──●──●──●── now
  (old trades)  (new trade being evaluated)

reference = average of all trades in window (excluding new trade)
```

If the window is empty, the circuit breaker has no reference to compare against, so the trade is accepted and simply added to the history. At engine startup this empty-window case is avoided by seeding a baseline reference from the symbol's opening price (see Day one (IPO) behaviour). It can otherwise only recur mid-session if the seed and every real trade have aged out of the window during a long quiet period.

16.5.4 Day one (IPO) behaviour

Introducing a new symbol is the exchange equivalent of an **IPO**: there is no trade history yet, only the opening reference price established during listing (`last_buy_price` / `last_sell_price`). Two things happen at startup so that the risk controls are active from the very first order rather than lying dormant until trades accumulate:

- **Collar reference** — the static band anchors on the resolved opening price (persisted `book_stats.json` if present, otherwise the configured `last_buy_price`, then `last_sell_price`).
- **Circuit-breaker reference** — the breaker's rolling history is pre-seeded with a single synthetic baseline point at that **same** resolved opening price. The first continuous trade is therefore evaluated against the IPO reference, so a violent first print can trip the breaker on day one. Auction prints (opening, closing, reopening, recovery) enter the history but never trip the breaker themselves.

Because both controls draw their reference from the same source, the collar and the circuit breaker agree at the open.

Notes and edge cases:

- The seed is a *baseline for comparison only*. It is not published as a trade and does not appear in market data.
- The synthetic seed sits in the rolling window like any trade, so it ages out after `reference_window_ns`. Under normal continuous trading, real fills have replaced it long before then; it matters only if a symbol stays completely silent for the

whole window after the open.
- If a symbol has **no** opening reference at all (no persisted stat and neither `last_buy_price` nor `last_sell_price` set), there is nothing to seed: the collar stays inactive and the circuit breaker has no reference until trades build up. This is why an opening reference price is required when listing a symbol — see Configuration - Symbol Universe.

16.5.5 Why a halt always reopens with an uncross

A halt is not a pause with the book frozen — it is the **call phase of a reopening auction**. 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, `_flush_circuit_breakers()` calls `_run_uncross()` for that symbol — the same equilibrium-price algorithm used for scheduled auctions — and publishes an `auction.result.{symbol}` carrying `reason: "REOPEN"` before the resume event.

For a timed (circuit-breaker or level-named operator) halt this is unconditional and there is no setting that skips it. (A plain operator `risk.symbol_resume` / `risk.circuit_breaker_resume_all` only clears the halt and does not uncross.) Crossed interest accumulates for the whole halt, so resuming straight into continuous matching would begin on a crossed book. If nothing crossed, `compute_equilibrium()` finds no equilibrium price and the uncross is a no-op, indistinguishable from an immediate continuous resume.

Earlier versions exposed a per-level `resumption_mode` (`AUCTION` or `CONTINUOUS`). It never changed engine behaviour — both values uncrossed — and has been removed. The halt and resume payloads now carry `halt_source` (`CB` or `ADMIN`) instead, which says what caused the halt; whether it ends by itself is already expressed by the presence of `resume_at_ns`.

16.5.6 Automated Corridor Expansion (ACE)

Reopening at *whatever* price the accumulated interest implies has two problems, and ACE addresses both.

Problem one: the reopen price can be absurd. A halt fires precisely when prices are moving violently. The orders that pile up during the call phase are entered under uncertainty, often thinly, and the equilibrium they imply can sit far from any price the market would sustain. Printing it turns a protective halt into the cause of an erroneous execution.

Problem two: the reopen instant is a target. If everyone knows the exact nanosecond the uncross runs, the last order in — placed with full sight of the book, too late for anyone to react — is strictly advantaged. This is why real venues randomise the end of a call phase.

ACE Is modelled after the Deusche Börse Xetra. The mechanism is publically described in Market Model for the Trading Venue Deutsche Börse Xetra

It has two independent halves.

The corridor and the expansion ladder

When a halt begins, the engine latches a **corridor reference**: the circuit breaker's rolling mean at the moment of the halt (`CircuitBreakerState. corridor_reference`). It is latched, not recomputed, so an uncross elsewhere cannot move the target mid-halt. The corridor is that reference plus and minus `initial_band_pct` of it.

At the end of every call phase the engine runs `compute_equilibrium()` as a **dry run** — it computes the price the symbol *would* reopen at without executing anything:

- **Inside the corridor** → the symbol reopens. `_run_uncross()` executes at the equilibrium and `circuit_breaker.resume.{symbol}` is published.
- **Outside the corridor** → the symbol does **not** reopen. The corridor widens by the next rung's `widen_pct`, a fresh call phase of that rung's `min_duration_ns` begins, and `circuit_breaker.extend.{symbol}` is published. Orders keep resting throughout; nothing is cancelled.

Widening is **additive on the reference price**, not compounding on the previous width. Each rung adds its `widen_pct` of the *original* reference. This matters: it is what makes the corridor grow linearly and predictably rather than exploding.

The last rung repeats indefinitely. This is the design's terminating argument, and it is why there is no maximum-extensions setting. Because the corridor grows without bound, it eventually contains any finite price, so the symbol always reopens on its own — the only question is when. A cap would force a choice between printing outside the corridor (defeating the purpose) and never reopening at all.

The random end

`halt_duration_ns` on the triggered level is the **minimum** length of the first call phase, not its exact length. `min_duration_ns` plays the same role for each extension. On top of the minimum the engine adds a delay drawn uniformly from `[0, random_end_max_ns]`, so no call phase — initial or extended — ends at a time anyone can predict.

The generator is engine-wide (`EngineProcess._reopening_rng`). Leave `random_seed` unset for OS entropy, which is what an operating venue wants. Set it to an integer for reproducible teaching demos and tests. Setting `random_end_max_ns: 0` disables the random end entirely, making reopen times exactly predictable — useful in a classroom, wrong in production.

Worked example

Configuration: `initial_band_pct: 0.10`, expansions `[{0.10, 2min}, {0.20, 5min}]`, `random_end_max_ns: 0` (so the timings below are exact). Symbol ABC, reference price **$100.00**, an L1 halt with `halt_duration_ns` of 5 minutes fires at **13:30:00**.

Heavy one-sided buying accumulates during the call; the indicative price settles at **$122.00** and stays there.

Time	Event	Indicative	Corridor	Half-width	Outcome
13:30:00	L1 halt fires	–	90.00 – 110.00	±10%	Call phase 1 opens (min 5 min)
13:35:00	Call 1 ends	$122.00	90.00 – 110.00	±10%	**Outside** → extend, widen +10%
13:37:00	Call 2 ends	$122.00	80.00 – 120.00	±20%	**Outside** → extend, widen +20%
13:42:00	Call 3 ends	$122.00	60.00 – 140.00	±40%	**Inside** → reopen, uncross at $122.00

Total halt: 12 minutes rather than the configured 5. The extra 7 minutes are the market being given time to supply offsetting interest against a 22% move before it prints. Had sellers arrived during call 2 and pulled the indicative back to $118, the symbol would have reopened at 13:37 inside the ±20% corridor.

These are exactly the numbers in the SEC order approving Nasdaq's rule ($100 reference, collars 90/110 → 80/120 → 60/140), which is a useful cross-check that the arithmetic is right.

With the random end at its default 30s, the three call ends above would instead fall somewhere in 13:35:00–13:35:30, 13:37:00–13:37:30 and 13:42:00–13:42:30.

End of day: the closing auction as backstop

ACE widens indefinitely, so on its own it never terminates — a symbol whose indicative price runs away could in principle extend past the close. The end of the trading day supplies the terminating condition.

On the transition to `CLOSED`, `_run_closing_backstop()` forces every halted symbol that has a circuit breaker configured to resolve:

1. The indicative price is computed one final time.
2. **Inside the corridor** → it prints there, as a normal reopen would.
3. **Outside the corridor** → it prints **at the corridor boundary**: the upper bound for a buy imbalance, the lower bound for a sell imbalance.
4. **No crossing interest at all** → nothing prints; the halt is simply cleared.

Step 3 is the only place in the engine where a price is **imposed rather than discovered**, and it is deliberate. A clamped print can leave the book crossed — bids and asks beyond the boundary do not trade. That is the intended outcome: that interest survives to the next session rather than executing at a price the corridor was built to reject.

The ordering matters and is load-bearing. The backstop runs *after* the scheduled `CLOSING_AUCTION → CLOSED` uncross, and `_run_uncross()` skips symbols that are still halted. Were it otherwise, the session sweep would uncross halted symbols at the true equilibrium and silently undo the entire mechanism.

A level configured with `halt_duration_ns: null` (rest-of-day) never enters the ACE cycle at all: it has no timed resume, so no call phase ever ends. It waits for this backstop or for an ADMIN resume.

Observability

Every corridor adjustment is logged and published, so an extension sequence can be watched as it happens:

```
CIRCUIT BREAKER HALT ABC: level=L1 trigger=12200, ref=10000 ticks, corridor=[9000, 11000] ticks (+/-10.0%)
ACE EXTEND ABC: indicative=12200 ticks outside [9000, 11000] -> expansion=1 corridor=[8000, 12000] (+/-20.0%) qty=500 next_call_ends=.
ACE EXTEND ABC: indicative=12200 ticks outside [8000, 12000] -> expansion=2 corridor=[6000, 14000] (+/-40.0%) qty=500 next_call_ends=.
CIRCUIT BREAKER RESUME ABC: after 2 ACE extension(s)
```

and at the close:

```
CLOSING BACKSTOP ABC: indicative=12200 ticks outside [8000, 12000] -> clamped to 12000 (BUY imbalance), after 1 ACE extension(s)
```

On the wire, `circuit_breaker.halt.{symbol}` and the new `circuit_breaker.extend.{symbol}` both carry `corridor_low`, `corridor_high` and `expansion`; `extend` adds `indicative_price`, `indicative_qty`, `imbalance_side` and the next `resume_at_ns`. On `extend` the three corridor fields are always present — the event is published by the widening itself. On `halt` all three are **omitted** when the halt has no corridor: either ACE is disabled, or the halt began with no reference price to centre one on, or it is an operator halt. A backstop resume carries `reason: "CLOSING_BACKSTOP"`, `clamped` and `print_price`.

16.5.7 Configuration

Define a global threshold ladder under `circuit_breaker_defaults`, then override per symbol only where needed:

```yaml
circuit_breaker_defaults:
  reference_window_ns: 300000000000
  levels:
    L1:
      price_shift_pct: 0.07
      halt_duration_ns: 300000000000   # 5 minutes
    L2:
      price_shift_pct: 0.13
      halt_duration_ns: 900000000000   # 15 minutes
    L3:
      price_shift_pct: 0.20
      halt_duration_ns:                 # null => rest of trading day
  reopening:                            # Automated Corridor Expansion
    enabled: true
    initial_band_pct: 0.10              # +/-10% corridor to start
    random_end_max_ns: 30000000000      # up to 30s random tail per call
    random_seed:                        # null => OS entropy
    expansions:                         # last rung repeats indefinitely
      - widen_pct: 0.10
        min_duration_ns: 120000000000   # 2 minutes
      - widen_pct: 0.20
        min_duration_ns: 300000000000   # 5 minutes

symbols:
  TSLA:
    tick_decimals: 2
    circuit_breaker:
      levels:
        L1:
          halt_duration_ns: 600000000000  # symbol-specific override
      reopening:
        initial_band_pct: 0.05           # tighter corridor for TSLA only
```

For each trade, the engine computes:

$$\text{price_shift} = \frac{|\text{trade_price} - \text{reference_price}|}{\text{reference_price}}$$

The highest level where `price_shift >= price_shift_pct` fires.

Parameter	Type	Default	Meaning
`reference_window_ns`	int	`300_000_000_000`	Lookback window for rolling reference price
`levels.<L>.price_shift_pct`	float	required	Trigger threshold fraction in `(0, 1)`
`levels.<L>.halt_duration_ns`	int or null	optional; omitted or `null` = rest of day	*Minimum* length of the reopening call phase in ns, or `null` for rest-of-day
`reopening.enabled`	bool	`true`	Apply ACE. When false, a halt reopens at the equilibrium price uncollared
`reopening.initial_band_pct`	float	`0.10`	Corridor half-width in `(0, 1)`, as a fraction of the reference price
`reopening.expansions`	list	Nasdaq ladder	Rungs of `{widen_pct, min_duration_ns}`. **The last rung repeats indefinitely**
`reopening.expansions[].widen_pct`	float	required	Added to the corridor half-width, in `(0, 1)`. Additive on the reference, not compounding
`reopening.expansions[].min_duration_ns`	int	required	Minimum length of that extension's call phase, `> 0`
`reopening.random_end_max_ns`	int	`30_000_000_000`	Upper bound of the uniform random tail on every call phase. `0` disables it
`reopening.random_seed`	int or null	`null`	Engine-wide. **Only valid in `circuit_breaker_defaults`** — setting it per symbol is an error (`S110`)

Two of these are exchange-wide and rejected on a symbol: `random_seed` (`S110`) and `expansions` (`S112`). The split is deliberate. `initial_band_pct` answers *how volatile is this instrument normally* — a thin small-cap legitimately needs a wider reopening corridor than a liquid blue chip, so it varies per symbol. The ladder answers *how quickly does the exchange stop protecting the price and let it through*, which is venue policy rather than an instrument property; keeping it uniform is what makes halt durations comparable across the book.

The line is not perfect — `min_duration_ns` lives in the ladder and is arguably instrument-shaped, since a thin name may need longer call phases for liquidity to arrive. Real venues disagree on where to draw it: Nasdaq varies nothing per security (the collar rule is universal; only the reference price differs), while Deutsche Börse publishes corridor widths *and* durations per instrument in reference data. EduMatcher sits between them, and enforces the choice rather than leaving it to whichever tool touched the file last.

When `circuit_breaker_defaults` and symbol-level `circuit_breaker` are both present, per-symbol values override global defaults by level key. The `reopening` block merges field-by-field the same way, so a symbol can override `initial_band_pct` alone without restating the ladder.

16.5.8 Why there is no selected "default breaker level"

Circuit-breaker levels are **trigger outcomes**, not configuration profiles to pick one from. A symbol defines a ladder of levels; the observed price shift determines which level is activated at runtime.

16.6 Price collars vs circuit breakers

Both controls are configured on symbols, but they protect the market at different points in the flow and for different failure modes.

- A **price collar** is a pre-trade admission guardrail on priced orders.
- A **circuit breaker** is a post-trade volatility interrupt that can halt the full symbol after an extreme move is observed.

16.6.1 Side-by-side comparison

Dimension	Price collar (symbol)	Circuit breaker (symbol)
Trigger moment	Before matching (order admission)	After a trade is executed
Data checked	Incoming order price vs static/dynamic bands	Trade price shift vs rolling reference
Scope of effect	Single incoming order	Entire symbol
Typical action	Reject offending order (code `COLLAR_BREACH`, reason `STATIC_COLLAR_BREACH: …` or `DYNAMIC_COLLAR_BREACH: …`)	Halt symbol, cancel MM quotes, schedule/manual resume
Market state after trigger	Symbol keeps trading for other valid orders	Symbol is halted until resume condition is met
Configuration anchor	`symbols.<SYM>.collar` (or inherited level defaults)	`circuit_breaker_defaults` + `symbols.<SYM>.circuit_breaker` overrides
Primary objective	Prevent fat-finger / outlier order entry	Pause disorderly market after extreme realized move
Dependence on trade history	Dynamic band needs last trade, static band does not	Uses rolling trade history/reference window

16.6.2 Practical interpretation

Use collars to stop clearly invalid prices from entering the book. Use circuit breakers to pause trading when validly admitted orders still produce an abnormally large executed move. In practice, collars reduce bad inputs and circuit breakers contain fast market dislocations.

For field-level schema and merge precedence details, see Configuration - Circuit Breakers and Configuration - Risk Controls and Collars.

16.7 Interaction between mechanisms

All the admission-path controls can be active simultaneously on the same symbol. The engine applies them in this order for every incoming order:

```
1. Order size / notional limits    → reject MAX_ORDER_QTY / MAX_ORDER_VALUE
2. Session state (if enabled)      → reject "Market is closed" outside order-accepting phases
3. Is the symbol halted?           → reject MARKET/FOK/IOC; suppress matching for LIMIT
4. Is a collar configured?         → validate price against static and dynamic bands
5. (After match) Did a trade fire
   a circuit breaker?              → halt symbol, schedule resume
```

Because limits run first, an oversized MARKET order on a halted symbol is rejected with `MAX_ORDER_QTY`, not with the halt reason.

A circuit breaker halt feeds back into step 1: any subsequent orders on a circuit-breaker-halted symbol are subject to the same halt rules as an operator-initiated halt.

16.7.1 Combined example

```
Opening reference price: 10000 ticks (100.00 in display)
  → also seeds the CB's rolling trade_history as a single baseline point
Static collar:   ±20%  → [8000, 12000] ticks
Dynamic collar:  ±10%  → depends on last trade
CB levels:       L1=7%, L2=13%, L3=20%

A limit sell order arrives at price 7500 ticks:
  → Static collar: 7500 < 8000 → STATIC_COLLAR_BREACH → rejected

A limit buy at 10100 ticks trades (no trade yet, so only the static band applies).
  → CB reference = history average BEFORE this trade = 10000 (seed only)
  → deviation = |10100 - 10000| / 10000 = 1.0% < L1 (7%) → no halt
  → history becomes [10000, 10100] (sum=20100, len=2)

A limit buy at 10700 ticks trades.
  → CB reference = 20100 // 2 = 10050 (integer-tick average)
  → deviation = |10700 - 10050| / 10050 ≈ 6.5% < L1 (7%) → no halt
  → history becomes [10000, 10100, 10700] (sum=30800, len=3)

A limit buy at 11000 ticks trades.
  → CB reference = 30800 // 3 = 10266 (floor division)
  → deviation = |11000 - 10266| / 10266 ≈ 7.1% ≥ L1 (7%)
  → L1 circuit breaker fires, symbol halted for L1 duration (5 min default);
    the timed resume runs a reopening uncross for the symbol

Separately, suppose the collars have been widened (say static ±30%, dynamic
±25%) so that a 12200-tick order can reach the matching engine at all, and a
later session's rolling reference has settled at 10100 when a limit buy at
12200 ticks trades:
  → deviation = |12200 - 10100| / 10100 ≈ 20.8% ≥ L3 (20%)
  → L3 circuit breaker fires, symbol halted for rest of trading day
  → next order at this symbol: if MARKET → rejected; if LIMIT → accepted, no match
```

Reference price is an integer-tick rolling average

`CircuitBreakerState.record_trade()` computes the reference as `sum(prices in window) // len(prices in window)` — floor (`//`) integer division, consistent with all prices being stored as integer ticks. The reference used for a given trade always excludes that trade itself; it is only added to the rolling history *after* the deviation check.

What the gateway operator sees when a collar rejects an order:

```
[TRADER01]> NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=75.00
[14:02:00.301] REJECTED  <order-id> code=COLLAR_BREACH  STATIC_COLLAR_BREACH: price 7500 ticks is outside [8000, 12000] ticks (±20% f
```

16.8 ZeroMQ messages

Risk events are broadcast on the engine's PUB socket (port 5556).

16.8.1 Circuit breaker halt

```
topic:   b"circuit_breaker.halt.MSFT"
payload: {
    "symbol":           "MSFT",
    "trigger_price":    108.00,   # display money, not ticks
    "reference_price":  101.00,
    "resume_at_ns":     <timestamp>,
    "halt_source":      "CB",
    "level":            "L1",
    "corridor_low":     90.00,    # all three omitted when there is
    "corridor_high":    110.00,   # no corridor — see above
    "expansion":        0
}
```

16.8.2 Circuit breaker extend (ACE)

Published when a call phase ends with the indicative price outside the corridor. The symbol stays halted and a fresh call phase begins.

```
topic:   b"circuit_breaker.extend.MSFT"
payload: {
    "symbol":           "MSFT",
    "indicative_price": 122.00,   # what it would have reopened at
    "indicative_qty":   500,
    "imbalance_side":   "BUY",   # omitted entirely when balanced
    "corridor_low":     80.00,    # corridor AFTER widening
    "corridor_high":    120.00,
    "expansion":        1,        # rungs consumed so far
    "resume_at_ns":     ...       # end of the new call phase
}
```

16.8.3 Circuit breaker resume

```
topic:   b"circuit_breaker.resume.MSFT"
payload: {
    "symbol":      "MSFT",
    "halt_source": "CB"
}
```

A resume forced by the end-of-day backstop carries three extra fields:

```
payload: {
    "symbol":      "MSFT",
    "halt_source": "CB",
    "reason":      "CLOSING_BACKSTOP",
    "clamped":     true,          # printed at the corridor boundary
    "print_price": 110.00
}
```

This detail is also on the CALF market-data wire

`pm-md-gwy` (the CALF gateway) exposes this same payload to external clients on a dedicated `CB` channel — `trigger_price`, `reference_price`, `level`, `resume_at_ns`, and `halt_source` all reach the wire, alongside the coarse `SESSION=HALTED` transition CALF's `STATE` channel already carried. See CALF Protocol Reference — `CB` for the field mapping (`halt_source` is carried as `SRC`). The reopening uncross (`auction.result.{SYMBOL}`) reaches CALF's `AUCTION` channel with `REASON=REOPEN`, which is what distinguishes it from the scheduled opening and closing auctions.

16.9 ADMIN-role operator controls

An operator gateway configured with `role: ADMIN` can trigger and lift an **exchange-wide** halt without waiting for a per-symbol circuit breaker to fire or expire. This is the manual emergency control used when a venue-wide technology incident,

regulatory instruction, or extreme market dislocation requires every symbol to be frozen simultaneously.

16.9.1 Prerequisites

Configure a dedicated gateway with `role: ADMIN` in `engine_config.yaml`:

```
participants:
  - id: GW_ADMIN
    description: "Operations desk"
    role: ADMIN
    disconnect_behaviour: CANCEL_QUOTES_ONLY
```

The gateway connects to the engine via the standard PUSH socket (port 5555) and subscribes to the PUB socket (port 5556) to receive ack messages.

See Role Privileges for the full permissions matrix.

Not available from the ALF terminal

The ALF console (`pm-alf-console`) only exposes `KILL` for risk actions — it has no `HALT`/`RESUME`/`CANCEL_SYM` command. To trigger an exchange-wide halt, a per-symbol halt/resume, or a symbol-level mass cancel, use the ADMIN operator console `pm-admin --id GW_ADMIN` (`HALT`, `RESUME`, `HALT_SYM|SYM=`, `RESUME_SYM|SYM=`, `CANCEL_SYM|SYM=`, `REOPEN|SYM=[|PRICE=][|DRY_RUN=1]`, `KILL|GW=[|SYM=]`), or send the raw ZMQ frames shown below directly. The per-symbol halt/resume and the symbol-level mass cancel are also reachable over REST (see API Gateway); the **exchange-wide** `risk.circuit_breaker_halt_all` / `risk.circuit_breaker_resume_all` have no REST endpoint — `pm-admin` or a raw ZMQ frame are the only ways to reach them.

16.9.2 Triggering an exchange-wide halt

Send the following frame to the engine's PUSH socket (port **5555**):

```
Frame 0 (topic):   b"risk.circuit_breaker_halt_all"
Frame 1 (payload): {"gateway_id": "GW_ADMIN"}
```

Python snippet using `pyzmq`:

```
import zmq, json

ctx = zmq.Context()
push = ctx.socket(zmq.PUSH)
push.connect("tcp://localhost:5555")

push.send_multipart([
    b"risk.circuit_breaker_halt_all",
    json.dumps({"gateway_id": "GW_ADMIN"}).encode(),
])
```

What the engine does:

1. Verifies `GW_ADMIN` is connected and carries role `ADMIN`.
2. Collects every known symbol (order books, circuit-breaker state, engine configuration).
3. Marks each symbol `HALTED` with `halt_source = "ADMIN"` (no auto-resume timer).
4. Cancels all outstanding market-maker quote legs for every symbol.
5. Publishes one `circuit_breaker.halt.<SYMBOL>` event per symbol on the PUB socket.
6. Sends the ack to the PUB socket.

Expected inbound events (subscribe to `circuit_breaker.*` and `risk.circuit_breaker_halt_all_ack.*`):

```
topic:   b"circuit_breaker.halt.AAPL"
payload: {
    "symbol":          "AAPL",
    "trigger_price":   null,
    "reference_price": null,
    "resume_at_ns":    null,
    "halt_source": "ADMIN",
    "level":           "ADMIN_ALL"
}

topic:   b"circuit_breaker.halt.MSFT"
payload: { ...same structure... }

topic:   b"risk.circuit_breaker_halt_all_ack.GW_ADMIN"
payload: {
    "accepted":        true,
    "reason":          "",
    "halted_symbols":  4,
    "halted_symbol_ids": ["AAPL", "MSFT", "GOOG", "TSLA"],
    "cancelled_quotes": 12,
    "cancelled_quote_order_ids": ["ORD-1", "ORD-2", "..."]
}
```

If the gateway is not connected or does not carry role `ADMIN`, the engine returns `accepted: false` and no symbols are halted:

```
topic:   b"risk.circuit_breaker_halt_all_ack.GW_ADMIN"
payload: {
    "accepted": false,
    "reason":   "Global circuit-breaker halt is only allowed for ADMIN participants"
}
```

16.9.3 Resuming all trading

Once the situation is resolved, send `risk.circuit_breaker_resume_all` on the same PUSH socket:

```
Frame 0 (topic):   b"risk.circuit_breaker_resume_all"
Frame 1 (payload): {"gateway_id": "GW_ADMIN"}
```

Python snippet:

```
push.send_multipart([
    b"risk.circuit_breaker_resume_all",
    json.dumps({"gateway_id": "GW_ADMIN"}).encode(),
])
```

What the engine does:

1. Verifies `GW_ADMIN` is connected and carries role `ADMIN`.
2. Collects every symbol currently marked as halted.
3. Clears the halt flag and deactivates any circuit-breaker state for each symbol.
4. Publishes one `circuit_breaker.resume.<SYMBOL>` event per symbol.
5. Sends the ack.

Expected inbound events:

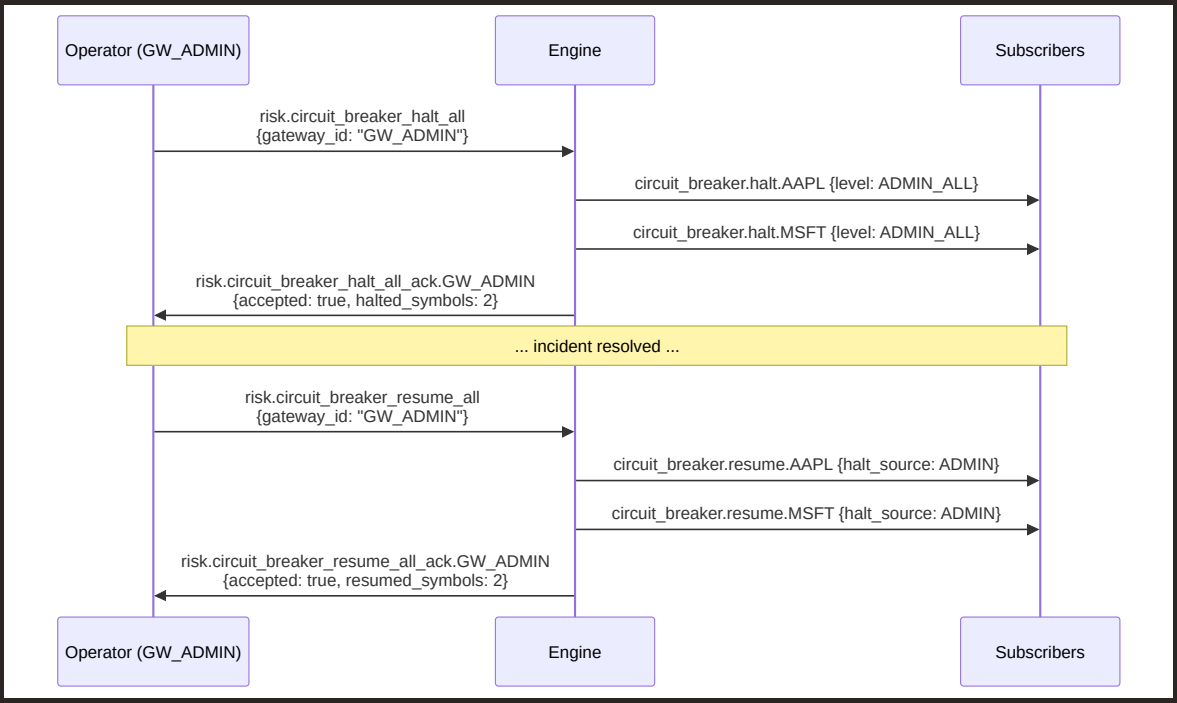
```
topic:   b"circuit_breaker.resume.AAPL"
payload: { "symbol": "AAPL", "halt_source": "ADMIN" }

topic:   b"circuit_breaker.resume.MSFT"
payload: { "symbol": "MSFT", "halt_source": "ADMIN" }

topic:   b"risk.circuit_breaker_resume_all_ack.GW_ADMIN"
payload: {
    "accepted":         true,
    "reason":           "",
    "resumed_symbols":  4,
    "resumed_symbol_ids": ["AAPL", "MSFT", "GOOG", "TSLA"]
}
```

After the ack is received, normal order flow and quote submission resume for all previously halted symbols. Market makers are expected to re-enter fresh quotes; the engine will begin enforcing MM obligation checks again immediately.

16.9.4 Sequence diagram



16.9.5 Key differences from automatic circuit breakers

Property	Automatic CB (per-symbol)	ADMIN global halt
Trigger	Trade price deviation	Operator command
Scope	Single symbol	All symbols
Resume	Scheduled timer, always via uncross	Explicit `risk.circuit_breaker_resume_all` (or `risk.symbol_resume` for a single symbol)
Quotes cancelled on halt	Yes	Yes
`halt_source`	`CB`	`ADMIN`
Who can send	Any connected gateway	`ADMIN` role only

16.9.6 Halting or resuming a single symbol

Alongside the exchange-wide `risk.circuit_breaker_halt_all` / `risk.circuit_breaker_resume_all` pair, an `ADMIN` gateway can halt or resume **one symbol at a time** using `risk.symbol_halt` / `risk.symbol_resume`. This is the command used by the `/api/v1/admin/symbols/{symbol}/halt` and `/api/v1/admin/symbols/{symbol}/resume` REST endpoints (see API Gateway) and is the mechanism behind the "Per-symbol operator halt" row in the Market-maker interaction table below.

```
Frame 0 (topic):   b"risk.symbol_halt"
Frame 1 (payload): {"gateway_id": "GW_ADMIN", "symbol": "AAPL"}
```

The payload accepts an optional `"level"` naming one of the symbol's configured circuit-breaker levels (e.g. `"L1"`). The halt then runs through the same state machine as a price-triggered halt: it gets a timed call phase and ACE corridor, `halt_source` stays `ADMIN`, and it ends with a reopening uncross. Without `level` the halt is indefinite (`ADMIN_SYMBOL`) and is cleared only by an explicit resume. Extra rejects for `level: "<SYMBOL> has no circuit breaker configured"` and `"Unknown circuit-breaker level for <SYMBOL>: <LEVEL>"`.

What the engine does:

1. Verifies `GW_ADMIN` is connected and carries role `ADMIN`; rejects with reason `"Per-symbol halt is only allowed for ADMIN participants"` otherwise.
2. Marks the symbol `HALTED` (`_halted_symbols[symbol] = True`) and, if a circuit breaker is configured for the symbol, sets its state to `halted` with `triggered_level = "ADMIN_SYMBOL"` and `halt_source = "ADMIN"`.
3. Cancels all outstanding market-maker quote legs for that symbol only, with cancellation reason `"Per-symbol halt"`.
4. Publishes `circuit_breaker.halt.<SYMBOL>` with `"level": "ADMIN_SYMBOL"` and `"halt_source": "ADMIN"`.
5. Sends `risk.symbol_halt_ack.<GW_ADMIN>` with `{"accepted": true, "symbol": "AAPL", "reason": "", "cancelled_quotes": <count>, "cancelled_quote_order_ids": [<the cancelled legs' order ids>]}`.

`risk.symbol_resume` mirrors this (note that an operator resume does **not** run an uncross — use `REOPEN` in `pm-admin` to uncross and clear the halt in one step): it requires `ADMIN` role (rejecting with `"Per-symbol resume is only allowed for ADMIN participants"`), rejects with `"<SYMBOL> is not halted"` if the symbol isn't currently halted, otherwise clears the halt, deactivates the circuit breaker state, publishes `circuit_breaker.resume.<SYMBOL>` with `"halt_source": "ADMIN"`, and acks on `risk.symbol_resume_ack.<GW_ADMIN>` with `{"accepted": true, "symbol": "AAPL", "reason": ""}`.

Both `symbol_halt` and `symbol_resume` also reject with `"symbol required"` if the `symbol` field is missing, and `symbol_halt` additionally rejects with `"Unknown symbol: <SYMBOL>"` if the engine has an allowlist of symbols and the requested symbol isn't in it.

16.9.7 Symbol-level mass cancel

`risk.cancel_symbol` cancels every resting order and quote leg for one symbol, **across all gateways** — not just the sender's own exposure like the kill switch. It does not halt the symbol; trading continues immediately with an empty book (aside from any order that arrives afterward). It is `ADMIN`-only and is the command behind the `/api/v1/admin/symbols/{symbol}/cancel-orders` REST endpoint (see API Gateway).

```
Frame 0 (topic):   b"risk.cancel_symbol"
Frame 1 (payload): {"gateway_id": "GW_ADMIN", "symbol": "AAPL"}
```

What the engine does:

1. Verifies `GW_ADMIN` is connected and carries role `ADMIN`; rejects with reason `"Symbol-level mass cancel is only allowed for ADMIN participants"` otherwise. Also rejects with `"symbol required"` if `symbol` is missing.
2. Cancels every resting order for the symbol that did not originate from a quote leg (`order.origin != OrderOrigin.QUOTE`), across every gateway.
3. Cancels every quote leg resting for the symbol, across every gateway, with cancellation reason `"Symbol mass cancel"`.
4. Sends `risk.cancel_symbol_ack.<GW_ADMIN>` with `{"accepted": true, "symbol": "AAPL", "reason": "", "cancelled_orders": <count>, "cancelled_order_ids": [<ids>], "cancelled_quotes": <count>, "cancelled_quote_order_ids": [<ids>]}`.

Unlike the kill switch (which is scoped to one gateway's own resting exposure) and the instrument halt (which stops the symbol from matching), `risk.cancel_symbol` clears standing interest for a symbol from every participant while leaving the symbol open for new orders.

16.10 Kill switch

The kill switch is a gateway-level emergency control that immediately cancels all resting orders and quotes owned by a specific gateway, without halting the symbol. It is designed for situations where a malfunctioning trading bot or gateway needs to be flushed without stopping the whole market.

16.10.1 Permissions

Unlike the exchange-wide halt, the kill switch does **not** require `ADMIN` role. Any authenticated, connected gateway can trigger a kill switch against its own orders.

16.10.2 Sending the command

```
Frame 0 (topic):   b"risk.kill_switch"
Frame 1 (payload): {"gateway_id": "GW01"}
```

An optional `"symbol"` field scopes the cancellation to a single instrument:

```
{"gateway_id": "GW01", "symbol": "AAPL"}
```

When `"symbol"` is empty or absent, all symbols are included.

16.10.3 What the engine does

1. Collects all resting orders and quote-origin legs for the specified gateway (and optionally symbol), including a partially filled quote's remainder after its quote has become inactive.
2. Cancels each order and quote leg, excluding child orders that were derived from a quote (those are cancelled as part of the quote leg cancellation).
3. Sends a `risk.kill_switch_ack.{GW_ID}` reply with the count of cancelled items and the ids of the orders/quote legs cancelled. Quote-origin remainders are included in `cancelled_quotes` and `cancelled_quote_order_ids`, not in the non-quote order count.

16.10.4 Reply

```
topic:   b"risk.kill_switch_ack.GW01"
payload: {
    "accepted":          true,
    "reason":            "",
    "cancelled_orders":  <count>,
    "cancelled_order_ids": [<the cancelled orders' ids>],
    "cancelled_quotes":  <count>,
    "cancelled_quote_order_ids": [<the cancelled quote legs' order ids>]
}
```

16.10.5 Differences from a halt

Property	Kill switch	Instrument halt
Scope	One gateway's orders	All orders on a symbol
Symbol trading	Continues uninterrupted	Paused
New orders accepted	Yes	LIMIT/ICEBERG only
Requires ADMIN role	No	No when auto-triggered by a circuit breaker; **Yes** for an operator-initiated halt, whether per-symbol (`risk.symbol_halt`) or exchange-wide (`risk.circuit_breaker_halt_all`)
Auto-resume	Not applicable	Yes (CB) / Manual (operator)

Cross-gateway kill switches need ADMIN

A gateway's own kill switch targets only that gateway. An `ADMIN` gateway can additionally cancel another gateway's exposure with `risk.kill_switch_gateway` (REST: `POST /api/v1/admin/gateways/{gateway_id}/kill-switch`), or every resting order and quote on the exchange with `risk.kill_switch_global` (REST: `POST /api/v1/admin/market/kill-switch`); both reject non-ADMIN callers. `pm-admin` exposes `KILL|GW=<gw>[|SYM=<sym>]`. To stop trading rather than clear the book, use a per-symbol `risk.symbol_halt` or an exchange-wide `risk.circuit_breaker_halt_all` (both ADMIN), or `risk.cancel_symbol` (ADMIN, see Symbol-level mass cancel) to clear resting interest for a symbol across every gateway.

16.11 Self-Match Prevention (SMP)

16.11.1 Motivation

A single trading firm often runs several independent strategies or bots against the same market, each connected through its own gateway session — or sometimes several sessions sharing one `gateway_id`. Without a safeguard, one of that firm's aggressive orders can cross against another resting order from the *same* firm. The trade prints, generates real fees and settlement obligations, and often serves no economic purpose: the firm has simply traded with itself. Self-Match Prevention (SMP) is the engine-level control that detects this situation at the moment of matching and takes a configurable, deliberate action instead of silently allowing the self-trade.

SMP is scoped to the **gateway identity** (`gateway_id`), not to individual order IDs or symbols. Two resting orders from `TRADER01` on `AAPL` are a self-match risk; a `TRADER01` order against a `TRADER02` order is not, no matter how similar the strategies behind them are.

16.11.2 The four actions

`SmpAction` is a four-value enum shared by every order-entry path in the system (`NEW`, `COMBO`, and the REST API carry it explicitly per request; `QUOTE` always inherits it from the gateway default — see Which paths carry an explicit per-request `SMP=` below):

Value	Aggressor (incoming order)	Resting order	Net effect
`NONE`	Trades normally	Trades normally	Self-trade is **allowed** — this is "SMP off"
`CANCEL_AGGRESSOR`	Cancelled, no fill against this resting order	Untouched, stays on the book	The incoming order backs off; existing resting interest is preserved
`CANCEL_RESTING`	Continues matching against the next eligible resting order	Cancelled	The older resting order is cleared out of the way; the aggressor's intent to trade is honoured against other participants
`CANCEL_BOTH`	Cancelled	Cancelled	Both sides of the would-be self-match are pulled from the book

`SmpAction` is evaluated **per potential match**, not once per order. A single aggressor sweeping several price levels can encounter same-gateway resting liquidity at one level and other-participant liquidity at another; SMP only engages at the levels where the gateway IDs actually collide.

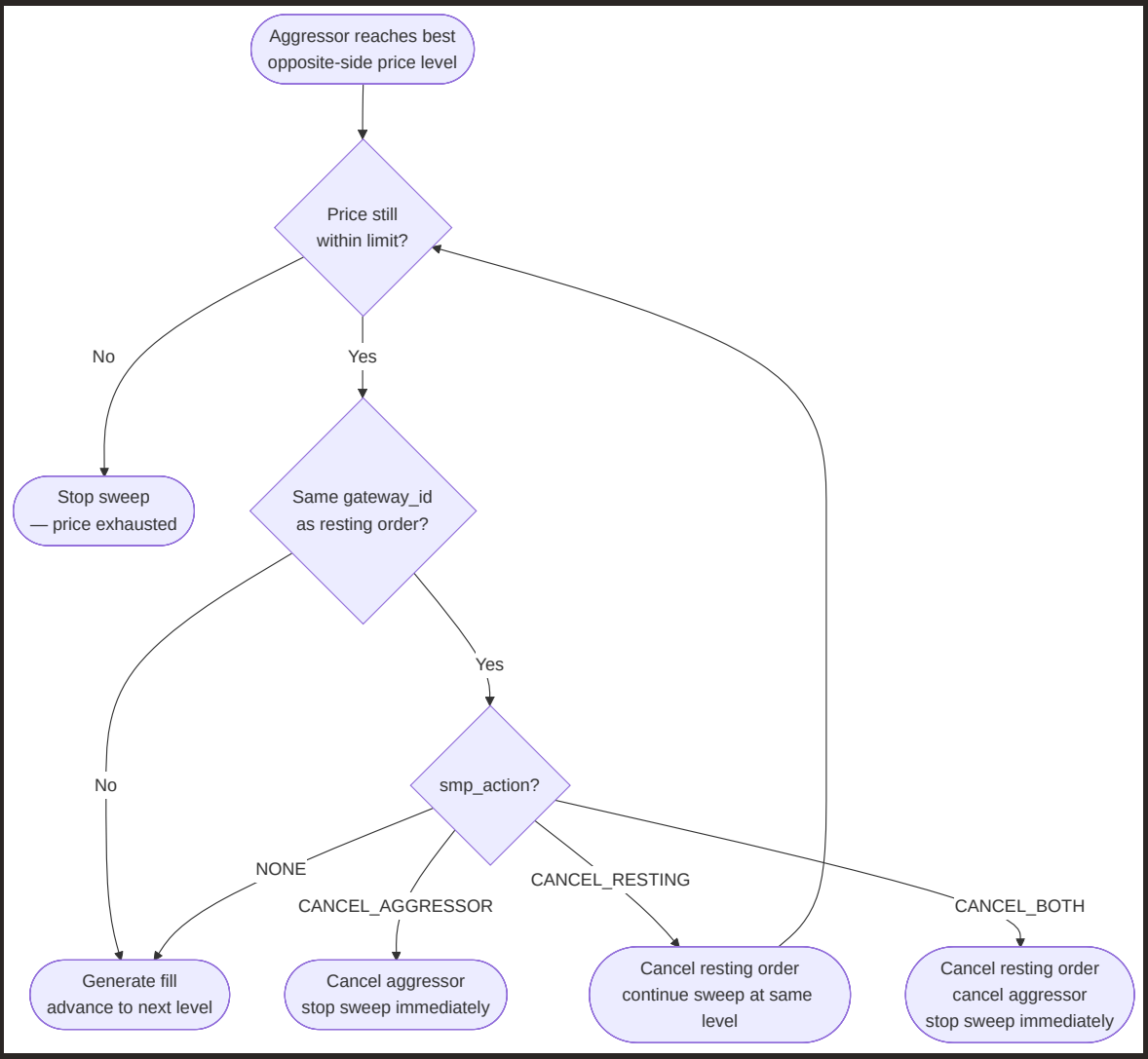
16.11.3 What the client sees

Every order SMP cancels — aggressor, resting order, or both — is reported as a cancellation carrying `CANCEL_REASON=SELF_MATCH_PREVE` on ALF, or `cancel_reason` on the REST WebSocket. It is **not** a rejection: the order was accepted, it may already have traded against other participants at better price levels, and only the part that would have self-matched is pulled.

Without that field an SMP cancel is indistinguishable from a kill switch or a halt cascade, which matters most under `CANCEL_RESTING` — where the order that disappears is one you placed earlier and were not thinking about.

16.11.4 Where SMP is checked — the matching-engine mechanics

SMP is enforced inside the order book's price-time sweep, immediately after the price-limit check and before a fill is generated. This applies uniformly to every order type that sweeps the book: `MARKET`, `LIMIT`, `IOC`, `FOK`, and `ICEBERG` (both as aggressor and, for iceberg-vs-iceberg matches, on the resting side too).



Three consequences follow directly from this being a per-level, in-sweep check rather than a pre-trade validation:

- **`CANCEL_AGGRESSOR` and `CANCEL_BOTH` stop the sweep outright.** If the aggressor had already filled against other participants at better price levels before reaching the self-match, those earlier fills stand — only the remainder is cancelled. The order can therefore end up `PARTIAL` + `CANCELLED`, exactly like a normal IOC/FOK partial-then-cancel outcome.
- **`CANCEL_RESTING` lets the sweep continue.** The resting order that would have caused the self-match is removed from the book, and the aggressor keeps walking the book — it may still fill fully against other participants deeper in the book.
- **A resting `ICEBERG` meets SMP when its displayed peak reaches the front of the queue**, the same price-time unit any other order competes with. `CANCEL_RESTING`/`CANCEL_BOTH` then cancel the **whole** iceberg — peak and hidden reserve — as one order with one `order.cancelled` message; the hidden reserve is not replenished into a new peak.
- **SMP applies in an auction uncross and its indicative quantity.** Both orders are resting, so the bid is treated as the aggressor for policy selection; if its policy is `NONE`, the ask's policy applies instead. `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, and `CANCEL_BOTH` cancel the corresponding orders before they can trade with their own gateway. The same rule then runs once more over whatever is still crossed after the uncross prints: a bid above the equilibrium price and an ask below it that belong to one gateway never traded, and are cancelled by their SMP policy (no trade), so the book always opens

uncrossed. The architecture book explains the walk and works two auctions through step by step: *Order Book Deep Dive*, "Self-Match Prevention in an Auction Uncross".

16.11.5 `FOK` — self-match liquidity is excluded before the pre-check

`FOK` (Fill-Or-Kill) is the one order type that resolves SMP *before* the sweep starts, because it needs to know up front whether the book has enough **eligible** liquidity to fill the whole order — same-gateway resting quantity that SMP would skip or cancel does not count as eligible liquidity:

1. The engine walks the opposite side **in priority order**, the way the sweep will meet it, and sums the liquidity it can reach. With any `smp_action` other than `NONE`, resting orders of the same `gateway_id` do not count. Under `CANCEL_AGGRESSOR` / `CANCEL_BOTH` the sweep would stop at the first such order, so nothing behind it counts either. A resting iceberg's hidden quantity counts, but behind everything else at its price, where its replenished peaks queue.
2. If that filtered total is still less than the FOK's quantity, the order cannot fill completely. SMP is then resolved the same way a sweep would have resolved it — `CANCEL_RESTING`/`CANCEL_BOTH` cancel the conflicting resting orders; `CANCEL_AGGRESSOR`/`CANCEL_BOTH` cancel the FOK itself — and the FOK is finalised as `CANCELLED` (`SELF_MATCH_PREVENTED`) only for `CANCEL_AGGRESSOR`/`CANCEL_BOTH` with a same-gateway conflict; otherwise — including `CANCEL_RESTING`, and a genuine shortfall with no same-gateway liquidity involved — it is `REJECTED`.
3. Only if eligible liquidity is sufficient does the FOK proceed into the normal sweep, which then also enforces SMP level by level as above (a defensive safety net; the pre-check should already guarantee the sweep completes cleanly).

This keeps FOK's all-or-nothing guarantee intact: a self-match candidate can never cause a partial fill.

16.11.6 Cancellation is visible, not silent

When SMP cancels a resting order, that order transitions to `CANCELLED` like any other cancellation and the owning gateway receives a normal `order.cancelled.{GW_ID}` message — there is no separate SMP-specific wire message. On the binary BALF protocol, a system-originated cancel is delivered as a `CANCEL_ACK` with `cancel_reason = 255` (system); an SMP cancel cannot be distinguished there from a session-end or expiry cancel. ALF (text) clients receive `CANCEL_REASON=SELF_MATCH_PREVENTED` on the cancelled event, and REST WebSocket clients receive a `cancel_reason` field (see above).

16.11.7 Two ways to specify `smp_action`

SMP can be set in two places, and they serve different purposes:

1. **Per order or per combo leg** — the `SMP=` field on a `NEW`/`COMBO` command (ALF, BALF), the `smp_action` field on the REST `OrderRequest` / `ComboRequest` payload (API Gateway), or the `smp_action` key on a `market_maker_combos[].legs[]` seed entry in `engine_config.yaml`. This is the client expressing "for *this* order, do X on self-match."
2. **`participants[].smp_action`** — a per-gateway default in `engine_config.yaml`, applied by the engine when an order or leg does **not** specify `SMP=` at all. This is the operator expressing "for *this gateway*, when nobody says otherwise, do X."

```
participants:
  - id: TRADER01
    description: "Prop desk algo 1"
    smp_action: CANCEL_RESTING   # gateway-level default
```

Field	Location	Required	Values	Default
`SMP=`	`NEW`/`COMBO` command (ALF), `smp` byte (BALF), `smp_action` (REST)	No	`NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	*(unspecified — falls back, see below)*
`participants[].smp_action`	`engine_config.yaml`	No	`NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`	`NONE`

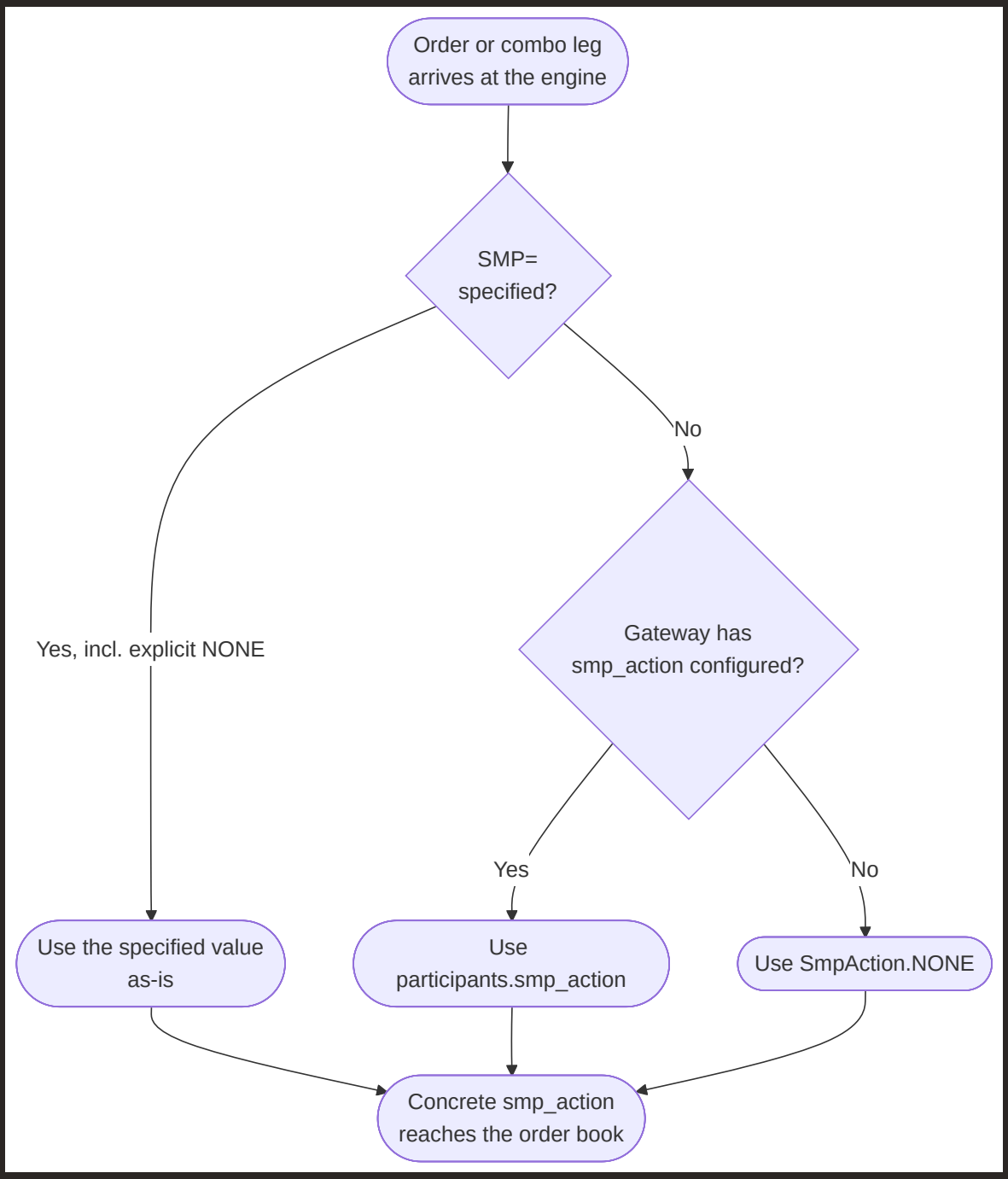
16.11.8 Precedence: why "omitted" and "explicit `NONE`" are not the same thing

This is the subtlety that makes SMP worth documenting carefully. `SMP=NONE` sent by a client is a **deliberate instruction**: "I know this might self-match, and I want the trade to happen anyway." An **omitted** `SMP=` means the client simply didn't think about it. Collapsing those two cases together would either force every client to think about SMP on every order (bad ergonomics) or silently let an operator's safety default be overridden by clients who never intended to override anything (a real safety gap). So the engine treats them as genuinely different values:

- Internally, `smp_action` on an `Order` or `ComboLeg` is typed as *optional* — `None` means "not specified," and is distinct from the concrete enum member `SmpAction.NONE`, which means "specified as off."
- Client-facing processes (`pm-alf-gwy`, `pm-alf-console`, the API Gateway) preserve that distinction all the way from the wire request into the internal order object — an omitted field parses to `None`, not to `SmpAction.NONE`.
- The engine resolves the final, concrete `smp_action` exactly once, at the point an order or combo leg is accepted (`Engine._handle_new_order`, `Engine._accept_combo`), using this precedence:

```
1. Order/leg specified SMP= explicitly (including explicit SMP=NONE)
      → use that value, always. The gateway default is never consulted.
2. Order/leg omitted SMP= entirely
      → use participants[<this gateway>].smp_action
3. Gateway has no smp_action configured (or is unknown to the engine)
      → use SmpAction.NONE
```

By the time an order reaches the order book's matching logic, `smp_action` is always a concrete value — the book itself never sees the "unspecified" state and needs no special-casing for it.



16.11.9 Which paths carry an explicit per-request `SMP=`, and which only have the gateway default

Not every order-entry path exposes its own `SMP=` field:

Path	Explicit per-request `SMP=`?	Effective SMP source
ALF `NEW` (single order)	Yes	Explicit value, else gateway default, else `NONE`
ALF `NEW|TYPE=COMBO`	Yes — but **one `SMP=` value for the whole combo**, applied identically to every leg (no `LEG<i>.SMP`)	Explicit value applied to all legs, else gateway default applied to all legs, else `NONE`
BALF `NEW_ORDER`	Yes (`smp` byte is mandatory in the fixed frame — see note below)	Always the value on the wire — see BALF Protocol Reference — `NEW_ORDER`

Path	Explicit per-request SMP=?	Effective SMP source
REST OrderRequest / ComboRequest	Yes (JSON field, optional; on a ComboRequest it is **one top-level smp_action for the whole combo**, applied to every leg)	Explicit value, else gateway default, else NONE
Market-maker QUOTE (both pm-mm-bot and the REST quoting endpoint)	**No** — quotes have no per-request SMP field	Always participants[].smp_action, else NONE
OCO (order.oco, every gateway)	**No** — the OCO has no SMP field	Always participants[].smp_action, else NONE, applied to both legs — including a stop leg that triggers later
market_maker_combos[].legs[] config-seeded combo	Optional smp_action key, settable **per leg**	Explicit per-leg value, else gateway default, else NONE — resolved independently per leg

BALF's smp byte is mandatory, not omittable

BALF is a fixed-width binary protocol — the smp byte at offset 51 of NEW_ORDER is always present on the wire (0x00–0x03; see BALF Protocol Reference — NEW_ORDER). There is no wire representation of "the client didn't send this field." A BALF client that wants the gateway default to apply must send 0x00 and rely on participants[].smp_action being configured to something other than NONE — sending 0x00 is indistinguishable from an explicit SMP=NONE and does **not** fall back to the gateway default. This is a deliberate scope decision: extending the None/omitted distinction into the binary frame would require a wire format change (e.g. a presence bitmap), which BALF does not currently have.

Market makers are the clearest illustration of why the gateway-level default exists at all: a QUOTE submits two legs (bid and ask) in one call with no room for a per-leg SMP=, yet a market maker's own stale bid and fresh ask can easily cross each other after a quote refresh. Configuring participants[<mm-gateway>].smp_action: CANCEL_RESTING is the *only* way to protect a quoting gateway from this — see Market-Maker Bot — Recommended settings for the concrete example.

16.11.10 Worked examples

Example 1 — omitted SMP=, gateway has a configured default.

```
participants:
  - id: TRADER01
    smp_action: CANCEL_RESTING
```

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00
```

No SMP= on the wire → resolves to CANCEL_RESTING (the gateway default). If TRADER01 already has a resting SELL 100 @ 150.00, that resting order is cancelled and the incoming buy continues matching against other participants (or rests itself, if none remain).

Example 2 — explicit SMP=NONE overrides a non-NONE gateway default.

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|SMP=NONE
```

Even though TRADER01's gateway default is CANCEL_RESTING, the explicit SMP=NONE wins: the order is allowed to trade against TRADER01's own resting sell, producing a genuine self-trade. This is the "I meant it" escape hatch — useful for, e.g., internal cross facilitation the firm has separately reconciled for.

Example 3 — quote leg, no gateway default configured.

```
participants:
  - id: MM_AAPL_02
    role: MARKET_MAKER
    # no smp_action set
```

A QUOTE from MM_AAPL_02 has no SMP= field to omit or specify — it always falls back to the gateway default, which here is unset, so it resolves to SmpAction.NONE. A stale leg from a prior quote can self-trade against the fresh replacement leg. This is the exact gap participants[].smp_action closes when configured — see Example 1's pattern applied to a market maker.

Example 4 — combo on the ALF text protocol: one SMP= value for every leg.

```
[TRADER01]> NEW|TYPE=COMBO|COMBO_ID=spread-1|COMBO_TYPE=AON|TIF=DAY|LEG_COUNT=2|
  LEG0.SYM=AAPL|LEG0.SIDE=BUY|LEG0.QTY=100|LEG0.PRICE=150.00|
  LEG1.SYM=MSFT|LEG1.SIDE=SELL|LEG1.QTY=50|LEG1.PRICE=400.00|SMP=CANCEL_BOTH
```

On the ALF text protocol there is a single combo-level `SMP=` field, not a per-leg `LEG<i>.SMP` — the parsed value (or, if `SMP=` is omitted entirely, the `None` sentinel) is applied identically to every leg's `ComboLeg`. Here `CANCEL_BOTH` applies to both `LEG0` and `LEG1`.

Example 5 — combo submitted via REST: combo-level `smp_action`.

The REST `ComboRequest` carries a single top-level `smp_action` (legs have no `smp_action` field; the schema rejects unknown keys), applied to every leg. Only the `market_maker_combos[].legs[]` config-seed path allows a per-leg value.

```
{
  "combo_id": "spread-1",
  "combo_type": "AON",
  "tif": "DAY",
  "smp_action": "NONE",
  "legs": [
    {"symbol": "AAPL", "side": "BUY",  "quantity": 100, "price": 150.00},
    {"symbol": "MSFT", "side": "SELL", "quantity": 50,  "price": 400.00}
  ]
}
```

`smp_action: "NONE"` applies to both legs, regardless of `TRADER01`'s gateway default. Omitting the field would make every leg fall back to `TRADER01`'s `participants[].smp_action` (else `NONE`).

16.11.11 SMP and the other risk controls

SMP is orthogonal to the four admission-path/exposure controls described above: it does not reject orders at admission time, does not halt a symbol, and is not something any gateway can trigger against another gateway. It only ever inspects and acts on orders belonging to the *same* `gateway_id` that are about to trade against each other during matching. An instrument halt, price collar, or circuit-breaker rejection all happen *before* SMP would ever be reached, since none of those orders get as far as the sweep.

Property	SMP	Kill switch
Trigger	Automatic, at match time, when both sides share a `gateway_id`	Explicit command from the gateway
Scope	One aggressor vs. one resting order at a time	All of a gateway's resting orders (optionally one symbol)
Configurable outcome	Yes — 4 actions	No — always cancels
Can a gateway opt out	Yes — `SMP=NONE` (explicit) or an unconfigured/`NONE` gateway default	No

16.11.12 Configuration reference

See Configuration — Participant Fields for the full `participants[].smp_action` field definition and Configuration Spec §5.2 for the normative schema entry, including how the same default extends to `market_maker_combos[].legs[]` config-seeded combos. Per-request field definitions live alongside each protocol's `NEW`/`COMBO`/`QUOTE` command reference: ALF Console — `NEW`, ALF Console — `NEW (Combo)`, Combos — `SMP=`, and API Gateway for the REST field.

16.12 Market-maker interaction

When any halt fires — whether triggered by a circuit breaker, a per-symbol operator halt, or the exchange-wide ADMIN halt — **all outstanding market-maker quote legs for the affected symbols are cancelled immediately**. This protects market makers from having stale quotes executed against them during a disorderly market. The cancellation reason included in the `order.cancelled` message distinguishes the halt source:

Halt source	Cancellation reason text
Circuit breaker	`"Circuit breaker halt"`
Per-symbol operator halt	`"Per-symbol halt"`
Exchange-wide ADMIN halt	`"Global circuit breaker halt"`

When a symbol resumes, market makers are expected to submit fresh quotes at updated prices. The engine will begin enforcing MM obligation checks again immediately upon resumption.

16.13 See also

In this book:

- The Configuration Workflow — full `engine_config.yaml` reference including collar, CB ladder, and `smp_action` config
- Auctions & Session Scheduling — the equilibrium-price uncross algorithm that timed circuit-breaker resumption always runs

Participant Guide:

- Order Types — how different order types behave under halt, and how SMP interacts with each type's sweep
- ALF Console — `KILL` command for triggering the kill switch via the ALF terminal, and the `NEW/COMBO SMP=` field
- Combos — per-leg `SMP=` on multi-leg orders
- Market-Maker Bot — why quoting gateways rely entirely on the `smp_action` gateway default

Reference Manual:

- Configuration Spec — normative schema for `participants[].smp_action` and `ComboLegSpec.smp_action`

Protocols and Clients:

- Drop Copy — how fill events are forwarded to risk systems

CHAPTER 17

Statistics and Reporting

Learning objectives

After reading this page you will understand:

- How EduMatcher defines a **trading date**, and why it is not the UTC date
- How to record market and exchange index statistics continuously using `pm-stats`
- How to query statistics data without writing SQL using `pm-stats-cli`
  - Common analyst workflows: end-of-day summaries, intraday price analysis, trade analysis, index level history, and order lifecycle investigation
- How to export statistics for external analysis (spreadsheets, BI tools)
- How the statistics system integrates with other tools like `pm-ticker`
- How to troubleshoot and validate statistics data

17.1 Overview — Statistics Architecture

EduMatcher has a two-part statistics system:

Component	Role	Type	Purpose
pm-stats	Subscriber	Long-running process	Listens to trades, book updates, index level updates, and private order lifecycle events; writes OHLCV, snapshots, trade log, index history, and `order_events` to `data/stats.db`
pm-stats-cli	Query tool	One-shot CLI	Reads from `data/stats.db` and prints human-friendly or machine-readable output without SQL

This split keeps the recorder separate from the query interface, so you can:

- Start and stop `pm-stats-cli` at any time without affecting the live recorder
- Reload historical data after the engine restarts
- Build reports, dashboards, and automated analysis without needing live connections
- Keep the database read-only for auditing and compliance

17.2 Dates and Timestamps — read this first

Every date question in `stats.db` has two possible answers, and mixing them up is the easiest way to produce a report that is quietly wrong. The database uses two distinct concepts:

Concept	Where it appears	What it is
Instant	every `ts` column	An ISO-8601 UTC timestamp with an explicit `+00:00` offset, e.g. `2026-06-14T09:00:01.000+00:00`
Trading date	the `date` column of `daily_stats` and `index_daily_stats`	The calendar date **in the exchange's session timezone** that an instant belongs to, e.g. `2026-06-14`

The trading date is deliberately *not* the UTC date. An exchange's daily rollup has to align to the calendar day its participants actually traded. A session that runs into the evening — or any session that straddles 00:00 UTC — would otherwise be split across two `date` values, and the daily summary would no longer describe a single trading session.

17.2.1 Setting the session timezone

Set it once, in the deployed config:

```
locale:
  session_timezone: Europe/Stockholm
```

`pm-stats` reads `locale.session_timezone` from the compiled config and writes that value into the database's `stats_meta` table, and every reader — `pm-stats-cli`, `pm-ticker`, `pm-viewer`, the API Gateway — picks it up from there. **Readers need no timezone configuration at all**, and cannot disagree with the recorder about which trading day a `--date` refers to:

```
pm-stats-cli daily --date 2026-06-14   # resolves in Europe/Stockholm automatically
```

Two guardrails back this up:

- Restarting `pm-stats` against an existing database with a *different* `locale.session_timezone` is **refused**, because the `date` column would then mean two different things within one file. Use a different `--db` instead.
- Passing `--timezone` to a reader overrides the recorded value and prints a warning when the two disagree:

```
$ pm-stats-cli --timezone UTC daily --date 2026-06-14
[WARN] --timezone UTC differs from the session timezone this database was
       recorded with (Europe/Stockholm); --date will resolve to a different
       trading day than pm-stats used
```

pm-clearing reads the same setting

`pm-clearing` keeps a separate database but reads the same `locale.session_timezone` from the same compiled config, for `trade_events.trade_date`. Since both recorders read one config, they cannot disagree with each other unless the config changes between restarts — if `daily_stats.volume` ever fails to reconcile against `gateway_daily_summary.traded_qty`, redeploy a consistent config and restart both.
If your exchange runs in UTC, set `session_timezone: UTC` and none of this applies.

17.2.2 Inspecting what a database was recorded with

`stats_meta` is a plain key/value table:

```
$ sqlite3 data/stats.db "SELECT key, value FROM stats_meta"
recorder|pm-stats
session_timezone|Europe/Stockholm
snapshot_interval_sec|900.0
```

The schema version is in `PRAGMA user_version`:

```
$ sqlite3 data/stats.db "PRAGMA user_version"
11
```

The version is bumped both when the table definitions change and when the *meaning* of stored values changes — version 2 introduced the distinct combo, OCO and quote `event_type` values, which the DDL alone cannot express, and version 3 added `feed_gaps` and widened `trade_log`'s primary key, and version 4 moved prices from REAL display money

to INTEGER ticks (except `order_events`, see below), version 5 added the `instruments` reference table, version 6 renamed `order_events.client_order_id` to `client_tag`, version 7 made the durable `trade_id` the sole primary key of `trade_log`, version 8 replaced `order_events.trade_id` (always empty) with `trade_ids`, and version 9 added `order_events.reject_code`, the rejection's machine-readable code (until then only the reason text was kept), version 10 added the order ids, `match_type` and `msg_id` to `trade_log` and `msg_id` to `order_events`, and version 11 keyed `index_level_snapshots` on an arrival `seq`, so that two updates in one millisecond are both kept.

`pm-stats` refuses to open a database whose `user_version` does not match the build, rather than writing new-format rows into an old-format file. If you hit that, move or delete the file and let `pm-stats` create a fresh one.

17.2.3 What `--date` means

`--date 2026-06-14` always means "the 2026-06-14 trading day". For `daily_stats` and `index_daily_stats` that is a direct match on the `date` column. For the timestamped tables (`trade_log`, `price_snapshots`, `order_events`, `index_level_snapshots`) it is resolved to the UTC instant range that trading day covers:

```
--timezone Europe/Stockholm --date 2026-06-14
    →  ts >= 2026-06-13T22:00:00+00:00
   and  ts <  2026-06-14T22:00:00+00:00
```

Daylight-saving transitions are handled: a trading day may be 23 or 25 hours long, and the range follows local midnight either side.

17.2.4 What `--from` / `--to` accept

`--from` and `--to` are **instants**, and all of these are accepted:

Form	Example	Interpreted as
UTC with `Z`	`2026-06-14T09:00:00Z`	09:00 UTC
Explicit offset	`2026-06-14T11:00:00+02:00`	09:00 UTC
No offset	`2026-06-14T11:00:00`	11:00 **session-local**, so 09:00 UTC in Stockholm

The three rows above are all converted to the same UTC bound. Both bounds are inclusive, but the comparison is against the stored timestamp *text*, and a bound written without fractional seconds (`16:30:00+00:00`) sorts *before* every stored `16:30:00.mmm+00:00` value. So `--to 2026-06-14T16:30:00` excludes the whole `16:30:00` second, including a trade at exactly `16:30:00.000`. Give the bound sub-second precision (`--to 2026-06-14T16:30:00.999`) if you want that second included.

17.3 Data Folder Location

The location where `pm-stats` writes `data/stats.db` depends on how EduMatcher is installed:

Running mode	Default location	Environment override
Source checkout (`poetry run pm-stats`)	`<repo>/src/data/stats.db`	`EDUMATCHER_DATA_DIR`
Installed (`pm-stats` on PATH)	`~/.local/share/edumatcher/stats.db`	`EDUMATCHER_DATA_DIR`

Set the data directory in your shell profile (`~/.zshrc` or `~/.bashrc`) to override either default:

```
export EDUMATCHER_DATA_DIR="$HOME/.local/share/edumatcher"
```

Then every `pm-*` command — including `pm-stats` and `pm-stats-cli` — will use that location automatically:

```
# Uses $EDUMATCHER_DATA_DIR/stats.db
pm-stats
pm-stats-cli daily
```

Common use cases:

Scenario	Environment variable	Purpose
Installed user (default for pipx)	(unset) → `~/.local/share/edumatcher`	Persistent user data folder
Source checkout (default for `poetry run`)	(unset) → `<repo>/src/data/`	Development environment
Isolated sessions	`~/sessions/morning`	Per-session isolation for demos or testing
Shared network	`/mnt/shared/trading/`	Shared data across machines

Example: Per-session isolation

```
# Session 1: Morning trading (uses custom data directory)
export EDUMATCHER_DATA_DIR="$HOME/sessions/morning"
poetry run pm-engine
poetry run pm-stats
poetry run pm-stats-cli daily

# Session 2: Afternoon trading (different database)
export EDUMATCHER_DATA_DIR="$HOME/sessions/afternoon"
poetry run pm-engine
poetry run pm-stats
```

Each session maintains its own `stats.db`, so historical data doesn't mix.

Finding your data

To see where `pm-stats` is writing data: `bash echo $EDUMATCHER_DATA_DIR # Shows override if set, otherwise empty poetry run python -c "from edumatcher.config import DATA_DIR; print(DATA_DIR)" # Shows resolved path ls -la $EDUMATCHER_DATA_DIR/stats.db # If env var is set`
See Processes — Environment variables for full details on `EDUMATCHER_DATA_DIR`.

17.4 Prices Are Stored As Integer Ticks

The single most important thing to know about this database: the price columns (with one exception, `order_events`, below) do not contain money. They contain an integer count of ticks. If you open `stats.db` in a SQL browser and see `15025`, that is `150.25`.

```
$ sqlite3 data/stats.db "SELECT symbol, price, quantity, tick_decimals FROM trade_log LIMIT 1"
AAPL|15025|100|2
```

```
$ pm-stats-cli trades --symbol AAPL --limit 1
ts                           | trade_id | symbol | price  | quantity | ...
-----------------------------+----------+--------+--------+----------+----
2026-06-14T09:00:01.000+00:00 | 000042-000000001 | AAPL | 150.25 | 100 | ...
```

Same row. `pm-stats-cli` and the API convert for you; raw SQL does not.

17.4.1 The conversion rule

Every table holding prices carries a `tick_decimals` column — the number of decimal places that symbol prices to. Divide by ten to that power:

```
display price = price / 10^tick_decimals
150.25        = 15025 / 10^2
1.2345        = 12345 / 10^4
```

Always use the **row's own** `tick_decimals`. Symbols do not share a tick size: an FX pair at four decimals and an equity at two sit in the same table, and assuming `2` for both silently mangles the FX prices.

```
-- Right
SELECT symbol, price / power(10, tick_decimals) AS display_price FROM trade_log;

-- Wrong: breaks on any symbol that is not 2-decimal
SELECT symbol, price / 100.0 FROM trade_log;
```

17.4.2 Why integers

The engine matches in integer ticks and converts to a float only to publish. `pm-stats` converts straight back at ingest, so the number stored is the exact number the engine matched on. Nothing rounds, and a day's turnover is an integer sum that cannot drift however many fills it spans — where a float sum would differ in its last digits depending on the order the trades were added.

This also lets `daily_stats` reconcile *exactly* against `pm-clearing`, which stores its own archive the same way (see P&L and Clearing).

17.4.3 The exceptions

Column	Type	Why
`daily_stats.vwap`	REAL	`turnover / volume` is a ratio and rarely lands on a tick. Both inputs are stored exactly beside it, so compute it yourself when you need full precision.
`price_snapshots.mid_price`	REAL	The midpoint of two adjacent ticks is a *half* tick, which no integer can hold. `best_bid` and `best_ask` beside it are exact.
index `level`, `aggregate_cap`, `divisor`	REAL	An index level is a computed, dimensionless number, not a price on a tick grid — there is no tick size to express it in.
`order_events.price`, `order_events.fill_price`	REAL	Stored exactly as the engine published them on the private order events: **display money**, with no `tick_decimals` column. Do not divide them.

`vwap` and `mid_price` are still **in ticks**, not in money, so they convert with the same rule. Index levels and the `order_events` prices are neither — they are already the number you want.

17.4.4 `turnover` — stored in ticks, converted like a price

`turnover` is `sum(price_ticks × quantity)`. Dividing by the tick scale gives `sum(price_money × quantity)` — the money notional, exactly:

```
notional in money = turnover / 10^tick_decimals
```

So `turnover` converts on output like every other tick column, and a row is self-consistent in whichever unit you are reading:

	`turnover`	`volume`	`vwap`	`turnover / volume`
Raw SQL (ticks)	`460000`	`40`	`11500.0`	`11500.0` = `vwap`
`pm-stats-cli` (money)	`4600.0`	`40`	`115.0`	`115.0` = `vwap`

17.4.5 What this means for exported data

A CSV export carries the converted display prices, since the conversion happens before output. But `--wide` also includes `tick_decimals`, so the exact integers remain recoverable:

```
price_ticks = round(display_price × 10^tick_decimals)
```

Keep that column if you intend to reconcile against `stats.db` or `clearing.db` later.

17.4.6 Residual float exposure

After all of the above, the only values where floating point can still bite are the three REAL exceptions. `aggregate_cap` is the most exposed: index market caps run to 10^12 or beyond, where a double's absolute resolution is around 10^-4. Treat its last few digits as noise, and compare with a tolerance rather than for equality.

17.5 The Statistics Database Schema

All statistics are stored in `data/stats.db`, a SQLite 3 database with seven data tables, an instrument reference table, and a metadata table.

17.5.1 Row identity and pagination cursors

Keyset pagination (`--after`, and `next_cursor` on the API) needs a total ordering, so each cursor carries the last row's sort key plus a tiebreaker. Which tiebreaker depends on the table:

Table	Cursor tiebreaker	Stable across `VACUUM`?
`order_events`	`seq` — a real `INTEGER PRIMARY KEY AUTOINCREMENT` column	Yes
`daily_stats`	`symbol` (with `date`) — real columns	Yes
`index_daily_stats`	`index_id` (with `date`) — real columns	Yes
`trade_log`	SQLite's implicit `rowid`	**Not guaranteed**
`price_snapshots`	SQLite's implicit `rowid`	**Not guaranteed**
`index_level_snapshots`	SQLite's implicit `rowid`	**Not guaranteed**

The last three have composite primary keys rather than an explicit `INTEGER PRIMARY KEY`, so SQLite is free to renumber their `rowid`s during a `VACUUM`. In practice it usually does not — SQLite 3.37 leaves them untouched in the straightforward case — but the documentation only says it *may*, so nothing should depend on it.

Do not hold a cursor across a VACUUM

A `VACUUM` between fetching one page and requesting the next can, on those three tables, make the cursor's `rowid` refer to a different row. The damage is bounded — `ts` is the primary sort and `rowid` only breaks ties — so at worst you skip or repeat rows *sharing a single timestamp*. It will not error.

This matters for the retention procedure below, which ends in a `VACUUM`. Finish paginating first, or re-run the query from the start afterwards.

17.5.2 `instruments`

Reference data for every instrument seen, so the database describes itself rather than requiring outside knowledge to interpret.

Primary key: `symbol`.

Column	Type	Description
`symbol`	TEXT	Instrument ticker

Column	Type	Description
`tick_decimals`	INTEGER	Decimal places a price trades in; 1 tick = `10^-tick_decimals`
`tick_size`	REAL	The tick as a number, e.g. `0.01`. Derivable from `tick_decimals`, stored so a BI tool can join without arithmetic
`currency`	TEXT	**Reserved for future use — always NULL.** See below
`source`	TEXT	`config` (authoritative, from engine `symbol_meta`) or `observed` (inferred from a message)
`updated_ts`	TEXT	When the row was last written

Two sources populate it, and **config wins**:

- **`config`** — the engine's `symbol_meta.tick_size`, received in the startup `system.symbols` response. Authoritative, and covers symbols that never trade and would otherwise never appear.
- **`observed`** — the `tick_decimals` on a trade or book message, for a symbol that has no config entry. Best effort: a message only reveals the scale *that message* was produced at, so it never overwrites a `config` row.

```
$ pm-stats-cli instruments
symbol | tick_decimals | tick_size | currency | source   | updated_ts
-------+---------------+-----------+----------+----------+-----------------------------
AAPL   | 2             | 0.01      |          | config   | 2026-06-14T08:59:12.004+00:00
FXPAIR | 4             | 0.0001    |          | config   | 2026-06-14T08:59:12.004+00:00
NEWCO  | 3             | 0.001     |          | observed | 2026-06-14T09:11:40.882+00:00
```

currency is reserved, not merely empty

EduMatcher has **no currency model at all**. Prices carry no currency, no conversion happens anywhere, and `index_daily_stats.aggregate_cap` sums across whatever the constituents happen to be priced in. The column is present and always `NULL` so that a consumer can see the field is *absent* rather than assume a currency — and so introducing a currency model later does not require migrating the reference table.
Do not write to it, and do not infer a currency from its emptiness.

17.5.3 `stats_meta`

Key/value provenance for the file, written by `pm-stats` on every start.

Key	Description
`session_timezone`	The IANA timezone the `date` columns are expressed in. Readers resolve this automatically
`snapshot_interval_sec`	The `--snapshot-interval` in force, so a consumer can tell a sparse series from a coarse one
`recorder`	The process that wrote the file (`pm-stats`)

The schema version lives in `PRAGMA user_version` rather than in this table, so it can be checked before any query is attempted.

17.5.4 `daily_stats`

Aggregated OHLCV (open, high, low, close, volume) and related metrics for each symbol per trading day.

Primary key: `(date, symbol)` — one row per symbol per trading day, upserted as trades arrive.

Column	Type	Null?	Description
`date`	TEXT	no	**Trading date** `YYYY-MM-DD` in the session timezone
`symbol`	TEXT	no	Instrument ticker
`open_price`	INTEGER	yes	First trade price of the day, **in ticks**; null if no trades
`high_price`	INTEGER	yes	Highest trade price, **in ticks**
`low_price`	INTEGER	yes	Lowest trade price, **in ticks**
`close_price`	INTEGER	yes	Last trade price, **in ticks**
`volume`	INTEGER	no	Total traded quantity; `0` if the day had no trades
`trade_count`	INTEGER	no	Number of trades; `0` if the day had no trades
`turnover`	INTEGER	no	`sum(price_ticks × quantity)` — exact. Divide by `10^tick_decimals` for money notional
`vwap`	REAL	yes	Volume-weighted average price **in ticks**. Derived (`turnover / volume`), so REAL
`open_bid`	INTEGER	yes	Best bid at first book update of the day, **in ticks**
`open_ask`	INTEGER	yes	Best ask at first book update of the day, **in ticks**
`close_bid`	INTEGER	yes	Best bid at engine shutdown, **in ticks**
`close_ask`	INTEGER	yes	Best ask at engine shutdown, **in ticks**
`largest_trade_qty`	INTEGER	yes	Quantity of the single largest trade. Note this is `0`, not null, on a day with no trades
`largest_trade_price`	INTEGER	yes	Price of the single largest trade, **in ticks**
`tick_decimals`	INTEGER	no	Decimal scale for this row's tick columns

Use case: End-of-day summaries, daily trend analysis, multi-day performance tracking.

Prices are unadjusted

`daily_stats` carries no corporate-action awareness. Prices are exactly what printed on the day, with no split or dividend adjustment and no marker indicating that an adjustment event occurred. A multi-day series that spans a corporate action is therefore discontinuous, and returns computed across it will be wrong. `pm-index` maintains the corporate-action audit record — see Market Index.

17.5.5 `price_snapshots`

Intraday mid-price, bid/ask, and percentage-change history, recorded at most once per interval per symbol (default: 15 minutes).

Primary key: `(ts, symbol)`. **Index**: `(symbol, ts)`.

Column	Type	Null?	Description
`ts`	TEXT	no	ISO-8601 UTC instant, **second** precision, e.g. `2026-06-14T09:00:00+00:00`
`symbol`	TEXT	no	Instrument ticker
`mid_price`	REAL	yes	Midpoint **in ticks**; REAL because a midpoint can be a half tick. See the fallback chain below — **not always a true mid**
`best_bid`	INTEGER	yes	Best bid at snapshot time **in ticks**; null if the bid side was empty
`best_ask`	INTEGER	yes	Best ask at snapshot time **in ticks**; null if the ask side was empty
`pct_change`	REAL	yes	Percentage change of `mid_price` from the previous persisted row (`1.25` means +1.25 %); a percentage, never converted
`tick_decimals`	INTEGER	no	Decimal scale for this row's tick columns

`mid_price` is resolved in this order, and only the first case is a genuine mid-price:

Book state	`mid_price`
Both sides present	`(best_bid + best_ask) / 2`, in ticks — may be a half tick
Bid only	`best_bid`
Ask only	`best_ask`
Neither, but a last trade exists	`last_price`
Nothing available	null

mid_price on a one-sided book

On a one-sided book `mid_price` is simply whichever side exists, so a series can silently mix true mid-prices with single-sided quotes. Because `pct_change` is computed from consecutive `mid_price` values, a book flipping between two-sided and one-sided produces a percentage move that reflects the change in *definition*, not a change in the market. Check `best_bid`/`best_ask` for null before treating `pct_change` as a return.
Related: `pct_change` always compares against the immediately preceding persisted row. If that row's `mid_price` was null, it becomes the baseline and the next row's `pct_change` is null too — a percentage never silently spans more than one interval.

17.5.6 `trade_log`

Append-only record of every matched trade — no aggregation, one row per trade.

Primary key: `trade_id`. Inserts are `OR IGNORE`, so a repeated delivery of the same durable trade is deduplicated regardless of its delivery timestamp.

Durable trade identity

`trade_id` is durable across engine restarts, formatted as `run_seq-counter` (for example, `000042-000000001`), so it is sufficient as the sole key. Older statistics database schemas are rejected and must be recreated; historical short IDs are not migrated.

Column	Type	Null?	Description
`ts`	TEXT	no	ISO-8601 UTC instant, **millisecond** precision. This is the **engine's** trade timestamp
`trade_id`	TEXT (PK)	no	Durable engine trade id, formatted as `run_seq-counter`
`symbol`	TEXT	no	Instrument ticker
`price`	INTEGER	no	Execution price, **in ticks** — divide by `10^tick_decimals`
`quantity`	INTEGER	no	Matched quantity
`tick_decimals`	INTEGER	no	Decimal scale for `price`
`buy_gateway_id`	TEXT	yes	Gateway that submitted the buy order
`sell_gateway_id`	TEXT	yes	Gateway that submitted the sell order
`aggressor_side`	TEXT	yes	`BUY` or `SELL` for a continuous match; `AUCTION` for an uncross print

Index: `(symbol, ts)`.
`aggressor_side` is mirrored from the engine payload verbatim:

Value	Meaning
`BUY`	An incoming buy order swept resting sell liquidity
`SELL`	An incoming sell order swept resting buy liquidity
`AUCTION`	An opening or closing uncross print — both sides were resting, so there is no true aggressor

This is what makes trade classification and order-flow imbalance possible, and it is also the only way to separate auction prints from continuous ones:

```
# Continuous-session buy-side pressure for one trading day
pm-stats-cli --format json trades --symbol AAPL --date 2026-06-14 --limit 100000 \
  | python3 -c "
import json, sys, collections
rows = json.load(sys.stdin)
by_side = collections.Counter()
for r in rows:
    by_side[r['aggressor_side']] += r['quantity']
print(dict(by_side))
"
```

Use case: Trade-by-trade analysis, order-flow imbalance, separating auction from continuous volume, audit trails.

17.5.7 `feed_gaps`

Messages (including trades) the recorder can prove it never received.

Primary key: `seq` (`AUTOINCREMENT`). **Index**: `(ts)`.

Column	Type	Description
`seq`	INTEGER	Monotonic local sequence
`ts`	TEXT	UTC instant the gap was detected: the trade's timestamp for a trade-id gap, the receipt time for a publisher-sequence gap
`stream`	TEXT	Which topic the gap was detected on — `trade.executed` or any other subscribed topic (e.g. `book.AAPL`)
`expected_id`	INTEGER	The counter value expected next (trade-id suffix, or publisher sequence number)
`received_id`	INTEGER	The counter value that actually arrived
`missing_count`	INTEGER	How many trades are unaccounted for between them

ZeroMQ PUB/SUB drops messages silently once a subscriber falls behind its high-water mark, so without this table a session that lost trades is indistinguishable from a quiet one. Because the engine numbers trades with a monotonic counter, a jump in that counter is direct evidence of loss, and each jump is written here inside the same transaction as the trade that revealed it.

Use case: answering "is this session's data complete?" before trusting a volume, VWAP or turnover figure.

Detection works two ways, and both write here:

Source	Mechanism	Covers
Publisher sequence	Every published message carries a per-topic monotonic counter in a third ZeroMQ frame; a jump means messages were dropped	**Every subscribed stream** — `trade.executed`, `book.*`, `order.*`, `combo.*`, `oco.*`, `quote.*`, `index.update`
Engine trade id	The suffix of the durable trade id is monotonic within a run, so a jump is independent evidence	`trade.executed` only

The two overlap on trades deliberately: the trade-id check still catches loss that happened upstream of the publisher, which a publisher-side counter cannot see by construction.

What an empty feed_gaps does and does not prove

A sequence counter restarts when its publisher restarts. For trade ids, the run-sequence prefix changes on engine restart, so the recorder treats the new prefix as a restart rather than a gap — messages genuinely lost across a publisher restart are not detectable.

A topic published by a process that does not stamp sequences is reported once at `WARNING` ("carries no sequence frame; loss on it cannot be detected") and then ignored, so a silent stream cannot masquerade as a clean one.

17.5.8 `order_events`

Append-only order lifecycle history captured from private engine topics. This table is used by API Gateway history endpoints to reconstruct per-gateway order, fill, cancel, amend, combo, OCO, and quote events.

Primary key: `seq` (`AUTOINCREMENT`). **Indexes**: `(order_id)`, `(gateway_id, ts)`, `(symbol, ts)`, `(event_type, ts)`.

order_events.ts is a different clock from trade_log.ts

`order_events.ts` is the wall-clock instant at which **`pm-stats` recorded** the event. `trade_log.ts` is the instant the **engine** stamped on the trade. The two therefore cannot be merged into one ordered timeline: a `FILL` row can carry a timestamp *earlier or later* than the `trade_log` row for the same execution, depending on delivery latency. Within `order_events` alone, order by `seq`, which is monotonic and reliable.

Column	Type	Description
`seq`	INTEGER	Monotonic local sequence assigned by SQLite for stable event ordering
`ts`	TEXT	ISO-8601 timestamp (UTC, millisecond precision) when `pm-stats` recorded the event
`event_type`	TEXT	Normalized event category — see the full value table below
`order_id`	TEXT	Order-like identifier; for combo/OCO/quote events this may be `combo_id`, `oco_id`, or `quote_id`
`gateway_id`	TEXT	Gateway identity that owns the private event
`symbol`	TEXT	Instrument ticker when present in the event payload; `NOT NULL`, so an empty string when the payload carries none
`side`	TEXT	`BUY` or `SELL` when applicable
`order_type`	TEXT	Order type from the original order or lifecycle event
`tif`	TEXT	Time-in-force value when present
`price`	REAL	Limit/order price when present, in **display money** (not ticks; there is no `tick_decimals` on this table)
`quantity`	INTEGER	Original or submitted quantity when present
`remaining_qty`	INTEGER	Quantity remaining after the event when provided by the engine
`status`	TEXT	Engine status value when present
`fill_price`	REAL	Execution price for fill events, in **display money** (not ticks)
`fill_qty`	INTEGER	Executed quantity for fill events

`trade_ids`	TEXT	FILL only: JSON array of the trade ids the fill executed — one fill can aggregate several trades
`reason`	TEXT	Rejection, cancel, expire, or status reason when provided
`client_tag`	TEXT	Client-supplied order correlation tag when present
`combo_parent_id`	TEXT	Parent combo identifier for combo child events
`oco_group_id`	TEXT	OCO group identifier for linked order events
`priority_reset`	INTEGER	`1` when an amend reset queue priority, `0` when it did not, null when not applicable

Use case: API Gateway order history, support investigations, per-gateway audit trails, fill-only history, and lifecycle reconstruction for a single order ID.

`event_type` values

Every value the recorder can write, and the engine topic it comes from:

`event_type`	Source topic	Meaning
`ACK`	`order.ack.*` with `accepted: true`	Order accepted
`REJECT`	`order.ack.*` with `accepted: false`	Order rejected
`FILL`	`order.fill.*`	Execution against the order
`AMEND`	`order.amended.*`	Order amended
`CANCEL`	`order.cancelled.*`, and `oco.cancelled.*` for the leg it names	Order cancelled; an OCO sibling's row carries the pair in `oco_group_id`
`EXPIRE`	`order.expired.*`	Order expired
`COMBO_ACK`	`combo.ack.*` with `accepted: true`	Combo accepted
`COMBO_REJECT`	`combo.ack.*` with `accepted: false`	Combo rejected
`COMBO_STATUS`	`combo.status.*`	Combo status update
`OCO_ACK`	`oco.ack.*` with `accepted: true`	OCO pair accepted
`OCO_REJECT`	`oco.ack.*` with `accepted: false`	OCO pair rejected
`OCO_CANCEL`	`oco.cancelled.*`	One leg of an OCO pair cancelled (row keyed on the OCO id; the leg also gets a `CANCEL`)
`QUOTE_ACK`	`quote.ack.*` with `accepted: true`	Quote accepted
`QUOTE_REJECT`	`quote.ack.*` with `accepted: false`	Quote rejected
`QUOTE_STATUS`	`quote.status.*`	Quote status update
`UNKNOWN`	any ack topic missing its `accepted` flag	See the note below
`EVENT`	any other subscribed private topic	Unclassified

The outcome of an OCO, combo or quote *cancel* (`oco.cancel_ack`, `combo.cancel_ack`, `quote.cancel_ack`) is not a row of its own: the legs it cancelled appear as `CANCEL` rows, and a combo's as `COMBO_STATUS CANCELLED`. The `*_ACK` and `*_REJECT` rows are submissions only.

Every `*_REJECT` row keeps the engine's `reject_code` (for example `INSUFFICIENT_LIQUIDITY`, `ORDER_NOT_FOUND`) next to the free-text `reason`.

Combo, OCO and quote events each carry their own accept / reject / cancel / status value rather than a single family name, so a rejected combo is distinguishable from an accepted one and `oco.cancelled` is findable as a cancellation. Filters are validated

against this list — `pm-stats-cli` rejects an unknown `--event-type` at parse time and the API returns `400`, rather than silently returning an empty page.

UNKNOWN means the engine did not say

Every ack-style payload carries an `accepted` flag, so its absence indicates a bug or a corrupted message. When that happens the recorder writes `UNKNOWN` and logs at `ERROR`, rather than defaulting to `REJECT` — recording a rejection the engine never asserted would put a fabricated fact into the audit trail. If you see `UNKNOWN` rows, check the `pm-stats` log for the topic that produced them.

17.5.9 `index_daily_stats`

Aggregated daily OHLC (open, high, low, close) for each configured exchange index, one row per `(date, index_id)`, upserted on every `index.update` event `pm-stats` receives from `pm-index`.

Primary key: `(date, index_id)`. **Index**: `(index_id, date)`.

Column	Type	Description
`date`	TEXT	**Trading date** `YYYY-MM-DD` in the session timezone
`index_id`	TEXT	Index identifier (e.g. `EDU100`)
`open_level`	REAL	Index level at the first update of the day
`high_level`	REAL	Highest index level seen during the day
`low_level`	REAL	Lowest index level seen during the day
`close_level`	REAL	Index level at the *most recently received* update — see the finality note below
`close_session_state`	TEXT	Session state as of that most recent update (e.g. `CONTINUOUS`, `CLOSED`) — the key to knowing whether `close_level` is final
`open_aggregate_cap`	REAL	Aggregate constituent market cap at the first update, in **display money**
`close_aggregate_cap`	REAL	Aggregate constituent market cap at the most recent update, in **display money**
`update_count`	INTEGER	Number of `index.update` events folded into this day's row

Use case: Daily index trend analysis, comparing index performance across trading dates, spotting days with unusually few updates (a thin `update_count` may indicate a quiet index or a connectivity gap), and — the most common ask — looking up an index's official end-of-day (EOD) closing level for a chosen date.

Note: an index has no independent trades or volume of its own — its level is computed from constituent prices — so this table has no `volume`/`trade_count`/`vwap` columns the way `daily_stats` does.

close_level is only final once close_session_state is CLOSED

`close_level` (and `close_session_state`) are updated on *every* `index.update` tick — they always reflect whatever was most recently received for that date, not necessarily the actual end-of-day print. For any **past** date this is a non-issue: no more updates can arrive for a date that has rolled over, so `close_level` is guaranteed final. But if you query **today's** date while the session is still open, `close_level` is a live "last level so far" that will keep changing intraday, and `close_session_state` will show whatever state the market is currently in (e.g. `CONTINUOUS`), not `CLOSED`.

To reliably get the true EOD close for a given date:

- **Simplest**: query a date that has already ended — `close_level` for a prior date is always final.
- **To confirm today's row is final**: check that `close_session_state == "CLOSED"`. `pm-index` sets this via a forced publish when the session transitions to `CLOSED`, so once you see it, `close_level` for that date will not change again.

See Getting the EOD index level for a date below for a worked example.

17.5.10 `index_level_snapshots`

Time series of every index level update received from `pm-index`, one row per `index.update` event (no additional throttling in `pm-stats` — `pm-index` already rate-limits its own publications via `publish_interval_sec` before it ever sends one).

Primary key: `seq`, the arrival order. Every update is its own row, including several for one index in the same millisecond.
Index: `(index_id, ts)`.

Column	Type	Description
`seq`	INTEGER	Arrival order (primary key)
`ts`	TEXT	ISO-8601 UTC instant, millisecond precision
`index_id`	TEXT	Index identifier
`level`	REAL	Current index level at this update. Dimensionless — an index level is a computed number, not a price, so it is neither in ticks nor in money
`aggregate_cap`	REAL	Aggregate constituent market cap at this update, in **display money** — see the currency note below
`divisor`	REAL	Index divisor in effect at this update. Dimensionless: `level = aggregate_cap / divisor`
`session_state`	TEXT	Index session state at this update — see the full set below
`day_open`	REAL	Day's opening level, when known at this update
`day_high`	REAL	Day's high level so far, when known at this update
`day_low`	REAL	Day's low level so far, when known at this update

aggregate_cap is money, but no currency is recorded

Market cap is the one monetary value in `stats.db` that is **not** in ticks — an index spans several instruments, so there is no single tick grid to express it on, and it is stored as display money in whatever currency the constituents are priced in. That currency is not recorded anywhere in the database. If your exchange lists in more than one currency, `aggregate_cap` sums across them and is not meaningful without knowing the mix.
It is also the value most exposed to floating-point error: caps run to 10^12 or beyond, where a double resolves to roughly 10^-4. Treat its last few digits as noise and compare with a tolerance.
`session_state` (in both this table and `index_daily_stats.close_session_state`) takes exactly one of the five values from the engine's session model:

Value	Meaning
`PRE_OPEN`	Orders accepted, no matching
`OPENING_AUCTION`	Auction collection before the opening uncross
`CONTINUOUS`	Normal continuous matching
`CLOSING_AUCTION`	Auction collection before the closing uncross
`CLOSED`	Market closed — **the only value that makes `close_level` final**

See Session Scheduling and Auctions for the full phase model.
Use case: Intraday index charting, index-level history queries for `pm-terminal`-style viewers, reconstructing an index's level trajectory over any time window.
Why this table exists: `pm-index` also keeps its own append-only JSONL history file (`data/indexes/<id>_history.jsonl`) for corporate-action, delisting, and constituent-change audit records — that file remains the source of truth for those event types and is unaffected by this table. But that file is not indexed and every query against it is a full linear scan, which does not scale as a session runs longer. `index_level_snapshots` exists specifically to give the level time series (the data an index chart needs) a queryable, indexed home, the same way `price_snapshots` already does for instrument prices — it does not replace or duplicate the JSONL file's audit role.

17.6 Running the Statistics Recorder

Start `pm-stats` as a background process after the engine starts:

```
# Terminal 1: Start the engine
pm-engine --verbose

# Terminal 2: Start statistics recorder (after engine is ready)
pm-stats
```

`pm-stats` will:

1. Connect as a subscriber to the engine's PUB socket (:5556)
2. Connect as a second, independent subscriber to `pm-index`'s own PUB socket (:5558 by default) for `index.update` events — `pm-index` binds a separate endpoint from the engine, so this is a distinct ZMQ connection, not an additional topic filter on the engine socket
3. Wait briefly for ZMQ subscriptions to propagate, then request the symbol list from the engine via PUSH (:5555); on receipt, request a current book snapshot per symbol so opening bid/ask and initial price rows are captured even before new trading activity
4. Begin recording trades to `daily_stats` as they execute, detecting and logging any trades that never arrived (see `feed_gaps`)
5. Write an intraday snapshot on the first book update for a symbol, then at most one per `--snapshot-interval` (default 15 minutes) — a quiet symbol produces none
6. Write trade-by-trade records to `trade_log` immediately
7. Write private order lifecycle events to `order_events`
8. Write every received index update to `index_level_snapshots` and upsert the day's rollup into `index_daily_stats` — no exchange indexes configured means no `index.update` traffic and these two tables simply stay empty, which is expected and not an error
9. At engine shutdown, record the final close bid/ask to `daily_stats`
10. Answer `pm-stats-cli health` pings on its health socket (`127.0.0.1:5561` by default) for as long as it runs

Startup options:

Flag	Default	Description
`--db`	`<DATA_DIR>/stats.db`	Custom statistics database path
`--snapshot-interval`	`900` (15 min)	Seconds between `price_snapshots` rows per symbol; minimum `1`. Lower values give finer intraday resolution at the cost of more database writes. Anything under a second is **rejected at startup**: `price_snapshots` is keyed on `(ts, symbol)` at second precision, so it cannot hold more than one row per second per symbol and the extras would be silently discarded
`--keep-days`	`0` (keep everything)	Retention: keep the last N days of `trade_log`, `order_events`, `price_snapshots` and `feed_gaps`; older rows are deleted at startup and then once an hour. The daily rollups (`daily_stats`, `index_daily_stats`) are always kept. See Retention
`--keep-order-events-days`	as `--keep-days`	Retention for `order_events` alone, overriding `--keep-days` for them. Fractions are accepted: `0.1` is about 2.4 hours
`--human-gateways`	none	Comma-separated glob patterns naming the gateways that are people rather than bots, e.g. `'TRADER*'`
`--keep-human-order-events-days`	as `--keep-order-events-days`	Retention for the order events of `--human-gateways`
`--sql-trace`	off	Log executed SQLite statements from the stats writer connection — useful for debugging what `pm-stats` is actually writing
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v`, `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q`, `--quiet`	off	Reduce output to warnings/errors
`--log-target`	`server`	Where this process's own operational log records go: `server` (auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file`	—	Operational log file path — required when `--log-target file`
`--log-failover-timeout`	`30`	Grace window in seconds before falling back to a local log file once `pm-log-srv` becomes unreachable

Use `--db` if you want to record into a different location:

```
pm-stats --db /tmp/session_stats.db
```

Use `--snapshot-interval` to change how often intraday price snapshots are recorded:

```
pm-stats --snapshot-interval 60    # one-minute snapshots
pm-stats --snapshot-interval 300   # five-minute snapshots
pm-stats --snapshot-interval 3600  # hourly snapshots
```

Start order: `pm-stats` can be started before the engine. If the engine is not reachable it logs `pm-stats waiting for pm-engine to start` and retries the startup symbol request every second until it succeeds, so the opening book snapshots are still captured. Starting it after the engine is simply the quieter route.

17.6.1 Retention

`--keep-days N` bounds the per-event tables. Rows older than N days are deleted when `pm-stats` starts and then once an hour; the daily rollups are kept for good. The file stops growing once it holds N days, because SQLite reuses the freed pages.

Under an AI swarm, `order_events` are almost all of the file. Give them a retention of their own so a long run keeps hours of them and days of everything else:

```
pm-stats --keep-days 7 --keep-order-events-days 0.25
```

A retention short enough for a swarm would also delete the few order events of a person. `--human-gateways` names the gateways that are people, and `--keep-human-order-events-days` gives their order events a slower clock:

```
pm-stats --keep-days 30 --keep-order-events-days 0.25 \
         --human-gateways 'TRADER*' --keep-human-order-events-days 7
```

Matched gateways are still pruned — just later.

17.6.2 Exit codes

Code	Meaning
`0`	Stopped cleanly via `Ctrl-C` / `SIGTERM`
`1`	Startup failed, **or** the receive loop terminated unexpectedly and the process stopped recording

A non-zero exit means data was not being recorded. Supervise `pm-stats` on its exit code — a running process is not by itself evidence that recording is happening, and the log will carry a `pm-stats stopped recording: …` line.

17.6.3 Restarting mid-session

`pm-stats` can be restarted at any point during a trading day without losing the day's figures. On the first event it sees for a given symbol it rebuilds that symbol's running totals from `trade_log`, and the opening bid/ask from the existing `daily_stats` row, before applying the new event. Open, high, low, volume, trade count, VWAP and largest-trade all continue from where they were.

Two limits worth knowing:

- Anything the engine published **while `pm-stats` was down** was never received and cannot be recovered — ZeroMQ PUB/SUB has no replay. The rollup is consistent with what was recorded, not necessarily with what traded.
- For an index, `update_count` is rebuilt as the number of retained `index_level_snapshots` rows, which can be marginally lower than the number of updates originally received if any shared a millisecond.

17.6.4 One recorder per database

Exactly one `pm-stats` process may write to a given `stats.db`, and this is **enforced**. Two recorders against one file would each keep their own in-memory rollup and overwrite each other's `daily_stats` rows, producing figures that describe neither process — so a second one refuses to start:

```
$ pm-stats --db data/stats.db
<timestamp> ERROR edumatcher.stats.main - fatal startup error: another pm-stats process is already recording to data/stats.db (lock he
$ echo $?
1
```

The lock is an exclusive transaction on a sidecar `stats.db.lock` file, held for the life of the process and released on shutdown. A few consequences worth knowing:

- The `.lock` file appears alongside `stats.db`. It holds no data and can be deleted when no recorder is running.
- If `pm-stats` is killed with `SIGKILL`, the operating system releases the lock along with the process — there is no stale lock to clean up by hand.
- Readers are unaffected: `pm-stats-cli`, `pm-ticker` and the API Gateway never take this lock.
- Running two recorders deliberately is still fine, as long as each has its own `--db`.

17.7 Querying with pm-stats-cli

Once `pm-stats` has recorded data, use `pm-stats-cli` to query without SQL.

17.7.1 Basic Syntax

```
pm-stats-cli [--db PATH] [--format table|json|csv] [--timezone TZ] COMMAND [options]
```

Global options:

Flag	Default	Description
`--db`	`<DATA_DIR>/stats.db`	Path to statistics database
`--format`	`table`	Output format: `table` (human), `json` (structured), or `csv` (export)
`--no-header`	off	Omit the header row from `table` and `csv` output
`--timezone`	from the DB	Override the session timezone that `--date` and offset-less `--from`/`--to` resolve in. Defaults to the value the database was recorded with; overriding warns on mismatch

17.7.2 Row limits and truncation

Every list command has a `--limit`. When more rows match than the limit allows, the CLI prints a warning **to stderr** and gives you a cursor to continue from:

```
$ pm-stats-cli trades --date 2026-06-14 --limit 200
... 200 rows ...
[WARN] Output truncated at --limit 200. More rows exist; re-run with
       --after eyJ0cyI6IjIwMjYtMDYtMTRUMTA6MDA6MDAuMDAwKzAwOjAwIiwicm93aWQiOjIwMH0= for the next page.
```

Because the warning goes to stderr it never corrupts a redirected CSV or JSON payload — but it also means **you will not see it if you redirect both streams**. When a count has to be exact, either raise `--limit` beyond the expected row count or page through with `--after` until no warning appears.

`--after` takes the cursor verbatim and is available on `daily`, `snapshots`, `trades`, `order-events`, `index-daily` and `index-snapshots`.

17.7.3 Available Commands

`daily` — Daily OHLCV Summary

Show daily summary rows from `daily_stats`.

```
pm-stats-cli daily
pm-stats-cli daily --date 2026-06-14
pm-stats-cli daily --date 2026-06-14 --symbol AAPL
pm-stats-cli daily --wide  # include bid/ask and largest-trade columns
pm-stats-cli daily --limit 10

# Multi-day history for one symbol, oldest first
pm-stats-cli daily --symbol AAPL --from-date 2026-06-01 --to-date 2026-06-30
```

Options:

Option	Default	Description
`--date`	latest available	One trading date to query
`--from-date`	—	Start of an inclusive multi-day range
`--to-date`	—	End of an inclusive multi-day range
`--symbol`	all	Limit to one symbol
`--limit`	100	Maximum rows to return
`--after`	—	Continue from a previous run's truncation cursor
`--wide`	off	Include open/close bid/ask and largest-trade fields

daily returns a single date unless you ask for a range

With no date filter at all, `daily` returns rows for the **latest available date only** — `--limit` bounds how many *symbols* come back, not how many days. To get history across dates you must pass `--from-date` and/or `--to-date`; either bound may be omitted for an open-ended range. Range results are ordered oldest first, which is the order a chart plots them in. An explicit `--date` overrides a range if both are given.

Example output (default `table` format):

date	symbol	open_price	high_price	low_price	close_price	volume	trade_count	vwap
2026-06-14	AAPL	150	153.25	149.5	152.75	5000	12	151.82
2026-06-14	MSFT	414	418.5	413	417	3200	8	415.63

Null values render as an **empty cell** in `table` and `csv` output. Only `--format json` writes an explicit `null`.

`snapshots` — Intraday Price History

Show periodic price snapshots from `price_snapshots` for one symbol over a time range. The recording interval is set by `pm-stats --snapshot-interval` (default: 15 minutes).

```
pm-stats-cli snapshots --symbol AAPL
pm-stats-cli snapshots --symbol AAPL --date 2026-06-14
pm-stats-cli snapshots --symbol MSFT --from 2026-06-14T09:00:00+00:00 --to 2026-06-14T16:30:00+00:00
pm-stats-cli snapshots --symbol AAPL --limit 50
```

Options:

Option	Required	Default	Description
`--symbol`	Yes	—	Symbol to query
`--date`	No	all dates	Restrict to one trading date
`--from`	No	—	Start timestamp (inclusive, ISO format)
`--to`	No	—	End timestamp (inclusive, ISO format)
`--limit`	No	500	Maximum rows to return
`--after`	No	—	Continue from a previous run's truncation cursor

Example output:

```
ts                       | symbol | mid_price | best_bid | best_ask | pct_change
-------------------------+--------+-----------+----------+----------+-----------
2026-06-14T09:00:00+00:00 | AAPL   | 150.5     | 150      | 151      |
2026-06-14T09:15:00+00:00 | AAPL   | 151       | 150.5    | 151.5    | 0.3322
2026-06-14T09:30:00+00:00 | AAPL   | 151.25    | 151      | 151.5    | 0.1656
```

The first row's `pct_change` is empty because there is no previous snapshot to compare against (it is also empty when the previous row's `mid_price` was null). Timestamps always carry the `+00:00` offset — copy them verbatim into `--from`/`--to`.

`trades` — Trade-by-Trade History

Show individual trades from `trade_log` with optional filtering.

```
pm-stats-cli trades
pm-stats-cli trades --symbol AAPL
pm-stats-cli trades --symbol AAPL --date 2026-06-14
pm-stats-cli trades --symbol MSFT --from 2026-06-14T09:00:00+00:00 --to 2026-06-14T10:00:00+00:00
pm-stats-cli trades --limit 50
```

Options:

Option	Default	Description
`--symbol`	all	Limit to one symbol
`--date`	all dates	Restrict to one trading date
`--from`	—	Start timestamp (inclusive)
`--to`	—	End timestamp (inclusive)
`--limit`	200	Maximum rows to return
`--after`	—	Continue from a previous run's truncation cursor

Example output:

```
ts                           | trade_id         | symbol | price | quantity | aggressor_side | buy_gateway_id | sell_gateway_id
-----------------------------+------------------+--------+-------+----------+----------------+----------------+----------------
2026-06-14T09:00:01.000+00:00 | 000042-000000001 | AAPL   | 150   | 100      | BUY            | TRADER01       | MM01
2026-06-14T09:00:05.123+00:00 | 000042-000000002 | AAPL   | 150.5 | 50       | AUCTION        | MM01           | TRADER02
2026-06-14T09:00:10.456+00:00 | 000042-000000003 | AAPL   | 150.2 | 200      | SELL           | TRADER02       | TRADER01
```

`order-events` — Private Order Lifecycle Events

Show order lifecycle events from `order_events` for one gateway. The gateway is required because lifecycle history is private per participant.

```
pm-stats-cli order-events --gateway TRADER01
pm-stats-cli order-events --gateway TRADER01 --symbol AAPL
pm-stats-cli order-events --gateway TRADER01 --event-type FILL
pm-stats-cli order-events --gateway TRADER01 --date 2026-06-14 --limit 50
pm-stats-cli --format json order-events --gateway TRADER01 --from 2026-06-14T09:00:00+00:00
```

Options:

Option	Required	Default	Description
`--gateway`	Yes	-	Gateway ID that owns the private events
`--symbol`	No	all symbols	Restrict to one symbol

Option	Required	Default	Description
`--event-type`	No	all event types	Restrict to one normalized type — see `event_type` values. An unknown value is rejected at parse time
`--date`	No	all dates	Restrict to one trading date
`--from`	No	-	Start timestamp (inclusive)
`--to`	No	-	End timestamp (inclusive)
`--limit`	No	500	Maximum rows to return
`--after`	No	-	Continue from a previous run's truncation cursor

Example output:

```
seq | ts                           | event_type | order_id | gateway_id | symbol | side | order_type | tif | price | quantity | rema
----+------------------------------+------------+----------+------------+--------+------+------------+-----+-------+----------+-----
1   | 2026-06-14T09:00:00.100+00:00 | ACK        | O-AAPL-1 | TRADER01   | AAPL   | BUY  | LIMIT      | DAY | 150   | 100      | 100
2   | 2026-06-14T09:00:01.000+00:00 | FILL       | O-AAPL-1 | TRADER01   | AAPL   | BUY  |            |     |       |          | 0
```

All 21 `order_events` columns are printed (the table is wide). `price` and `fill_price` are display money, exactly as stored.

`order-lifecycle` — One Order's Event Trail

Show every lifecycle event for one order-like ID owned by a gateway. For combo, OCO, and quote events, the ID may be a `combo_id`, `oco_id`, or `quote_id` stored in the `order_id` column.

```
pm-stats-cli order-lifecycle --gateway TRADER01 --order-id O-AAPL-1
pm-stats-cli --format csv order-lifecycle --gateway TRADER01 --order-id O-AAPL-1
```

Options:

Option	Required	Default	Description
`--gateway`	Yes	-	Gateway ID that owns the private event trail
`--order-id`	Yes	-	Order, combo, OCO, or quote identifier to reconstruct

`symbols` — Symbol Discovery

List all symbols with data in the statistics DB.

```
pm-stats-cli symbols
pm-stats-cli symbols --date 2026-06-14  # symbols with data on a specific date
```

`dates` — Trading Date Discovery

List all available trading dates recorded in `daily_stats`.

```
pm-stats-cli dates
pm-stats-cli dates --symbol AAPL  # dates with data for a specific symbol
```

Example output:

```
date
----------
2026-06-15
2026-06-14
2026-06-13
```

`index-daily` — Daily Index OHLC Summary

Show daily index summary rows from `index_daily_stats`.

```
pm-stats-cli index-daily
pm-stats-cli index-daily --date 2026-06-14
pm-stats-cli index-daily --date 2026-06-14 --index-id EDU100
pm-stats-cli index-daily --wide  # include open/close aggregate market cap
pm-stats-cli index-daily --limit 10
```

Options:

Option	Default	Description
`--date`	latest available	One trading date to query
`--from-date`	—	Start of an inclusive multi-day range
`--to-date`	—	End of an inclusive multi-day range
`--index-id`	all indexes	Limit to one index
`--limit`	100	Maximum rows to return
`--after`	—	Continue from a previous run's truncation cursor
`--wide`	off	Include open/close aggregate market cap columns

As with `daily`, omitting every date filter returns the **latest date only**; pass `--from-date`/`--to-date` for a series across days.

Example output (default `table` format):

```
date       | index_id | open_level | high_level | low_level | close_level | close_session_state | update_count
-----------|----------|------------|------------|-----------|-------------|---------------------|-------------
2026-06-14 | EDU100   | 1042.1     | 1056.3     | 1040.05   | 1048.73     | CLOSED              | 512
```

`close_session_state` is `CLOSED` above, so `close_level` (`1048.73`) is confirmed as the final EOD print for that date — see Getting the EOD index level for a date below.

`index-snapshots` — Intraday Index Level History

Show every recorded index level update from `index_level_snapshots` for one index over a time range. Unlike `snapshots` for instruments, there is no configurable recording interval to tune — every `index.update` event `pm-stats` receives is recorded (`pm-index` has already rate-limited its own publications before `pm-stats` ever sees them).

```
pm-stats-cli index-snapshots --index-id EDU100
pm-stats-cli index-snapshots --index-id EDU100 --date 2026-06-14
pm-stats-cli index-snapshots --index-id EDU100 --from 2026-06-14T09:00:00+00:00 --to 2026-06-14T16:30:00+00:00
pm-stats-cli index-snapshots --index-id EDU100 --limit 50
```

Options:

Option	Required	Default	Description
`--index-id`	Yes	—	Index to query
`--date`	No	all dates	Restrict to one trading date
`--from`	No	—	Start timestamp (inclusive, ISO format)
`--to`	No	—	End timestamp (inclusive, ISO format)
`--limit`	No	500	Maximum rows to return
`--after`	No	—	Continue from a previous run's truncation cursor

Example output:

```
ts                           | index_id | level   | aggregate_cap | divisor | session_state   | day_open | day_high | day_low
-----------------------------+----------+---------+---------------+---------+-----------------+----------+----------+--------
2026-06-14T09:00:00.000+00:00 | EDU100   | 1042.1  | 7.35e+12      | 1.25    | OPENING_AUCTION | 1042.1   | 1042.1   | 1042.1
2026-06-14T09:00:05.500+00:00 | EDU100   | 1043.85 | 7.362e+12     | 1.25    | CONTINUOUS      | 1042.1   | 1043.85  | 1042.1
```

The table renderer prints floats with `%g`, which keeps six significant digits and switches to exponent form for large values — hence `7.35e+12`. Use `--format json` or `--format csv` for the full-precision numbers.

`instruments` — Instrument Reference Data

Show the tick scale for each symbol, from the `instruments` table.

```
pm-stats-cli instruments
pm-stats-cli instruments --symbol AAPL
```

Options:

Option	Default	Description
`--symbol`	all	Limit to one symbol

Use this to interpret raw SQL output: divide any tick column by `10^tick_decimals` to get display money.

`gaps` — Detected Feed Gaps

Show messages (trades and other subscribed topics) the recorder never received, from `feed_gaps`.

```
pm-stats-cli gaps
pm-stats-cli gaps --date 2026-06-14
pm-stats-cli gaps --from 2026-06-14T09:00:00Z --to 2026-06-14T12:00:00Z
```

Options:

Option	Default	Description
`--date`	all dates	Restrict to one trading date
`--from`	—	Start timestamp (inclusive)
`--to`	—	End timestamp (inclusive)
`--limit`	500	Maximum rows to return

Example output:

```
seq | ts                           | stream         | expected_id | received_id | missing_count
----+------------------------------+----------------+-------------+-------------+--------------
1   | 2026-06-14T09:14:02.115+00:00 | trade.executed | 3           | 6           | 3
2   | 2026-06-14T11:41:55.008+00:00 | trade.executed | 8           | 20          | 12
```

`No rows found.` is the healthy result. See the `feed_gaps` warning for what detection does and does not cover.

`health` — Recorder and Database Health

Check that `pm-stats` is alive and actually recording. By default two fast checks run:

- **Process** — a `pm-stats` process is running (found with `pgrep`).
- **ZMQ ping** — `pm-stats` answers a ping on its health socket (port `5561`) with its current UTC time. The reply comes from the recorder's own receive loop, so it proves the loop is running, not just that the process exists.

The database check is opt-in with `--extended`, because it runs a full-file `PRAGMA quick_check` that can take many seconds on a large `stats.db`. It checks that the database passes the integrity check, has the expected schema version, and accepts a (rolled-back) write. The check is allowed up to 45 seconds before it is reported as failed.

```
pm-stats-cli health
pm-stats-cli health --extended
pm-stats-cli health --extended --verbose
pm-stats-cli --format json health
pm-stats-cli health --quiet
```

```
pm-stats process: OK - pm-stats is running
stats database:  OK - skipped (pass --extended to check integrity and read/write access)
stats ZMQ ping:  OK - replied at 2026-10-17T14:23:37.797827+00:00
health: OK
```

Options:

Option	Description
`--extended`	Also check `stats.db` integrity, schema version and read/write access (up to 45 s)
`-v`, `--verbose`	List exactly which checks ran; with `--extended`, also print the database size and row counts per table
`-q`, `--quiet`	Print nothing; report only through the exit code

With `--extended --verbose` the report ends with the checks performed and the database figures:

```
Checks performed:
  pm-stats process (pgrep):      yes
  ZMQ health ping:                yes
  Database integrity/read-write: yes

Database stats:
  path: data/stats.db
  size: 412.3MB
  trade_log rows: 1843022
  order_events rows: 5210334
  price_snapshots rows: 96120
  daily_stats rows: 1240
  feed_gaps rows: 0
```

The exit code is `0` when healthy and `1` otherwise. With `--quiet` nothing is printed and the exit code is `0` when healthy or `-1` (255 in a shell) otherwise. `--format json` prints the same report as JSON.

The health socket binds to `127.0.0.1:5561` by default; set `EDUMATCHER_STATS_BIND_HOST` / `EDUMATCHER_STATS_HEALTH_PORT` to change where `pm-stats` binds it, and `EDUMATCHER_ENGINE_HOST` / `EDUMATCHER_STATS_HEALTH_PORT` to change where `pm-stats-cli` connects.

`index-ids` — Index Discovery

List all index IDs with data in the statistics DB.

```
pm-stats-cli index-ids
pm-stats-cli index-ids --date 2026-06-14  # indexes with data on a specific date
```

If no exchange indexes are configured, this returns no rows — that is expected, not an error.

17.7.4 Order Lifecycle History Queries

`order_events` can be queried directly with `pm-stats-cli` or through the API Gateway history endpoints. Use `pm-stats-cli` for local support, audit, and offline analysis. Use API Gateway history when a client should see only the private history for its authenticated trading credential.

Direct CLI examples:

```
pm-stats-cli order-events --gateway TRADER01 --symbol AAPL --event-type FILL --limit 50
pm-stats-cli order-lifecycle --gateway TRADER01 --order-id ORDER_ID
pm-stats-cli --format json order-events --gateway TRADER01 --date 2026-06-14
```

For API Gateway history queries, start the recorder, engine, stats database, and API gateway with matching config:

```
pm-engine --verbose
pm-stats --db data/stats.db
pm-api-gwy --instance desk
```

Then query order lifecycle history through HTTP with a trading API key:

```
curl -H 'Authorization: Bearer key-trader-demo' \
   'http://127.0.0.1:8080/api/v1/history/orders?symbol=AAPL&event_type=FILL&limit=50'
```

API filters for `/api/v1/history/orders`:

Query parameter	Required	Description
`symbol`	No	Restrict to one symbol
`event_type`	No	Restrict to one normalized type — see `event_type` values. An unknown value returns `400`
`date`	No	Restrict to one `YYYY-MM-DD` date based on `order_events.ts`
`from`	No	Inclusive ISO timestamp lower bound
`to`	No	Inclusive ISO timestamp upper bound
`limit`	No	Maximum rows to return, default `500`, maximum `5000`
`after`	No	Opaque keyset-pagination cursor from a previous response's `next_cursor`; fetches the next page

`/history/orders` (and `/history/fills`) responses include `next_cursor` when more rows are available. See the REST pagination contract for the full `count`/`has_more`/`next_cursor` contract, including which endpoints are exceptions to it.

To reconstruct one order's lifecycle, use the order ID path:

```
curl -H 'Authorization: Bearer key-trader-demo' \
   'http://127.0.0.1:8080/api/v1/history/orders/ORDER_ID'
```

For fill-only history, use the shortcut endpoint:

```
curl -H 'Authorization: Bearer key-trader-demo' \
   'http://127.0.0.1:8080/api/v1/history/fills?symbol=AAPL&date=2026-06-14'
```

Responses include an `events` array, `count`, and for list-style queries a `has_more` flag. Each event row mirrors the `order_events` table columns, so JSON output can be loaded directly into audit notebooks or support tooling.

Read-only API keys with `gateway_id: null` cannot query private order lifecycle history. Use a trading credential whose `gateway_id` owns the orders being investigated.

17.8 Output Formats

17.8.1 Table Format (default)

Human-readable aligned columns, designed for terminal viewing.

```
pm-stats-cli daily --date 2026-06-14
```

Good for: interactive exploration, demos, quick spot-checks.

17.8.2 JSON Format

Machine-readable structured output for automation and downstream tools.

```
pm-stats-cli --format json daily --date 2026-06-14 | jq '.[] | select(.symbol == "AAPL")'
```

Output:

```
[
  {
    "date": "2026-06-14",
    "symbol": "AAPL",
    "open_price": 150.0,
    "high_price": 153.25,
    ...
  },
  ...
]
```

Good for: scripts, APIs, BI tools, data pipelines.

17.8.3 CSV Format

Comma-separated values suitable for spreadsheets and data analysis tools.

```
pm-stats-cli --format csv trades --symbol AAPL --date 2026-06-14 > trades.csv
```

Output:

```
ts,trade_id,symbol,price,quantity,aggressor_side,buy_gateway_id,sell_gateway_id
2026-06-14T09:00:01.000+00:00,000042-000000001,AAPL,150.0,100,BUY,TRADER01,MM01
2026-06-14T09:00:05.123+00:00,000042-000000002,AAPL,150.5,50,AUCTION,MM01,TRADER02
```

Good for: Excel, Google Sheets, R/Python data frames, general-purpose analysis.

Use `--no-header` to suppress the header row:

```
pm-stats-cli --format csv --no-header trades --symbol AAPL >> all_trades.csv
```

17.9 Common Analyst Workflows

17.9.1 End-of-Day Summary Report

Generate a quick summary of all symbols for a given trading date:

```
pm-stats-cli daily --date 2026-06-14 --wide
```

This shows open/close prices, open/close best bid and ask, volume, trade count, turnover, VWAP and the largest trade for every symbol.

Follow-up questions: - Which symbol had the highest volume? - What was the spread between open bid and close bid? - Did any symbol experience a large single trade?

17.9.2 Intraday Price Volatility Analysis

Check mid-price movement for one symbol throughout the day:

```
pm-stats-cli snapshots --symbol AAPL --date 2026-06-14 | head -20
```

Look at the `pct_change` column to spot: - Periods of high volatility (large jumps) - Periods of stagnation (flat pricing) - Potential technical support/resistance levels - Times when the book was empty (null bids/asks)

17.9.3 Trade Flow Analysis

Examine all trades for a symbol to identify patterns:

```
pm-stats-cli --format csv trades --symbol AAPL --date 2026-06-14 > aapl_trades.csv
```

Then analyze in a spreadsheet or Python:

```
import pandas as pd
trades = pd.read_csv('aapl_trades.csv', parse_dates=['ts'])
trades['hour'] = trades['ts'].dt.hour

# Trades per hour
print(trades.groupby('hour').size())

# Average trade size
print(trades.groupby('hour')['quantity'].mean())

# Who are the active participants?
print(trades['buy_gateway_id'].value_counts() + trades['sell_gateway_id'].value_counts())
```

17.9.4 Participant Performance Analysis

Export trade logs and group by participant to see:

```
pm-stats-cli --format json trades --date 2026-06-14 | jq '.[] | {buyer: .buy_gateway_id, seller: .sell_gateway_id, price: .price, qty
```

Then aggregate in your tool of choice: - How many trades did each participant execute? - What was their average trade size? - Did they tend to be buyers or sellers?

17.9.5 Multi-Day Price Trends

Compare the same symbol across multiple trading dates. A date **range** is required — without one, `daily` returns only the latest date:

```
pm-stats-cli --format csv daily --symbol AAPL \
  --from-date 2026-01-01 --to-date 2026-06-30 --limit 1000 > aapl_history.csv
```

Either bound may be omitted for an open-ended range, so this exports everything recorded for the symbol:

```
pm-stats-cli --format csv daily --symbol AAPL --from-date 1970-01-01 --limit 1000 \
  > aapl_history.csv
```

Rows come back oldest first. Check stderr for a truncation warning — if one appears, raise `--limit` or page with `--after`.

This gives you historical OHLCV to track trends, seasonal patterns, or support/resistance zones over time. Remember that prices are **unadjusted**: a series spanning a corporate action is discontinuous and returns computed across it will be wrong.

17.9.6 Index Level History

Chart or export an exchange index's intraday level trajectory:

```
pm-stats-cli index-snapshots --index-id EDU100 --date 2026-06-14 | head -20
```

Look for the same signals `snapshots` gives for instruments — periods of rapid level movement, gaps that may indicate a connectivity issue between `pm-index` and `pm-stats`, and the `session_state` column shifting from `OPENING_AUCTION`/`CONTINUOUS`/`CLOSED`.

Compare the index's daily performance across dates the same way you would for a symbol — again, a date range is required for more than one day:

```
pm-stats-cli --format csv index-daily --index-id EDU100 \
  --from-date 2026-01-01 --to-date 2026-06-30 --limit 1000 > edu100_history.csv
```

17.9.7 Getting the EOD index level for a date

To look up an index's official end-of-day closing level for a specific date, query `index-daily` for that date and index:

```
pm-stats-cli index-daily --date 2026-06-14 --index-id EDU100
```

```
date       | index_id | open_level | high_level | low_level | close_level | close_session_state | update_count
-----------|----------|------------|------------|-----------|-------------|---------------------|-------------
2026-06-14 | EDU100   | 1042.1     | 1056.3     | 1040.05   | 1048.73     | CLOSED              | 512
```

`close_level` is the answer. For any date in the past this is always safe to read directly — no more `index.update` events can arrive for a date once it has rolled over, so `close_level` cannot change after the fact.

If you are querying **today's** date, confirm the row is actually final before trusting it, since `close_level` is updated on every tick and is a live "last level so far" until the session closes:

```
pm-stats-cli --format json index-daily --index-id EDU100 \
  | python3 -c "import json,sys; r=json.load(sys.stdin)[0]; print(r['close_level'], r['close_session_state'])"
```

If `close_session_state` prints `CLOSED`, `close_level` is the final EOD print. Any other value (e.g. `CONTINUOUS`, `OPENING_AUCTION`) means the session is still running and `close_level` will keep moving — re-query after the close, or wait for `close_session_state` to flip to `CLOSED`.

For scripting, JSON output makes this a one-line check:

```
pm-stats-cli --format json index-daily --date 2026-06-14 --index-id EDU100 \
  | python3 -c "
import json, sys
row = json.load(sys.stdin)[0]
if row['close_session_state'] != 'CLOSED':
    sys.exit('not final yet: ' + row['close_session_state'])
print(f\"EOD close for {row['date']} {row['index_id']}: {row['close_level']}\")
"
```

17.9.8 Validation — Did the Trade Complete Correctly?

After a trading session ends, verify key metrics:

0. **Check the recording is complete — do this first:**

   ```
   pm-stats-cli gaps --date 2026-06-14
   ```

   `No rows found.` means no loss was detected. Any row means the figures below are computed from an incomplete trade record, and `volume`, `turnover` and `vwap` for the affected symbols will be understated. There is no way to recover the missing trades after the fact — ZeroMQ PUB/SUB has no replay — so treat the day's numbers as approximate and investigate why `pm-stats` fell behind.

   `pm-stats` also logs a session total at `INFO` (visible with `-v`) when it shuts down. It lists every counter that was non-zero, in alphabetical order:

   ```
   session totals: book_topics=10432, messages_received=12905, trades_persisted=2471
   ```

1. **Check daily summary recorded:**

   ```
   pm-stats-cli daily --date 2026-06-14
   ```

   Verify: all symbols present, volume > 0, open/close prices are reasonable.

2. **Check trade count:**

   ```
   # --no-header so every line is a trade; --limit must exceed the expected
   # count, or the result is silently truncated.
   pm-stats-cli --format csv --no-header trades --date 2026-06-14 --limit 100000 | wc -l
   ```

   Verify: matches expected number from the trading floor. If a `[WARN] Output truncated` line appears on stderr, the count is **not** complete — raise `--limit` and re-run.

   The authoritative count without any limit concerns:

   ```
   pm-stats-cli --format json daily --date 2026-06-14 \
     | python3 -c "import json,sys; print(sum(r['trade_count'] for r in json.load(sys.stdin)))"
   ```

3. **Check for any empty books:**

   ```
   pm-stats-cli --format json snapshots --symbol AAPL --date 2026-06-14 \
     | python3 -c "
   import json, sys
   for row in json.load(sys.stdin):
       if row['best_bid'] is None or row['best_ask'] is None:
           print(row['ts'], row['best_bid'], row['best_ask'])
   "
   ```

   Empty books during active trading hours may indicate a problem. Use JSON here: `table` and `csv` render a null as an

empty cell, so a text search for `null` finds nothing.

4. **Check largest trade vs. typical trade size:**

```
pm-stats-cli daily --wide --date 2026-06-14 --symbol AAPL
```

Look at `largest_trade_qty` vs. average (`volume / trade_count`). Outliers warrant investigation.

17.9.9 Order Lifecycle Investigation

Use `order_events` when the question is about what happened to a submitted order rather than what trades printed to the market.

Examples:

```
# All recent events for a gateway
pm-stats-cli order-events --gateway TRADER01 --limit 100

# One order from ACK through fills, cancels, expiry, or rejection
pm-stats-cli order-lifecycle --gateway TRADER01 --order-id ORDER_ID

# Fill-only view for one symbol and date
pm-stats-cli order-events --gateway TRADER01 --symbol AAPL --event-type FILL --date 2026-06-14
```

Use this workflow to answer:

- Was the order accepted or rejected? (`ACK` / `REJECT`, and equally `COMBO_ACK` / `COMBO_REJECT`, `OCO_ACK` / `OCO_REJECT`, `QUOTE_ACK` / `QUOTE_REJECT`)
- Did an amend reset priority?
- Which fills belong to this order ID?
- Was the order cancelled or expired? (`CANCEL`, `EXPIRE`, and `OCO_CANCEL` for an OCO leg)
- Was it linked to a combo or OCO group? (`combo_parent_id`, `oco_group_id`)
- Does API Gateway history match the live private WebSocket events seen by the client?

```
# Every rejection across all event families for one gateway
for t in REJECT COMBO_REJECT OCO_REJECT QUOTE_REJECT; do
  pm-stats-cli order-events --gateway TRADER01 --event-type "$t"
done
```

17.10 Integration with Other Tools

17.10.1 Combining with pm-ticker

`pm-ticker` uses `data/stats.db` to display OHLCV and volume context in its live display.

To verify `pm-stats` is recording correctly while `pm-ticker` runs:

```
# Terminal 1: Start engine
pm-engine --verbose

# Terminal 2: Start stats
pm-stats

# Terminal 3: Start ticker (reads from stats DB)
pm-ticker

# Terminal 4: Live-check stats as trades occur
watch -n 5 'pm-stats-cli daily | tail -5'
```

17.10.2 Exporting to BI Tools

Example: Export daily summaries to a cloud data warehouse:

```
# Export as CSV — pass a date range, or you get only the latest date
pm-stats-cli --format csv daily --from-date 2026-01-01 --limit 10000 > daily_stats.csv

# Upload to BigQuery, Redshift, Snowflake, etc.
bq load my_dataset.daily_stats daily_stats.csv

# Or load into local database
sqlite3 analysis.db <<EOF
.mode csv
.import --skip 1 daily_stats.csv daily_stats
EOF
```

Carry the session timezone with the export

`stats.db` records the session timezone in `stats_meta`, but a CSV export does not — and `date` cannot be interpreted without it. Read it before exporting and keep it with the file:

```
sqlite3 data/stats.db \
  "SELECT value FROM stats_meta WHERE key = 'session_timezone'"
```

Tick size and price precision **are** now recorded, in the `instruments` table — export it alongside your data and the prices become interpretable on their own. Currency still is not: EduMatcher has no currency model, and `instruments.currency` is a reserved NULL rather than a value to read.

17.10.3 Python / Pandas Integration

Query and analyze directly in Python:

```
import subprocess
import json
import pandas as pd

# Get daily stats as JSON
result = subprocess.run(
    ['pm-stats-cli', '--format', 'json', 'daily', '--date', '2026-06-14'],
    capture_output=True,
    text=True
)

daily = pd.DataFrame(json.loads(result.stdout))

# Pivot to wide format for correlation analysis
daily_pivot = daily.set_index('symbol')
print(daily_pivot[['open_price', 'close_price', 'volume']])

# Calculate returns
daily['return_pct'] = (daily['close_price'] - daily['open_price']) / daily['open_price'] * 100
print(daily[['symbol', 'return_pct']])
```

17.11 Troubleshooting

17.11.1 No data recorded — where did the trades go?

1. **Verify `pm-stats` is running:**

   ```
   ps aux | grep pm-stats
   ```

   If not running, start it.

2. **Check that `pm-stats` connected to the engine:**

   ```
   pm-engine --verbose
   ```

   Look for log messages showing that `pm-stats` sent a `book.snapshot_request`.

3. **Verify the database file exists and has the right tables:**

```
sqlite3 data/stats.db ".tables"
```

You should see: `daily_stats`, `price_snapshots`, `trade_log`, `order_events`, `index_daily_stats`, `index_level_snapshots`, `feed_gaps`, `instruments`, and `stats_meta`.

4. **Check for recent trades:**

```
pm-stats-cli trades --limit 5
```

If empty, no trades have executed yet. Execute a test trade first.

5. **Check for order lifecycle history:**

```
sqlite3 data/stats.db "SELECT ts,event_type,order_id,gateway_id,symbol FROM order_events ORDER BY seq DESC LIMIT 5;"
```

If empty, no private order lifecycle topics have reached `pm-stats` yet. Submit, amend, cancel, or fill an order while `pm-stats` is running.

17.11.2 No index data recorded — where did the index updates go?

1. **Confirm the exchange actually has an index configured.** If `engine_config.yaml` has no `indexes:` block, `pm-index` publishes nothing and `index_daily_stats`/`index_level_snapshots` staying empty is correct behavior, not a bug.

2. **Verify `pm-index` is running:**

```
ps aux | grep pm-index
```

`pm-stats` connects to `pm-index`'s own PUB socket (default port 5558), separate from the engine's PUB socket — if `pm-index` isn't running, there is nothing for `pm-stats` to receive.

3. **Check for recorded index updates:**

```
pm-stats-cli index-ids
pm-stats-cli index-snapshots --index-id EDU100 --limit 5
```

If `index-ids` returns nothing, `pm-stats` has not received any `index.update` event yet — confirm `pm-index` is up and has finished its own startup index calculation.

4. **Check `pm-stats` logs at `-vv`/`DEBUG`** for `recorded index update index_id=...` lines (they are logged at `DEBUG`, so `-v` will not show them), or run with `--sql-trace` to see the underlying `INSERT`/`UPDATE` statements against `index_level_snapshots`/`index_`

17.11.3 Queries return "No rows found" but I know data should exist

1. **Check the date format:**

```
pm-stats-cli dates  # What dates are actually in the DB?
```

Use the exact date returned, e.g., `--date 2026-06-14`.

2. **Verify the symbol exists:**

```
pm-stats-cli symbols
```

`pm-stats-cli` upper-cases `--symbol`, `--gateway` and `--index-id` before querying, so `--symbol aapl` and `--symbol AAPL` behave identically. The values stored in the database are whatever the engine published — if those are not upper-case, the CLI cannot match them and you must query with `sqlite3` directly.

3. **Check the session timezone.** `--date` selects a *trading day* in the `--timezone` you pass. If `pm-stats-cli` uses a different `--timezone` than `pm-stats` recorded with, `--date` resolves to the wrong window and returns too few rows, too many, or none — with no error:

```
pm-stats-cli --timezone Europe/Stockholm trades --date 2026-06-14
```

4. **Check the time window for snapshots/trades:**

```
pm-stats-cli snapshots --symbol AAPL --date 2026-06-14
```

If using `--from` / `--to`, both bounds are inclusive instants. A bound without an offset is read as session-local time; append `Z` or an explicit offset to be unambiguous.

17.11.4 Database is locked or "unable to open"

1. **`stats.db` runs in WAL mode**, so one writer (`pm-stats`) and any number of readers (`pm-stats-cli`, `pm-ticker`, the API Gateway) proceed concurrently without blocking each other. Both the writer and readers also set a 5-second `busy_timeout` to absorb the brief exclusive lock taken during a WAL checkpoint.
2. **A second writer is still excluded.** Attempting to write directly with `sqlite3` while `pm-stats` is running can still produce a lock error, and running two `pm-stats` processes against one file corrupts the daily rollups (see "One recorder per database" above). Use `pm-stats-cli` for queries and keep a single recorder per file.
3. **If you need to copy the DB for backup**, do it online — do not `cp` the file. In WAL mode a plain `cp` can capture a torn database without the companion `-wal` file:

```
# Safe while pm-stats is running; produces a consistent, compacted copy
sqlite3 data/stats.db "VACUUM INTO 'data/stats_backup.db'"

# Equivalent, and works on older sqlite3 builds
sqlite3 data/stats.db ".backup 'data/stats_backup.db'"
```

Growth and retention

By default `trade_log`, `order_events`, `price_snapshots` and `feed_gaps` are append-only and unbounded. Set `--keep-days N` (see Retention) to cap them, and monitor the file size either way. Note that `--keep-days` also prunes `feed_gaps`, which is the only record that a session was incomplete: pruning it silently turns a known-lossy dataset back into an apparently clean one. If you need that record for longer than the other tables, archive first (copy out with `VACUUM INTO`; see the cursor caveat under Row identity and pagination cursors) and keep the copy.

17.11.5 Snapshot times seem wrong or are missing

- The **first** `book.*` message for a symbol always writes a snapshot. After that, a snapshot is written when a `book.*` message arrives **and** the configured interval has elapsed since the last one for that symbol.
- The default interval is **15 minutes** (`--snapshot-interval 900`). If you need finer resolution, start `pm-stats` with a smaller value, e.g. `--snapshot-interval 60` for one-minute snapshots.
- If trading is light and no book updates occur during the interval, no snapshot is recorded for that period. This is by design — snapshots only record when the market moves.

To verify:

```
pm-stats-cli snapshots --symbol AAPL --date 2026-06-14 | awk '{print $1}' | uniq -c
```

You should see roughly one entry every 15 minutes. Large gaps indicate periods with no trading.

17.11.6 VWAP calculation looks wrong

VWAP is recalculated on every trade and stored at that moment. The final VWAP for the day is stored in `daily_stats` after the close.

To verify VWAP manually:

```
# --limit must exceed the day's trade count, or you will be comparing a
# partial recomputation against the full-day figure and they will not match.
pm-stats-cli --format csv trades --symbol AAPL --date 2026-06-14 --limit 100000 | \
  awk -F, 'NR>1 {qty_sum += $5; price_qty += $4*$5} END {print price_qty/qty_sum}'
```

This calculates $\sum(price \times qty) / \sum(qty)$ from the trade log. Compare it to the value in `daily_stats`:

```
pm-stats-cli daily --symbol AAPL --date 2026-06-14
```

Both the awk figure and `pm-stats-cli daily` are in **display money**, because `pm-stats-cli` converts the prices (including `vwap` and `turnover`) on the way out. Only if you read `daily_stats.vwap` with raw SQL is it in **ticks**; divide it by `10^tick_decimals` before comparing, or the two will differ by a factor of a hundred and look catastrophically wrong when nothing is. To compare everything in one place:

```
pm-stats-cli --format json daily --symbol AAPL --date 2026-06-14 --wide \
  | python3 -c "
import json, sys
row = json.load(sys.stdin)[0]
scale = 10 ** row['tick_decimals']
print('vwap  (money):', row['vwap'])          # already converted on output
print('vwap  (ticks):', row['vwap'] * scale)
print('turnover/vol :', row['turnover'] / row['volume'])  # turnover is money too
"
```

Because prices are stored as exact integers, the recomputation should now agree to full precision rather than merely to within rounding — `turnover` is an integer sum that does not depend on the order the trades were added.

If the figures disagree, check in this order:

1. **Are you comparing ticks against money?** By far the most common cause.
2. **Was the output truncated?** A `[WARN]` line on stderr means the awk sum covered only part of the day.
3. **Does `pm-stats-cli --timezone`, if passed, match the `locale.session_timezone` `pm-stats` recorded with?** If not, `--date` selects a different set of trades than the one folded into the `daily_stats` row, and the two will never match.

17.12 See Also

In this book:

- Audit Trail — `pm-audit-cli` for querying the full event log
- Persistence — where all data files are stored

Reference Manual:

- Processes — pm-stats and pm-stats-cli — full process documentation
- Processes — pm-ticker — live ticker that uses statistics data

CHAPTER 18

Market Index (pm-index)

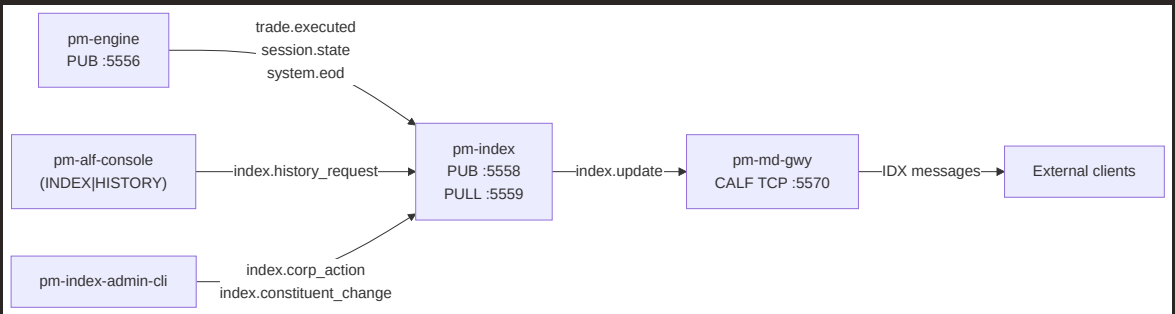
Learning objectives

After reading this page you will understand:

- What pm-index does and how it fits into the EduMatcher process model
- How to start pm-index and which CLI options are available
- How to configure one or more indices in the engine config (engine_config.yaml)
- How cap-weighted index calculation and divisor normalisation work
- How to apply corporate actions without disrupting the index level
- How to query the current index value, structural/audit records, and level/EOD time-series history
- Where state and history files are stored and how recovery works

18.1 What this process is

pm-index is a standalone index calculation process. It subscribes to trade events published by pm-engine and recomputes all configured indices in real time. Calculated values are re-published on a dedicated ZMQ socket so the market-data gateway (pm-md-gwy) can forward them to external subscribers over the CALF INDEX channel.



pm-index does **not** connect to the engine PULL socket. It never sends order or session commands to the engine — it only listens and publishes.

18.2 Prerequisites

- pm-engine running
- At least one index defined in engine_config.yaml, compiled and installed with pm-config-deploy (see Engine Configuration)
- Every constituent symbol listed in symbols: with outstanding_shares set and at least one of last_buy_price / last_sell_price (its reference price)

pm-index can start before or after pm-engine; the ZMQ subscriber will reconnect automatically. Order only matters for the first few seconds of a fresh session — if pm-index misses early trades the reference prices from config are used as fallback until

real trades arrive.

18.3 Starting pm-index

Installed mode:

```
pm-index
pm-index --reset
```

Developer / Poetry mode:

```
poetry run pm-index
poetry run pm-index --reset
```

`pm-index` takes no config-file argument. Like the engine, it reads the compiled configuration that `pm-config-deploy` installed (`<DATA_DIR>/ref_data/engine_config.json`), so it always agrees with the running engine about constituents and reference prices. If no compiled configuration is deployed it logs a warning (`no compiled configuration at ... — no indices will be calculated. Run pm-config-deploy to install one.`) and calculates nothing.

18.3.1 CLI options

Option	Default	Description
`--reset`	off	Delete persisted state files and re-initialise all indices from config
`--version`	—	Print the version and exit
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	Accepted for symmetry with the other processes; `WARNING` is already the default
`--log-target`	`server`	Where this process's own log records go: `server` (auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file`	—	Operational log file path — required when `--log-target file`
`--log-failover-timeout`	`30`	Grace window in seconds before falling back to a local log file once `pm-log-srv` becomes unreachable

`--reset` is useful after:

- changing constituent lists or `base_value` in config
- replacing a corrupted state file
- starting a new multi-day session from a clean baseline

Use -reset with care

`--reset` deletes the divisor and all persisted last prices for every configured index. The index restarts from scratch using only the reference prices in the engine config. The structural/audit JSONL file is **not** deleted; only the small state JSON file is removed.

18.4 Configuration

All index configuration lives in the top-level `indices:` section of `engine_config.yaml` (deployed with `pm-config-deploy`). See Engine Configuration for the full field reference; a complete YAML example is shown below.

18.4.1 Minimal example

```
symbols:
  AAPL:
    tick_decimals: 2
    outstanding_shares: 15000000000
  MSFT:
    tick_decimals: 2
    outstanding_shares: 7400000000
  TSLA:
    tick_decimals: 2
    outstanding_shares: 3200000000

participants:
  - id: TRADER01
    role: TRADER
    disconnect_behaviour: CANCEL_ALL

indices:
  - id: EDU100
    description: "EduMatcher broad index"
    base_value: 1000.0
    publish_interval_sec: 1.0
    history_file: data/indexes/EDU100_history.jsonl
    state_file: data/indexes/EDU100_state.json
    constituents:
      - AAPL
      - MSFT
      - TSLA
```

18.4.2 Field reference

Field	Type	Default	Description
`id`	string	—	Alphanumeric index identifier; used in events, file names, and gateway commands
`description`	string	—	Human-readable label emitted in index events; required, non-empty
`base_value`	float	`1000.0`	Starting index level on first launch; must be `> 0`
`publish_interval_sec`	float	`1.0`	Throttle on how often `index.update` is broadcast; must be `> 0`
`history_file`	string	`data/indexes/<ID>_history.jsonl`	Append-only JSONL **structural/audit trail** — corporate actions and constituent changes only, not level history (see State and History)
`state_file`	string	`data/indexes/<ID>_state.json`	Checkpoint file for divisor and last prices
`constituents`	list	—	Symbols included in the index

Constraints:

- Maximum **5** indices per config file
- `id` must be alphanumeric, and `id` values must be unique
- `constituents` must be a non-empty list with no duplicates
- Every constituent symbol must appear in `symbols:` with `outstanding_shares` set
- Relative `history_file` / `state_file` paths (including a leading `data/`) are resolved inside the data directory (`EDUMATCHER_DATA_DIR`); absolute paths are used as given

Index eligibility is set at listing (IPO)

`outstanding_shares` is what makes a symbol index-eligible, and it is part of the symbol's initial listing (IPO). Set it when you list the symbol; a constituent without a positive `outstanding_shares` is rejected at startup. Share counts change only through corporate actions (splits, issuance), not intra-day trading.

18.4.3 Generating with `pm-config-gen`

`pm-config-gen` can emit the full `indices:` block automatically:

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 OPS01:ADMIN MM01:MARKET_MAKER \
  --outstanding-shares AAPL:15000000000 \
  --outstanding-shares MSFT:7400000000 \
  --outstanding-shares TSLA:3200000000 \
  --seed-mm-mid-range 100:400 \
  --seed-last-prices-from-mm \
  --no-mm-seed-quotes \
  --sessions-enabled \
  --index EDU100:"EduMatcher broad index" \
  --index-constituents EDU100:AAPL,MSFT,TSLA \
  --output engine_config.yaml
```

File paths are derived from the index ID automatically when `--index-history-file` and `--index-state-file` are not specified.

Why the market maker and price-seeding flags are needed

`pm-index` refuses a constituent that has neither `last_buy_price` nor `last_sell_price`, since it sets the initial divisor from them. The only way `pm-config-gen` writes real prices is to seed them from a market-maker midpoint, so the command adds a `MARKET_MAKER` participant with `--seed-mm-mid-range MIN:MAX --seed-last-prices-from-mm`. `--no-mm-seed-quotes` keeps the book empty at startup despite that participant. Without them `pm-config-gen` stops with an error rather than writing a config `pm-index` could not start on; the same applies to a constituent with no `--outstanding-shares`. Install the result with `pm-config-deploy engine_config.yaml`.

18.5 How Calculation Works

18.5.1 Cap-weighted formula

Each index uses **market-capitalisation weighting**. A constituent's influence on the index level is proportional to its total market value relative to all other constituents:

$$\text{aggregate_cap} = \sum_{\text{sym} \in \text{constituents}} \text{last_price(sym)} \times \text{outstanding_shares(sym)}$$

$$\text{index_level} = \frac{\text{aggregate_cap}}{\text{divisor}}$$

A company trading at $200 with 1 billion shares has a market cap of $200 billion. A company trading at $100 with 500 million shares has a market cap of $50 billion. The first company contributes four times as much to the index level as the second.

18.5.2 Divisor initialisation

On first startup (no persisted state), the divisor is chosen so the initial index level equals `base_value`:

$$\text{initial_divisor} = \frac{\text{aggregate_cap at launch}}{\text{base_value}}$$

Example with three stocks and `base_value = 1000`:

Symbol	Price	Outstanding shares	Market cap
AAPL	209.50	15,000,000,000	3,142,500,000,000
MSFT	415.00	7,400,000,000	3,071,000,000,000
TSLA	248.00	3,200,000,000	793,600,000,000
Total			**7,007,100,000,000**

```
initial_divisor = 7,007,100,000,000 / 1000.0 = 7,007,100,000
```

The divisor is a large number, but the ratio `aggregate_cap / divisor` always yields a level close to the configured `base_value`. After the first launch it evolves only through explicit corporate-action adjustments.

18.5.3 Update trigger

`pm-index` recalculates after every `trade.executed` message for a constituent symbol. Trades in non-constituent symbols are ignored. The calculation is O(N) in the number of constituents — fast enough for any educational exchange.

18.5.4 Reference prices

When no trade has occurred for a constituent since startup, a reference price seeded from the engine config is used: the average of `last_buy_price` and `last_sell_price` when both are set, or whichever one is set if only one is present. A symbol with neither field set fails to load as an index constituent. Once the first real trade arrives the reference price is replaced by the actual trade price.

18.5.5 Publish throttle

Recalculation happens on every trade, but publishing is throttled to `publish_interval_sec` (default 1 s). The internal level is always up to date; the throttle controls only how often `index.update` messages are broadcast.

18.5.6 Intraday OHLC

`pm-index` tracks day open, high, and low internally. They are cleared at every `OPENING_AUCTION` or `CONTINUOUS` session transition and start again from the next level update; the day is finalised with a closing value when the session reaches `CLOSED`.

18.6 Corporate Actions

Corporate actions change a company's share structure without destroying value. Without divisor adjustment, a 2-for-1 stock split would make the index drop by roughly half — which would be wrong because no value was lost.

The divisor is adjusted so the **index level is preserved at the moment of the action**. Future price movements then reflect real value changes.

18.6.1 Divisor adjustment formula

$$\text{new_divisor} = \text{old_divisor} \times \frac{\text{new_aggregate_cap}}{\text{old_aggregate_cap}}$$

This guarantees continuity:

$$\frac{\text{new_cap}}{\text{new_divisor}} = \frac{\text{new_cap}}{\text{old_divisor} \times \frac{\text{new_cap}}{\text{old_cap}}} = \frac{\text{old_cap}}{\text{old_divisor}} \quad \checkmark$$

18.6.2 Supported corporate actions

Action	action value	Required parameters	Effect
Stock split	SPLIT	ratio_numerator, ratio_denominator	Increases shares, reduces price by the same factor; divisor corrected for integer-rounding drift
Cash dividend	CASH_DIVIDEND	dividend_per_share	Reduces price by the dividend amount; divisor adjusted to preserve index level
Shares issuance	SHARES_ISSUANCE	new_shares_outstanding	Sets the new total share count outright (not a delta); divisor rescaled to preserve index level — typically upward for a genuine issuance, but the code does not require new_shares_outstanding to exceed the current count, so supplying a smaller value rescales the divisor downward instead

18.6.3 Applying corporate actions

Use pm-index-admin-cli to apply corporate actions to a running pm-index process:

```
pm-index-admin-cli --id OPS01 split    --index EDU100 --sym AAPL --ratio 2:1
pm-index-admin-cli --id OPS01 dividend --index EDU100 --sym MSFT --amount 2.50
pm-index-admin-cli --id OPS01 shares   --index EDU100 --sym TSLA --new-shares 3500000000
pm-index-admin-cli --id OPS01 shares   --index EDU100 --sym TSLA --delta -200000000   # buy-back
```

Each mutating subcommand prints a confirmation prompt before sending (skip it with -y/--yes), supports --dry-run to preview the outbound payload without sending it, and blocks for the index.corp_action_ack.{gateway_id} response. pm-index applies the action in-process, immediately publishes an updated index value live, and writes a CORP_ACTION record to the structural/audit history file. See Index Administration (pm-index-admin-cli) for the full subcommand reference, --dry-run output, and exit codes.

Apply corporate actions before the market opens

It is safest to apply corporate actions during PRE_OPEN before trading starts for the day. Applying them during continuous trading means the index will have a brief moment where the pre-adjustment divisor is used for one more trade before the action takes effect.

Under the hood: ExchangeCommandClient

pm-index-admin-cli is a thin wrapper over the ExchangeCommandClient class (edumatcher.commands.client) — the same class pm-admin-cli uses internally for its engine commands. The CLI calls its index_corp_action() method directly:

```
from edumatcher.commands import ExchangeCommandClient

client = ExchangeCommandClient("GW_ADMIN")

client.index_corp_action(
    "EDU100", "SPLIT", "AAPL",
    ratio_numerator=2, ratio_denominator=1,
)
client.index_corp_action(
    "EDU100", "CASH_DIVIDEND", "MSFT",
    dividend_per_share=2.50,
)
client.index_corp_action(
    "EDU100", "SHARES_ISSUANCE", "TSLA",
    new_shares_outstanding=3500000000,
)

client.close()
```

Unlike the engine's PUSH socket, `pm-index`'s PULL socket has no `connect()`/auth handshake — see Index Admin CLI — Why there is no connect() step. Writing a script against this class directly still works and is how `pm-index-admin-cli` itself is implemented, but the CLI adds validation, a confirmation prompt, and `--dry-run`, so it's the preferred path for interactive/manual use.

18.6.4 Constituent changes

Constituents can be added or removed without restarting `pm-index`, also via `pm-index-admin-cli`:

```
pm-index-admin-cli --id OPS01 add    --index EDU100 --sym AMZN --shares 10500000000 --price 195.00
pm-index-admin-cli --id OPS01 delist --index EDU100 --sym TSLA
```

Adding a constituent adjusts the divisor so the index level does not change at the moment of addition. Delisting a constituent adjusts the divisor so the remaining constituents continue smoothly.

Every corporate action and constituent change is written to the `history_file` as a `CORP_ACTION`, `ADD_CONSTITUENT`, or `DELIST` record — queryable with `pm-index-cli`.

18.7 State and History

18.7.1 State file

`pm-index` writes a small JSON checkpoint file at startup (right after computing the initial level), after every EOD finalization, on every session phase change, on shutdown, after any corporate action, rebalance or constituent change (add/delist), and — while trading — at most every 30 seconds if prices have moved since the last write. A trade does not write the file by itself.

```
{
  "index_id": "EDU100",
  "description": "EduMatcher broad index",
  "divisor": 7007100000.0,
  "constituents": ["AAPL", "MSFT", "TSLA"],
  "outstanding_shares": {
    "AAPL": 15500000000,
    "MSFT": 7430000000,
    "TSLA": 3180000000
  },
  "last_prices": {
    "AAPL": 211.3,
    "MSFT": 418.5,
    "TSLA": 251.0
  },
  "day_open": 1002.45,
  "day_high": 1012.35,
  "day_low": 998.4,
  "last_level": 1008.9195,
  "last_updated": 1749760800.0
}
```

On restart, this file is the authority for the **live** index: the divisor, the constituent list, each constituent's share count and its last price. The configuration only supplies a fresh start. So an `add`, `delist`, split, issuance or rebalance made through `pm-index-admin-cli` survives a restart without touching the config; if the persisted constituents differ from the configured ones, `pm-index` logs a warning and keeps the persisted set. If the `index_id` in the file does not match, `pm-index` exits with an error telling you to use `--reset`, which discards the file and returns to the configuration (including a new divisor). A crash can lose at most the price moves of the last 30 seconds; they are replaced by the next trade in each constituent. `day_open`, `day_high` and `day_low` are diagnostics only: the session's open/high/low starts afresh in a new process.

18.7.2 History file

The JSONL history file is a **structural/corporate-action audit log**, not a level-history store. Only events that change an index's composition or divisor are appended (one JSON object per line):

Record type	When written
`INIT`	First-ever startup (no prior state); records `base_value`, `divisor`, constituent list
`CORP_ACTION`	After every split, dividend, or shares-issuance adjustment
`ADD_CONSTITUENT`	After a new symbol is added to the index
`DELIST`	After a symbol is removed from the index
`REBALANCE`	After a batch share-outstanding update

Unlike the other four record types, `REBALANCE` has no `pm-index-admin-cli` subcommand — the only way to trigger one is the REST API's batch endpoint, `POST /admin/indexes/{id}/rebalance`. It applies several symbols' `new_shares_outstanding` in one atomic step (`index.rebalance` on `pm-index`'s PULL socket), unlike the single-symbol `shares` subcommand.

Every record carries `type`, `ts_ns` (epoch **nanoseconds**), `index_id` and `level`. Records are written as compact JSON (no spaces after `:` or `,`). Example history file extract:

```
{"type":"INIT","ts_ns":1749733100000000000,"index_id":"EDU100","base_value":1000.0,"divisor":7007100000.0,"constituents":["AAPL","MSF
{"type":"CORP_ACTION","ts_ns":1749847200000000000,"index_id":"EDU100","symbol":"AAPL","action":"SPLIT","detail":"2:1","old_divisor":7
```

History files are not deleted by `--reset`. They accumulate across sessions and provide a permanent structural audit trail.

Level and EOD history live in pm-stats, not here

Every `index.update` broadcast — including the throttled intraday ticks and the forced end-of-day publish — is recorded by pm-stats in two SQLite tables: `index_level_snapshots` (every tick) and `index_daily_stats` (daily OHLC rollup). Query them with `pm-stats-cli index-snapshots` and `pm-stats-cli index-daily`. This JSONL file intentionally does **not** duplicate

that data; it exists purely so operators can audit *why* an index's divisor changed (splits, dividends, constituent changes) without scanning a high-frequency time series.
Note that `index_daily_stats.close_level` reflects the *most recent* update for that date, not necessarily the final close, until the day ends or `close_session_state` reads `CLOSED` — see Getting the EOD index level for a date.

18.8 Getting Index Values

18.8.1 From a gateway terminal (`INDEX` command)

Connect through any `pm-alf-console` session (including `ADMIN` role) and run:

```
INDEX
```

Sample output:

```
[10:15:23.411] EDU100 1048.73 +6.63 +0.64% O=1042.10 H=1056.30 L=1040.05 CONTINUOUS
```

Note

`pm-index` must be running. If it is not, the gateway prints "No index data received yet. Is pm-index running?"

Structural/audit history queries

```
INDEX|HISTORY|INDEX=EDU100
```

`INDEX=<id>` is required unless the console has already received at least one `index.update` broadcast for an index in this session, in which case that index's ID is used as the default and `INDEX=` may be omitted. Without either, the console prints `INDEX|HISTORY requires INDEX=<id> or prior index.update.` and sends nothing.

Returns the last 30 days of structural/audit records (`INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST`, `REBALANCE`) — corporate actions and constituent changes, not level ticks. The result is printed as a table titled *Index structural history* with the columns Type, Time (local), Symbol, Detail and Level; when nothing matches it prints a "No structural index history records returned" hint instead.

```
INDEX|HISTORY|INDEX=EDU100|FROM=2026-06-01|TO=2026-06-12
```

Returns records within the given date range, oldest first. `FROM` and `TO` are dates (`YYYY-MM-DD`) read as **local dates**: `FROM` starts at local midnight, and `TO` includes the whole date, through the final nanosecond before the next local midnight. Thus `FROM=2026-06-12|TO=2026-06-12` includes records from 12 June. Exact timestamp bounds on the history request are inclusive. An unparseable date is silently ignored and the default (30 days back / now) is used.

Level/EOD history is a different tool

`INDEX|HISTORY` only returns structural/audit events. For index level or end-of-day time-series data, use `pm-stats-cli index-snapshots` or `pm-stats-cli index-daily` instead (see Statistics and Reporting).

Offline analysis with pm-index-cli

For richer filtering, CSV/JSON export, or scripting of structural/audit records, use `pm-index-cli` instead. It reads the JSONL history files directly without going through the gateway socket.

18.8.2 From the market-data gateway (CALF)

If `pm-md-gwy` is running and connected to `pm-index`, external clients can subscribe via the CALF `INDEX` channel:

```
HELLO|CLIENT=student01|PROTO=CALF1
SUB|CH=INDEX|SYM=EDU100
```

The gateway sends an initial `SNAP` and then live `IDX` messages:

```
SNAP|CH=INDEX|SYM=EDU100|SEQ=1|TS=2026-06-12T10:15:23.000Z|LEVEL=1048.73|SESSION=CONTINUOUS|OPEN=1042.1|CHG=+6.63|PCTCHG=+0.64|HIGH=10
IDX|CH=INDEX|SYM=EDU100|SEQ=2|TS=2026-06-12T10:15:24.411Z|LEVEL=1051.2|SESSION=CONTINUOUS|OPEN=1042.1|CHG=+9.10|PCTCHG=+0.87|HIGH=1056
```

The `SNAP` carries the same fields as the most recent `IDX`. `LEVEL`, `OPEN`, `HIGH` and `LOW` are the raw numbers (no fixed decimal places — `1042.1`, not `1042.10`); `CHG` and `PCTCHG` are rounded to two places. The field order on the wire is as shown, but clients

should parse by key.

IDX message fields

Field	Type	Description
CH	string	Always INDEX
SYM	string	Index ID (e.g. EDU100)
SEQ	int	Monotonic sequence number for gap detection
TS	string	UTC ISO-8601 timestamp
LEVEL	decimal	Current index level
CHG	decimal	Change from day open (signed); omitted before first open
PCTCHG	decimal	Percentage change from day open (signed); omitted before first open
OPEN	decimal	Day open level; omitted until the first level update after the day's open/high/low were last cleared (each OPENING_AUCTION / CONTINUOUS transition)
HIGH	decimal	Day high; omitted together with OPEN
LOW	decimal	Day low; omitted together with OPEN
AGGCAP	int	Current aggregate market cap
SESSION	string	Current session state

18.8.3 Using `pm-index-cli` for structural/audit records

`pm-index-cli` queries the structural/audit JSONL files directly from the command line — no running `pm-index` process is needed, except for its `health` command, which pings the running process. It does **not** have level or EOD subcommands; use `pm-stats-cli` for that (next section).

`--config`, `--data-dir`, `--format`, and `--no-header` are global options and must come **before** the subcommand (`events` / `indices`); putting them after the subcommand is a parse error since only the top-level parser defines them.

```
# Corporate actions and constituent changes (all time)
pm-index-cli --config engine_config.yaml events --index EDU100

# Filter to a specific structural event type
pm-index-cli --config engine_config.yaml events --index EDU100 --type CORP_ACTION

# Export to CSV for import into a spreadsheet (--format comes before the subcommand)
pm-index-cli --config engine_config.yaml --format csv events > edu100_events.csv

# JSON output for scripting
pm-index-cli --config engine_config.yaml --format json events --index EDU100 \
  | python3 -c "import json,sys; [print(r['ts'], r['type'], r['detail']) for r in json.load(sys.stdin)]"

# Date-range query
pm-index-cli --config engine_config.yaml events \
  --index EDU100 --from 2026-06-01 --to 2026-06-25

# List all configured indices
pm-index-cli --config engine_config.yaml indices

# --config is optional: without it the deployed (compiled) configuration is used.
# --data-dir is only a fallback for indices the configuration does not know
# (default data/indexes, inside the data directory); a configured history_file wins.
pm-index-cli --data-dir /custom/path/indexes events --index EDU100
```

See the pm-index-cli reference for all subcommands, column descriptions, and output-format options.

18.8.4 Querying level/EOD history with pm-stats-cli

For index level ticks or daily OHLC history, use pm-stats-cli instead, which reads from pm-stats' SQLite database:

```
# Daily open/high/low/close rollup
pm-stats-cli index-daily --index-id EDU100

# Raw level snapshots (every recorded tick) in a time window
pm-stats-cli index-snapshots --index-id EDU100 --from 2026-06-01 --to 2026-06-25

# List all index IDs that have recorded data
pm-stats-cli index-ids
```

See Statistics and Reporting for the full command reference.

18.8.5 Via the REST API

pm-api-gwy also exposes both the structural/audit log and the level/EOD time series over HTTP, under /api/v1/history:

Endpoint	Data	Backing store
GET /history/index-daily	Daily index OHLC rows	pm-stats SQLite
GET /history/index-snapshots	Intraday index level ticks	pm-stats SQLite
GET /history/indexes	Index IDs with recorded statistics	pm-stats SQLite
GET /history/index-events	Structural/audit records (INIT, CORP_ACTION, ADD_CONSTITUENT, DELIST, REBALANCE)	Live round-trip to pm-index, not pm-stats

```
curl -s -H "Authorization: Bearer key-trader-demo" \
  "http://127.0.0.1:8080/api/v1/history/index-daily?index_id=EDU100&date=2026-06-14"

curl -s -H "Authorization: Bearer key-trader-demo" \
  "http://127.0.0.1:8080/api/v1/history/index-events?index_id=EDU100&from=1750000000&to=1760000000"
```

See API Gateway — History endpoints for the full parameter reference, pagination rules, and the note that index-events is the one history endpoint that is not served from pm-stats and has no pagination cursor.

18.8.6 Reading the structural/audit file directly

The JSONL file is plain text — one JSON object per line. This can be useful for quick one-off inspections or when pm-index-cli is not available:

```
tail -20 data/indexes/EDU100_history.jsonl

# Corporate action records only
grep '"type":"CORP_ACTION"' data/indexes/EDU100_history.jsonl | python3 -m json.tool

# Structural records between two timestamps
python3 - <<'EOF'
import json
from_ts = 1749700000.0
to_ts   = 1749800000.0
with open("data/indexes/EDU100_history.jsonl") as f:
    for line in f:
        rec = json.loads(line)
        ts = rec["ts_ns"] / 1e9          # epoch nanoseconds -> seconds
        if from_ts <= ts <= to_ts:
            print(f"{ts:.0f}  {rec['type']}  {rec.get('symbol', '')}")
EOF
```

18.9 Example Configurations

The `pm-config-gen` commands below seed reference prices from a market-maker midpoint, as explained in the note above; `pm-index` needs them to compute each index's initial divisor.

18.9.1 Single index, three constituents

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 TRADER02 OPS01:ADMIN MM01:MARKET_MAKER \
  --outstanding-shares AAPL:15000000000 \
  --outstanding-shares MSFT:7400000000 \
  --outstanding-shares TSLA:3200000000 \
  --seed-mm-mid-range 100:400 \
  --seed-last-prices-from-mm \
  --no-mm-seed-quotes \
  --sessions-enabled \
  --index EDU100:"EduMatcher broad index" \
  --index-constituents EDU100:AAPL,MSFT,TSLA \
  --output engine_config.yaml
```

18.9.2 Two indices with different base values and publish rates

```
pm-config-gen \
  --symbols AAPL MSFT TSLA \
  --participants TRADER01 OPS01:ADMIN MM01:MARKET_MAKER \
  --outstanding-shares AAPL:15000000000 \
  --outstanding-shares MSFT:7400000000 \
  --outstanding-shares TSLA:3200000000 \
  --seed-mm-mid-range 100:400 \
  --seed-last-prices-from-mm \
  --no-mm-seed-quotes \
  --index BROAD:"Broad market" \
  --index-constituents BROAD:AAPL,MSFT,TSLA \
  --index-base-value BROAD:1000.0 \
  --index TECH2:"Technology pair" \
  --index-constituents TECH2:AAPL,MSFT \
  --index-base-value TECH2:500.0 \
  --index-interval TECH2:2.0 \
  --output engine_config.yaml
```

18.9.3 Full manual YAML for two indices

```
indices:
  - id: BROAD
    description: "Broad market"
    base_value: 1000.0
    publish_interval_sec: 1.0
    history_file: data/indexes/BROAD_history.jsonl
    state_file: data/indexes/BROAD_state.json
    constituents:
      - AAPL
      - MSFT
      - TSLA

  - id: TECH2
    description: "Technology pair"
    base_value: 500.0
    publish_interval_sec: 2.0
    history_file: data/indexes/TECH2_history.jsonl
    state_file: data/indexes/TECH2_state.json
    constituents:
      - AAPL
      - MSFT
```

18.10 Operational Checklist

1. Ensure every constituent has `outstanding_shares` set in `symbols:`.
2. Create the `data/indexes/` directory before first run (or let `pm-index` create it automatically — it creates parent directories as needed).
3. Start `pm-engine` before `pm-index` during normal operations; `pm-index` will reconnect if the order is reversed but may miss the first few trades.
4. Use `--reset` only when deliberately changing the constituent list or `base_value`. This discards the divisor history for all indices.
5. Apply corporate actions during `PRE_OPEN` to avoid mid-session index jumps, using `pm-index-admin-cli`.
6. Check `data/indexes/<ID>_state.json` after a crash to confirm the divisor was persisted. If the file is missing or truncated, use `--reset`.

CHAPTER 19

Index Administration (`pm-index-admin-cli`)

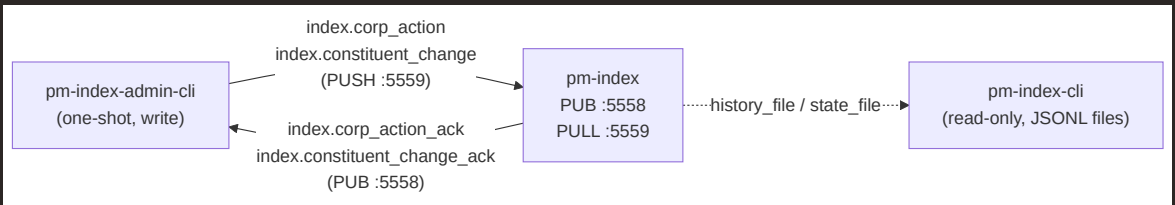
Learning objectives

After reading this page you will understand:

- Why corporate actions and constituent changes need a dedicated CLI instead of a hand-written script
- The full set of subcommands: `split`, `dividend`, `shares`, `add`, `delist`, `history`
- How `--dry-run`, confirmation prompts, and `--yes` work together to make mutations safe
- How the `shares` subcommand models buy-backs using `--delta`
- Why this tool talks only to `pm-index`, not to `pm-engine`, and what that means for authentication

19.1 What this tool is

`pm-index-admin-cli` is a one-shot command-line tool for applying corporate actions (splits, cash dividends, share issuances and buy-backs) and constituent changes (add / delist) to a running `pm-index` process. It replaces the "write a short script" workaround described in Market Index — Applying corporate actions: every action that script could perform is now a documented subcommand with `--help` text, client-side validation, a confirmation prompt, and a `--dry-run` preview mode.



Under the hood, `pm-index-admin-cli` is built directly on the same `ExchangeCommandClient` class `pm-admin-cli` uses for engine commands — specifically its `index_corp_action()`, `index_delist()`, `index_add_constituent()`, and `index_history()` methods. No engine-side or `pm-index`-side code changes were needed to add this tool; it is a pure client addition.

19.2 Why not `pm-index-cli`?

`pm-index-cli` already exists as a **read-only** query tool over `pm-index`'s history/state JSONL files — it never touches the live ZMQ bus. Corporate actions are a live, mutating operation against a running process, which is a different responsibility, so it lives in a separate binary rather than a write mode bolted onto `pm-index-cli`. This mirrors the existing split between `pm-admin`/`pm-admin-cli` (mutating engine commands) and `pm-stats-cli`/`pm-clearing-cli`/`pm-audit-cli`/`pm-index-cli` (read-only query tools): a command that can change exchange state gets its own explicitly-named entry point so the blast radius of a mistyped command is obvious from the binary name alone.

19.3 Why there is no connect() / authentication step

Unlike pm-admin-cli, which authenticates against the engine's PUSH/PULL socket (system.gateway_connect → system.gateway_auth before running a command, pm-index-admin-cli sends directly — there is no handshake.

pm-index's PULL socket (port 5559) has no authentication of any kind: no ZAP/CURVE, no gateway allowlist, and no role check. It only verifies that the gateway_id field on an inbound message is non-empty before processing it — the field is used purely as an ack-routing key (which SUB topic to reply on), never as an identity or permission check. This is documented in Exchange Commands — PUSH socket has no authentication for the engine's own socket, and the index socket has even less: it doesn't even have the engine's payload-level role: ADMIN check.

Practically, this means --id on pm-index-admin-cli is a free-text label, not a credential — any non-empty value is accepted by pm-index, though using a real gateway ID makes acks easier to correlate in logs. Any process that can reach tcp://127.0.0.1:5559 (localhost-only by default) can already apply corporate actions, whether through this CLI or a hand-written script; this tool does not change that exposure. The confirmation-prompt-by-default behaviour described below is a safeguard against accidental or scripted mistakes, not against malicious use — appropriate for a learning system running on localhost.

19.4 Starting point

```
pm-index-admin-cli --id OPS01 COMMAND [options]
```

pm-index must already be running and reachable (default tcp://127.0.0.1:5559 for commands, tcp://127.0.0.1:5558 for acks). pm-engine does not need to be running for this tool specifically, though in practice you will normally have both up since pm-index itself depends on pm-engine's trade feed to do anything useful.

Global options:

Flag	Default	Description
--id	*(required)*	Ack-routing label. Not authenticated — see above.
--push	tcp://127.0.0.1:5559	pm-index PULL socket address
--sub	tcp://127.0.0.1:5558	pm-index PUB socket address
--timeout	3000	Ack timeout in milliseconds
--dry-run	off	Validate and print the outbound payload; do not send it
-y, --yes	off	Skip the confirmation prompt
--format	table	Output format: table (human-readable) or json

Exit codes:

Code	Meaning
0	Accepted by pm-index (or --dry-run validated cleanly)
1	Rejected by pm-index, ack timeout, or the confirmation prompt was declined
2	Usage error — bad flags or a client-side validation failure

19.5 Unknown index ID fails fast

If --index does not match any configured index, pm-index replies with a generic error on index.error.<gateway_id> rather than the normal ack topic for the command (index.corp_action_ack.<gateway_id> / index.constituent_change_ack.<gateway_id>

/ index.history.<gateway_id>). ExchangeCommandClient recognizes this side-channel reply for every index method and raises immediately instead of waiting out the full --timeout for an ack that will never arrive:

```
$ pm-index-admin-cli --id OPS01 --yes split --index NOPE --sym AAPL --ratio 2:1
REJECTED  Unknown index_id 'NOPE'
```

--format json prints the raw error payload instead of the summary line, consistent with every other rejection:

```
$ pm-index-admin-cli --id OPS01 --format json --yes split --index NOPE --sym AAPL --ratio 2:1
{
  "accepted": false,
  "reason": "Unknown index_id 'NOPE'",
  "ts_ns": 1784500215390000000
}
```

Under the hood, this is implemented as an error_prefix parameter on ExchangeCommandClient._recv(): each index method passes index.error.<gateway_id> alongside its normal expected ack prefix, and a matching reply raises CommandError (a new exception distinct from CommandTimeoutError) with the error payload attached. pm-index-admin-cli catches CommandError in main() and reports it exactly like a normal accepted=False rejection. This is a client-library-level fix — it applies to any caller of index_corp_action(), index_delist(), index_add_constituent(), or index_history(), not just this CLI.

A genuine unreachable-pm-index timeout (no reply of any kind) is unaffected and still produces the original timeout message:

```
$ pm-index-admin-cli --id OPS01 --push tcp://127.0.0.1:59999 --yes split --index TECH10 --sym AAPL --ratio 4:1
Timed out waiting for pm-index: No ack with prefix 'index.corp_action_ack.OPS01' within 3000 ms
Is pm-index running and reachable at tcp://127.0.0.1:59999?
```

19.6 Confirmation prompts and --dry-run

Every mutating subcommand (split, dividend, shares, add, delist) prints a description of the action and asks Continue? [y/N] before sending anything. -y/--yes skips the prompt for scripted use. If stdin is not a TTY and --yes was not passed, the tool prints the prompt followed by REJECTED stdin is not a TTY; pass --yes/-y to confirm non-interactively. to stderr and exits 1 rather than hanging — this matters for CI/cron use.

--dry-run prints the exact outbound topic and JSON payload and exits 0 without sending the command or prompting (the payload carries no command_id; that is added only when a command is really sent). Use it to check a --ratio parse or a --delta-resolved share count before committing — note that shares --delta --dry-run still queries pm-index for the current share count, so pm-index must be running for it:

```
$ pm-index-admin-cli --id OPS01 split --index TECH10 --sym AAPL --ratio 4:1 --dry-run
DRY RUN — would send topic='index.corp_action'
{
  "action": "SPLIT",
  "gateway_id": "OPS01",
  "index_id": "TECH10",
  "ratio_denominator": 1,
  "ratio_numerator": 4,
  "symbol": "AAPL"
}
```

history is read-only and never prompts, regardless of --yes.

19.7 Subcommand reference

19.7.1 split — Apply a stock split

```
pm-index-admin-cli --id OPS01 split --index TECH10 --sym AAPL --ratio 4:1
```

Flag	Required	Description
--index	Yes	Index ID
--sym	Yes	Constituent symbol

Flag	Required	Description
`--ratio`	Yes	`N:M` split ratio, e.g. `4:1` for a 4-for-1 split, `1:10` for a 1-for-10 reverse split. Both components must be positive integers — validated locally before anything is sent.

```
$ pm-index-admin-cli --id OPS01 split --index TECH10 --sym AAPL --ratio 4:1
This will apply a SPLIT (4:1) to AAPL in index TECH10. Continue? [y/N] y
SPLIT OK   TECH10  AAPL  ratio=4:1  new_level=8452.17  new_divisor=118.3352
```

19.7.2 `dividend` — Apply a cash dividend

```
pm-index-admin-cli --id OPS01 dividend --index TECH10 --sym MSFT --amount 0.75
```

Flag	Required	Description
`--index`	Yes	Index ID
`--sym`	Yes	Constituent symbol
`--amount`	Yes	Dividend per share, in price units. Must be `> 0` (checked locally; `pm-index` separately rejects if the resulting price would be non-positive).

```
$ pm-index-admin-cli --id OPS01 --yes dividend --index TECH10 --sym MSFT --amount 500
REJECTED  Dividend would make price non-positive
```

19.7.3 `shares` — Set shares outstanding (issuance or buy-back)

```
pm-index-admin-cli --id OPS01 shares --index TECH10 --sym AAPL --new-shares 15200000000
```

Flag	Required	Description
`--index`	Yes	Index ID
`--sym`	Yes	Constituent symbol
`--new-shares`	One of these two	New absolute total shares outstanding
`--delta`	One of these two	Signed change from the last known share count (negative for a buy-back, positive for an issuance)

`pm-index` has no dedicated `BUYBACK` action — only `SHARES_ISSUANCE`, which always sets an absolute share count regardless of direction (see Market Index — Supported corporate actions). `--delta` is a CLI-side convenience: it queries the index's whole history and takes the most recent share count recorded for that symbol — either an `ADD_CONSTITUENT` record's `shares_outstanding` or a `SHARES_ISSUANCE` corporate action — applies the signed delta, and shows the computed absolute value before sending. So it works right after `add`. Two caveats: a `SPLIT` recorded since then is **not** reflected (the share count it starts from is the pre-split one), and `ADD_CONSTITUENT` records written by older versions carry no `shares_outstanding`. If no share count can be found, the command prints `REJECTED  no prior shares_outstanding found for SYM in index IDX — pass --new-shares instead.` to stderr and exits `1`.

```
$ pm-index-admin-cli --id OPS01 shares --index TECH10 --sym AAPL --delta -800000000
Last known shares_outstanding for AAPL: 16,000,000,000
This will set shares_outstanding to 15,200,000,000 (delta -800,000,000, a buy-back). Continue? [y/N] y
SHARES_ISSUANCE OK   TECH10  AAPL  new_shares_outstanding=15,200,000,000  new_level=8401.09  new_divisor=118.1187
```

19.7.4 `add` — Add a constituent

```
pm-index-admin-cli --id OPS01 add --index TECH10 --sym NVDA --shares 2470000000 --price 118.50
```

Flag	Required	Description
`--index`	Yes	Index ID
`--sym`	Yes	Symbol to add
`--shares`	Yes	Initial shares outstanding. Must be `> 0`.
`--price`	Yes	Initial reference price. Must be `> 0`.

The symbol must already be tradeable

`pm-index` does not check that `--sym` is a configured, tradeable symbol in `engine_config.yaml` — it only tracks it as an index constituent. If the symbol doesn't exist elsewhere in the config, it will be added to the index but never receive live price updates. The confirmation prompt states this requirement, but `pm-index-admin-cli` has no `--config` flag and cannot verify it for you. A change made this way lives in `pm-index`'s state file and survives restarts, but not `--reset` — see Market Index — State file for keeping the config in step.

19.7.5 `delist` — Remove a constituent

```
pm-index-admin-cli --id OPS01 delist --index TECH10 --sym XYZ
```

Flag	Required	Description
`--index`	Yes	Index ID
`--sym`	Yes	Symbol to delist

Delisting the last remaining constituent of an index is rejected by `pm-index` (it would zero out the index's aggregate market cap):

```
$ pm-index-admin-cli --id OPS01 --yes delist --index TECH10 --sym AAPL
REJECTED  Delisting last constituent would make aggregate cap zero
```

`delist` has no undo. Re-adding a delisted symbol requires supplying `--shares`/`--price` again via `add` — the confirmation prompt calls this out explicitly.

19.7.6 `history` — Show recent structural/corp-action history

```
pm-index-admin-cli --id OPS01 history --index TECH10 --limit 20
```

Flag	Required	Default	Description
`--index`	Yes	—	Index ID
`--from`	No	`--to` minus 24 h	Start of range, as a Unix timestamp (seconds)
`--to`	No	now	End of range, as a Unix timestamp (seconds)
`--types`	No	all	Comma-separated filter: `INIT`, `CORP_ACTION`, `DELIST`, `ADD_CONSTITUENT`, `REBALANCE`
`--limit`	No	`50`	Maximum rows shown — the oldest `N` of the matching, oldest-first list

This is the same underlying `index.history_request` used by `INDEX|HISTORY` in `pm-alf-console` (see Market Index — Structural/audit history queries), wrapped here so you can confirm an action landed without switching tools. Unlike `pm-index-cli events`, which reads the JSONL file directly from disk and needs no running process, `history` here round-trips through the live `pm-index` process.

```
$ pm-index-admin-cli --id OPS01 history --index TECH10 --from 1784000000 --limit 3
  1784106000000000000  ADD_CONSTITUENT NVDA     shares=2,470,000,000 price=118.5
  1784365200000000000  CORP_ACTION     AAPL     SHARES_ISSUANCE shares=16000000000 -> level=8390.02
  1784451302000000000  CORP_ACTION     AAPL     SPLIT 4:1 -> level=8452.17
```

The first column is the record's raw `ts_ns` (epoch nanoseconds), and records are listed oldest first. An empty result prints `No history records found.` `--format json` prints the raw record list instead:

```
pm-index-admin-cli --id OPS01 --format json history --index TECH10 --limit 3
```

19.8 Output formats

Default (`--format table`) output is a single-line `<COMMAND> OK  <fields>` summary on success, or a `REJECTED  <reason>` line on failure — the same convention `pm-admin-cli` uses. `new_level` and `new_divisor` are printed exactly as `pm-index` computed them (unrounded floats). `--format json` prints the raw ack payload (or, for `history`, a JSON array of records) instead, for scripting:

```
$ pm-index-admin-cli --id OPS01 --format json --yes split --index TECH10 --sym AAPL --ratio 4:1
{
  "accepted": true,
  "command_id": "cmd-3f9a1c0b7d2e4a65b81e0c94d7a2f356",
  "divisor": 118.3352,
  "index_id": "TECH10",
  "level": 8452.17,
  "old_divisor": 118.3352,
  "reason": "",
  "ts_ns": 1784451302000000000
}
```

19.9 Typical operator workflow

A full pre-market corporate-action sequence, applied before `PRE_OPEN` ends (see Market Index — Apply corporate actions before the market opens):

```
# Preview first
pm-index-admin-cli --id OPS01 split --index TECH10 --sym AAPL --ratio 4:1 --dry-run

# Apply the split
pm-index-admin-cli --id OPS01 split --index TECH10 --sym AAPL --ratio 4:1

# Apply a cash dividend to a different constituent
pm-index-admin-cli --id OPS01 dividend --index TECH10 --sym MSFT --amount 0.75

# Retire a delisted constituent
pm-index-admin-cli --id OPS01 delist --index TECH10 --sym XYZ

# Add its replacement
pm-index-admin-cli --id OPS01 add --index TECH10 --sym NVDA --shares 2470000000 --price 118.50

# Confirm everything landed
pm-index-admin-cli --id OPS01 history --index TECH10 --limit 10
```

For scripted/CI use (a nightly batch job applying pre-approved actions from another system), pass `--yes` on every mutating call and check the process exit code:

```
#!/usr/bin/env bash
set -e
pm-index-admin-cli --id BATCH01 --yes shares --index TECH10 --sym AAPL --delta -800000000
pm-index-admin-cli --id BATCH01 --yes dividend --index TECH10 --sym MSFT --amount 0.75
```

19.10 Further Reading

- Market Index — `pm-index` process, corporate-action semantics, divisor math, and the state/history file formats this tool writes to
- The Admin Console and Exchange Commands — `pm-admin-cli`, `ExchangeCommandClient`, and the sibling `pm-index-cli` read-only query tool
- Processes, Environment and Ports — full process/command reference table

CHAPTER 20

The Market Model (`pm-market-sim`)

Learning objectives

After reading this page you will be able to:

- Explain what the *true value* of a symbol is and who gets to see it
- Write or generate a `market_sim.yaml` and deploy it with the engine configuration
- Run `pm-market-sim`, check on it, and anchor a market maker to it
- Tell how a compressed trading day relates to a real one
- Publish news with `pm-news`, by hand or from a scenario

Prerequisites: Session Scheduling, Running the AI-Trader Swarm.

20.1 What it does

Prices on the exchange are whatever the participants agree on. `pm-market-sim` adds what a real market has behind its prices: a *true value* for every symbol, moving the way company values move — together with the market, with the sector, on its own, and now and then in a jump. Bots that trade on value pull prices toward it; noise and momentum traders push them away; the gap is what students can learn to read.

The values are for the bots and the instructor only. They travel on `pm-market-sim`'s own sockets (loopback by default), never on the engine bus that students' clients reach, and the API gateway serves them to ADMIN keys only.

Who	Uses the value for
AI traders with the `value` strategy (presets `value-investor`, `institutional`)	Buy below their view of the value, sell above it
`pm-mm-bot --anchor-sim W`	Move its quoted mid a share `W` of the way toward the value
ADMIN API keys	`GET /api/v1/admin/sim`, WebSocket `/api/v1/ws/admin/sim`
`pm-market-sim --status --id OPS01`	Where the market is furthest from the value

The model also publishes *news*: headlines that move the values (see News). Headlines are public — every API key, `pm-ticker` and both GUIs show them; what a headline does to the value is not.

20.2 The model

For each symbol, one step of trading time Δ (in days) moves the log value by

$$(\mu - \tfrac{1}{2}\sigma^2 u)\Delta + \sigma\sqrt{u\Delta} \cdot (\beta_m \cdot Z_{market} + \beta_s \cdot Z_{sector} + \sqrt{1 - \beta_m^2 - \beta_s^2} \cdot Z_{symbol}) + jumps$$

- and are annual (252 trading days); `u` is an intraday multiplier, high after the open and before the close, low at midday, averaging 1.
- `β_m` and `β_s` load each symbol on a market factor and its sector's factor, so symbols of one sector move together: two of them correlate by about `β_m² + β_s²`.

- Jumps arrive at `jump_rate` per day with normally distributed log size. No news announces them. Each one is logged (`value jump AAPL +4.10% to 83.97`) and recorded in `market_sim_news.db`'s `value_jumps` table, so you can see afterwards how the market followed it.
- When the exchange reopens after a close, one extra move with `overnight_fraction` of a day's variance is applied.

The model steps only during continuous trading. A day's variance accrues over the continuous phase *as the engine reports it* — from its start to the scheduler's countdown to the close — so a day compressed with `pm-scheduler --speed` moves as much as a real one. Without a countdown (no scheduler, or `pm-scheduler --now`) a 6.5-hour day is assumed.

Every source of randomness is its own stream seeded from the model seed, so a run is reproducible and adding a symbol does not change the others' paths.

20.3 `market_sim.yaml`

```
version: 1
seed: 150
step_sec: 1.0              # seconds between steps
overnight_fraction: 0.2
defaults: {vol: 0.30, drift: 0.05, jump_rate: 0.05, jump_mean: 0.0,
           jump_std: 0.04, beta_market: 0.5, beta_sector: 0.4}
sectors:
  TECH: {vol: 0.35, beta_market: 0.6, beta_sector: 0.45}
symbols:
  AAPL: {sector: TECH, vol: 0.28}
  NVDA: {sector: TECH, vol: 0.45, initial: 120.0}
```

A symbol's parameters are the defaults, overridden by its sector's, overridden by its own entry. A symbol of the engine configuration that the file does not list gets the defaults (sector `OTHER`); a symbol the engine does not list is an error. A symbol starts at its `initial`, or else at the engine's previous close, or else at the configuration's last price.

`pm-market-sim --init` writes a file for the deployed symbols, each in its sector (the usual industry sectors for the bundled tickers) with that sector's betas and a volatility spread around the sector's. The `s150-*` and `s300-load` examples ship one.

Keep the file next to the `engine_config.yaml` you deploy: `pm-config-deploy` checks it against the engine's symbols and installs it beside the compiled configuration; deploying a configuration without one removes the old one.

```
pm-market-sim --init --seed 7 > my-course/market_sim.yaml
pm-config-deploy my-course/engine_config.yaml
```

20.4 Running it

```
pm-market-sim -v
```

It follows the engine's session (asking for it at startup and after an engine restart), publishes `sim.value` once per `step_sec` during continuous trading and a `sim.state` heartbeat every second (`RUNNING` or `PAUSED`). It saves its values to `<DATA_DIR>/market_sim_state.js` every ten seconds and on exit, so a restart continues from where the model stood.

Socket	Port	Carries
PUB	`5553` (`EDUMATCHER_SIM_PUB_PORT`)	`sim.value`, `sim.state`, `news.event`, `sim.command_ack.<ID>`
PULL	`5554` (`EDUMATCHER_SIM_PULL_PORT`)	`sim.command` from ADMIN participants

Both bind to `EDUMATCHER_SIM_BIND_HOST` (default `127.0.0.1`).

`pm-market-sim --status --id OPS01` asks the running process where it stands and lists the symbols whose market mid is furthest from the value:

```
pm-market-sim: RUNNING seq=4120 symbols=150
symbol        value        mid  mid/value
TSLA         212.40     219.95     +3.55%
...
```

20.5 Anchoring a market maker

`pm-mm-bot --anchor-sim 0.3` moves the bot's mid 30% of the remaining way to the true value at every model step (once a second), and from the first value on the book no longer sets its mid: its quotes converge on the value within a few seconds. With `0` (the default) the bot ignores the model and quotes around the book. A market maker anchored this way is a strong pull toward the value; leave it off to let the value traders do the work.

20.6 News

A headline has a *scope* — one or more symbols, one or more sectors, or the whole market — a *kind* (`EARNINGS`, `GUIDANCE`, `MNA`, `REGULATORY`, `PRODUCT`, `LEGAL`, `MANAGEMENT`, `MACRO`, `OPERATIONS`, `RESEARCH`, `CONTRACT`, `ESG`), a public *sentiment* from −1 (bad) to 1 (good), and a secret *impact*: the log move of the true value (0.10 is about +10.5%). A sector headline moves each of its symbols by `β_s` times the impact, a market headline every symbol by `β_m` times it.

A headline can start as a **rumour**. A rumour carries a *credibility* (0 to 1) and moves nothing; it is later **confirmed** — a second headline, and the value moves then — or **retracted**, and nothing ever moves. Prices may well move on a rumour anyway: that is traders betting on it.

Who	Reads the news through
AI traders with the `news` strategy (preset `news-trader`)	Trade the headline within seconds, in its direction, harder the stronger the sentiment
AI traders with the `value` strategy	Shift their view of the value on a rumour, by sentiment × credibility
Every API key	`GET /api/v1/news`, and the `news` channel of the market-data WebSocket
`pm-ticker`, the trading GUI, TapeDeck	A news line, a News screen, a News view

20.6.1 Generated news

Without anyone doing anything, `pm-market-sim` publishes news at random during continuous trading: on average 6 symbol, 0.5 sector and 0.3 market headlines per trading day across the whole exchange. 30% start as rumours, which resolve on their own 0.02 to 0.15 trading days later — confirmed 60% of the time; a rumour that will be confirmed tends to carry a higher credibility. About a third of symbol headlines are *events*: fires, breakthroughs, trial results, big contracts, protests and recalls. They go one way, move the value 3–10 %, some happen only in their own industry, and they come far more often than in real life so that a class sees a few each day (see News and Rumours). Tune it in `market_sim.yaml`; a rate of 0 turns that scope off:

```
news:
  rate_symbol: 6.0
  rate_sector: 0.5
  rate_market: 0.3
  rumour_share: 0.3
  rumour_confirm: 0.6
  rumour_delay_min: 0.02   # trading days
  rumour_delay_max: 0.15
```

20.6.2 Publishing news yourself: `pm-news`

`pm-news` sends commands to the running `pm-market-sim` as an ADMIN participant (`--id`) and prints what happened:

```
pm-news --id OPS01 inject --symbol AAPL --kind EARNINGS --sentiment 0.8 --impact 0.08
pm-news --id OPS01 inject --sector ENERGY --kind REGULATORY --sentiment -0.7 \
    --impact -0.06 --rumour --credibility 0.6
N7
pm-news --id OPS01 confirm N7      # or: retract N7
pm-news list                       # print headlines as they are published
pm-news past --limit 20            # recent headlines, newest first
pm-news past --date 2026-10-14     # one session date
```

`inject` prints the new headline's id. `--headline` sets the text; without it one is written from the kind and the sentiment. A rumour you inject waits for your `confirm` or `retract` — it never resolves on its own. Commands from a participant that is not ADMIN are refused.

20.6.3 Scenarios

`pm-news play scenario.yaml` runs a timed script. Each step fires `after_sec` seconds into a session phase (or into the play when `phase` is left out); a phase already under way when the play starts counts from the start of the play. A step either injects a headline (same fields as the `inject` flags) or confirms or retracts an earlier step by its `name`:

```
version: 1
steps:
  - at: {phase: CONTINUOUS, after_sec: 60}
    name: bid
    inject: {symbol: JPM, kind: MNA, sentiment: 0.7, impact: 0.12, rumour: true,
             credibility: 0.5, headline: "JPM said to be in talks over a takeover bid"}
  - at: {phase: CONTINUOUS, after_sec: 240}
    confirm: bid
```

Four classroom scenarios ship in `docs/examples/news/` and fit every `s150-*` example and `s300-load`: `earnings-surprise.yaml`, `rumour-confirmed.yaml`, `rumour-retracted.yaml` and `chip-race.yaml`. The AI Traders & News training chapter uses them.

CHAPTER 21

Closing the Day: Settlement, Statements and Housekeeping

Learning objectives

After reading this page you will understand:

- What *settlement* is, in plain words, and why an exchange does it at the end of every trading day
- The money words used here — position, closing price, profit and loss, balance — each explained from scratch with numbers
- What `pm-eod` does after the market closes, step by step, and what it deliberately does *not* do
- How to set it up: the roster file, the scheduler hook, and running it without a scheduler
- How to read the statements and files it writes, and how to send them to your traders
- What the daily housekeeping (audit archive, log compression, backups) does, and how to restore a backup
- How to fix a day that went wrong, and what every warning and error message means

You do not need any finance background. Every financial word is explained the first time it appears, and again in the mini-glossary below. If you can follow a shopping receipt, you can follow this page.

Prerequisites: you can start the exchange and run a trading day (Running the Exchange), and you have read how sessions work (Sessions and Scheduling). The clearing process `pm-clearing` must be running during the day — it is part of the `default` profile.

21.1 The big picture

At the end of a trading day two jobs are waiting for the operator.

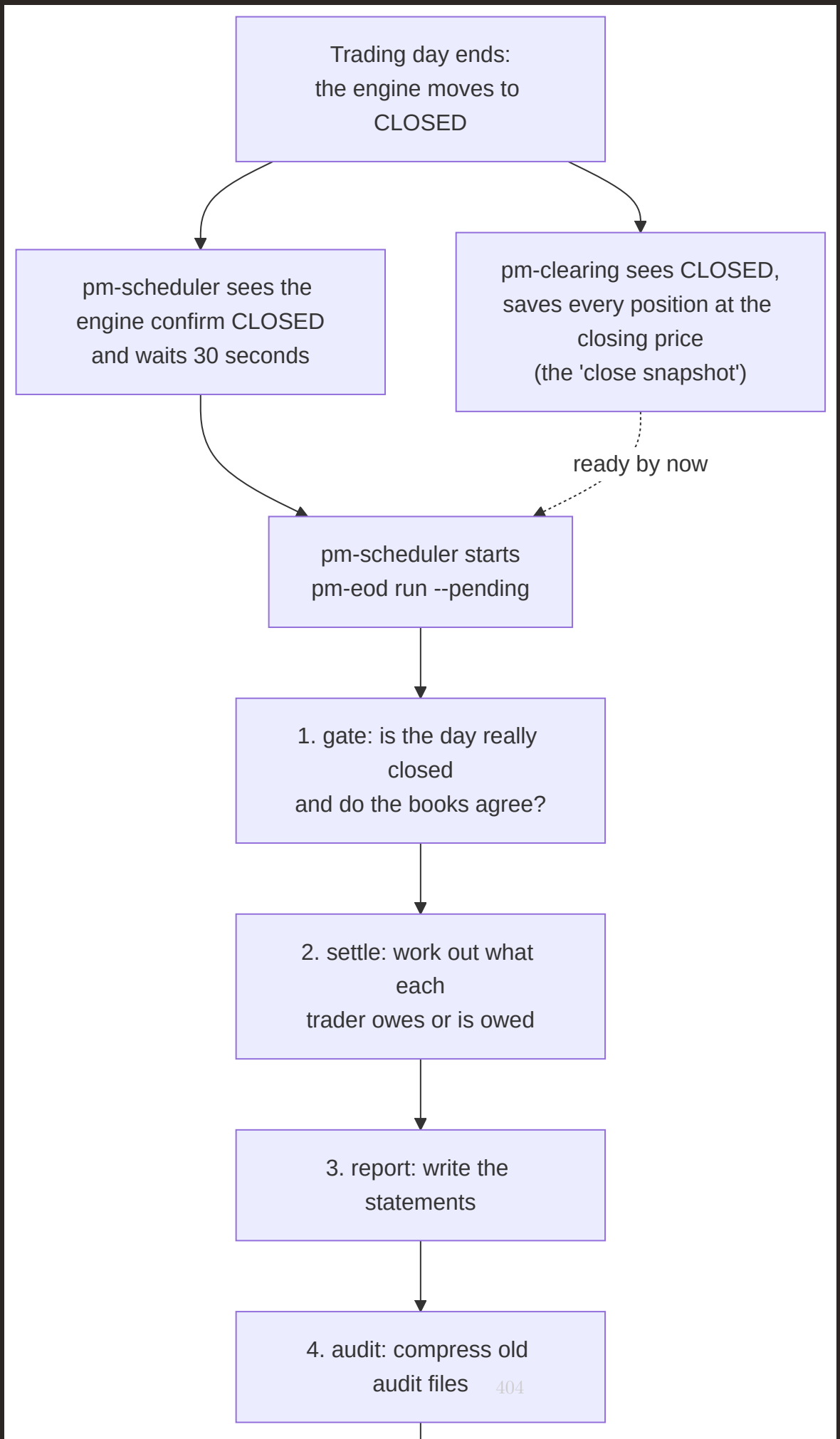
1. **Money.** During the day traders bought and sold shares. Some made a profit, some made a loss. Somebody has to add it all up, work out *who should pay how much and who should receive how much*, and tell everyone. That job is called **settlement**.
2. **Housekeeping.** The exchange has produced a lot of files — an audit trail, logs, databases. Old files should be compressed to save disk space, and important databases should be copied somewhere safe in case a disk fails. That is routine maintenance.

`pm-eod` ("end of day") does both, automatically, right after the market closes. It is a **one-shot command**: it starts, does its work in a few seconds, prints what it did and exits. It is not a long-running process.

The money is fictional

Everything in this chapter uses EduMatcher's **fictional currency**. No real money moves anywhere. `pm-eod` produces *numbers and files*: "Bob owes the exchange 20.00". What those numbers mean in your class — points, pretend dollars, a prize for the top of the leaderboard — is up to you. If you ever do attach real value to them, that is entirely outside EduMatcher; read What `pm-eod` does not do.

21.1.1 What happens, in order



Step	Plain-words purpose	Can it fail the day?
gate	Check it is safe to settle: the day is closed, the closing prices were saved, and the books add up	**Yes** — nothing is written if it fails
settle	Calculate each trader's amount for the day and add it to the running balance	**Yes** — nothing is written if it fails
report	Write one statement per trader and one summary for the instructor	**Yes** — but the settlement is kept and the report is retried next time
audit	Safety net: compress any older audit-trail days that were missed	No — a warning only
logs	Compress and tidy log files that nobody is writing to any more	No — a warning only
backup	Make a verified, compressed copy of the important databases	No — a warning only

The rule behind the last three rows: *housekeeping trouble must never stop traders from getting their statements.* If the backup disk is full, you get a warning, and the statements are still written.

21.1.2 What `pm-eod` does not do

- It **does not send email.** It writes files, including a ready-made table of names, email addresses and amounts, and *you* send them (see Sending the statements).
- It **does not move money**, real or pretend. It records numbers.
- It **does not stop anyone trading.** A trader whose balance is deeply negative can keep trading tomorrow. The balance is information, shown on the statement and the leaderboard.
- It **does not charge fees.** Settlement is purely the profit or loss from trading.
- It **does not touch the engine's own end-of-day files.** The engine keeps its own archive in `<data>/eod/`. `pm-eod` writes to a different place, `<data>/settlement/`, on purpose. (See Persistence.)

21.2 The ideas, from scratch

This section builds the vocabulary one word at a time. Each idea uses the same tiny example, and the next section runs it through several days. Skip it only if you already know what *mark-to-market* means.

21.2.1 Shares, symbols and trades

A **share** is a small slice of ownership in a company. Each company on the exchange has a short code, its **symbol** — in our examples `AAPL`.

A **trade** happens when one person agrees to buy and another agrees to sell, at the same **price**, for some **quantity** (how many shares).

> **Example.** Alice buys 100 `AAPL` from Bob at a price of **10.00** each. Alice is the **buyer**, Bob is the **seller**. The trade is worth $100 \times 10.00 = \mathbf{1000.00}$ in total (the **notional value**).

In EduMatcher every trader connects through a **gateway**, and the gateway has an identifier like `GW_ALICE`. The exchange never needs to know the trader's real name — the system sees gateway identifiers. The roster file is how *you* connect the two.

21.2.2 Positions: long and short

Your **position** in a symbol is how many shares you currently hold.

- If you own shares, you are **long**. After her purchase Alice is long 100.
- You can also sell shares you do not own (in the real world by borrowing them and promising to buy them back later; in EduMatcher the trader's position simply goes negative). Then you are **short**. After the sale Bob is short 100, shown as **−100**.

- If you hold nothing you are **flat** (position 0).

Someone who is long makes money when the price **goes up**. Someone who is short makes money when the price **goes down** (they sold high and hope to buy back low). That is the single most important fact for the rest of this page.

21.2.3 Average cost

If you buy in several steps at different prices, your **average cost** is the average price you paid, weighted by quantity. Buy 100 at 10.00 and then 50 at 10.20, and your average cost is $(100 \times 10.00 + 50 \times 10.20) \div 150 = \mathbf{10.0667}$. The accounting term is **VWAP** (volume-weighted average price). Average cost is what your profit is measured against. For the full mechanics, including what happens when you flip from long to short, see Positions and P&L.

21.2.4 The closing price and "marking to market"

What is your 100 shares worth *right now*? You cannot know until you sell them, but you can estimate with the latest price the market agreed on. At the end of the day that is the **closing price**: in EduMatcher, the price of the **last trade of the day** in that symbol.

Valuing what you hold at the latest market price, rather than at what you paid, is called **marking to market**, and the price used is the **mark** (or *mark price*). It is the same idea as checking your car's current resale value rather than what you paid for it.

21.2.5 Unrealized and realized profit and loss

P&L means *profit and loss*: the money made or lost.

- **Unrealized P&L** is the profit or loss *on paper*, on a position you still hold. Alice paid 10.00 and the closing price is 10.50, so her 100 shares have gained $100 \times (10.50 - 10.00) = \mathbf{+50.00}$ on paper. She has not sold, so it is not locked in; tomorrow the price may fall and the gain shrinks or turns into a loss.
- **Realized P&L** is profit or loss that is *locked in* because you closed (sold, for a long position) part of the position. If Alice later sells her 100 at 10.30, she realizes $100 \times (10.30 - 10.00) = \mathbf{+30.00}$, and that is final.

For a short position the sign flips. Bob is short 100 from 10.00 and the price rose to 10.50, so he has lost: $-100 \times (10.50 - 10.00) = \mathbf{-50.00}$ on paper.

21.2.6 Equity

Add the two together and you get one number for "how well am I doing in total on this symbol": **realized + unrealized P&L**. This page calls it **equity** (EduMatcher's tables call the parts `realized_pnl` and `unrealized_pnl`). Equity does not care *how* you gained — banked profit and paper profit count the same — which is exactly why it is the right number to settle.

21.2.7 Settlement

Settlement is the act of turning "who gained and who lost" into "who pays and who receives". Real exchanges do this every day for fast-moving products, because letting a large loss pile up unpaid for weeks is dangerous. EduMatcher does the same:

> **Each day, each trader settles the *change* in their equity since the previous close.**

On day 1 the "previous close" is nothing (everyone starts at zero). On day 2 it is the end of day 1, and so on. Settling *changes* means a gain is paid exactly once. If Alice's equity was +50.00 yesterday and +20.00 today, today's settlement is **−30.00**: she gave back 30.00 of her paper gain.

Why settle paper profits at all?

It might feel strange to pay out money for a gain that has not been locked in. The reason is fairness and simplicity: every day each trader's account is brought up to date with the market, so nobody can hide a growing loss in an open position. Futures exchanges in the real world work this way (it is called *daily mark-to-market settlement*). And because each day settles only the *change*, it all adds up: once a trader closes everything, their total balance equals their locked-in profit, as the worked example below shows.

21.2.8 The central pool (who pays whom)

A natural question: "if Alice gains 50.00, who pays her?" The honest answer is that Alice did not trade with a single, fixed opponent all day — she may have traded with many people, and the exchange matches anonymously. So EduMatcher does not try to invent a "Bob pays Alice" payment. Instead everyone settles with **the exchange itself**, which acts as a **central counterparty** — the single middleman that every trader deals with. In this chapter we call that the **central pool**:

- A trader with a gain is **owed** that amount by the pool: *"the exchange owes you 50.00"*.
- A trader with a loss **owes the pool** that amount: *"you owe the exchange 50.00"*.

Adding **netting** to this: each trader gets one single net amount per day for all their trades together (not one payment per trade).

Because every trade has exactly one buyer and one seller, every unit of money one trader gains is a unit another loses. The sum of everyone's amounts is therefore **exactly zero**. Systems with this property are called **zero-sum**. Zero-sum is the safety check at the heart of `pm-eod`: if the amounts do not add up to zero, something is wrong with the data and `pm-eod` refuses to write anything.

21.2.9 Balance and opening balance

A trader's **balance** is the running total of all their daily amounts. It starts at the trader's **opening balance** — zero unless you give them a starting amount in the roster — and each settled day adds that day's amount (positive or negative). A negative balance means "this trader is net down overall". It is only information; it does not block trading.

21.2.10 Currency, minor units and rounding

Money amounts are shown with **two decimals** (`12.34`). Inside the program they are stored as whole numbers of the smallest unit — the **minor unit**, the cent or penny: 12.34 is stored as `1234`. Whole numbers cannot drift the way decimals can ("0.1 + 0.2 = 0.30000000000000004" is a classic computer trap).

Real P&L has more precision than two decimals, so each amount must be **rounded**. Rounding each trader independently could leave the total off by a cent. Example: three traders have exact amounts of +0.333, +0.333 and −0.666. Rounded one by one: +0.33, +0.33, −0.67 → total −0.01, not zero. `pm-eod` instead rounds so that the rounded amounts *still add up* (the *largest remainder method*: round everyone down, then hand the leftover cents to the amounts that were rounded down the most). Here the result is +0.33, +0.33, −0.66 → total 0.00. If a tiny leftover truly cannot be placed (because of floating-point noise of a fraction of a cent), it is booked to a special account called `EXCHANGE` as a *rounding* line, so that the books always sum to exactly zero.

21.2.11 The trading date

Each trading day belongs to a **trading date** — the calendar date *in the exchange's own timezone* (`locale.session_timezone` in the configuration; default `Europe/Stockholm`). It is not the date on the server's clock and not the UTC date. Why it matters: a market that runs into the late evening can straddle midnight in UTC. All the exchange's daily figures — statistics, clearing, settlement — use this one definition so they always agree.

21.2.12 Other words used on this page

Word	Meaning
Ledger	A permanent, append-only record of money movements, like a bank statement. Here: the `eod.db` database.
Snapshot	A frozen copy of something at one moment. The *close snapshot* is every trader's position saved at the closing price.
Statement	A page sent to one trader summarising their day, like a bank statement.
Roster	The list that connects gateway identifiers to people (name, email).
Leaderboard	Traders ranked by balance, best first.

Word	Meaning
Gate	A checkpoint that must pass before work continues.
Idempotent	Safe to repeat: running it twice gives the same result as running it once. `pm-eod` is idempotent.
Audit trail	The exchange's complete, timestamped record of every message (Audit Trail).
Retention	How long old files are kept before they are deleted.
Compression	Storing a file in less space (here with the `zstd` format; typically 10× smaller for these files).
Backup	A second copy kept in case the original is lost or damaged.
Integrity check	The database's own self-test that none of its pages are corrupt.
Exit code	The number a command returns to the shell: 0 means "all well".

21.3 A complete worked example

Three traders: Alice (`GW_ALICE`), Bob (`GW_BOB`) and Carol (`GW_CAROL`). Alice starts with an **opening balance of 100.00**; the others start at 0. The symbol is `AAPL`.

21.3.1 Day 1 — Monday 2026-11-02

Alice buys 100 `AAPL` from Bob at **10.00**. It is the only trade of the day, so it is also the last one: the closing price is **10.00**.

Trader	Position at close	Average cost	Unrealized P&L at 10.00	Equity today	Equity yesterday	**Settlement**
Alice	+100	10.00	100 × (10.00 − 10.00) = 0.00	0.00	0	**0.00**
Bob	−100	10.00	−100 × (10.00 − 10.00) = 0.00	0.00	0	**0.00**
Carol	0	–	–	0	0	**0.00**

Zero-sum check: 0.00 + 0.00 + 0.00 = **0.00**.

Nobody gained or lost: a trade on its own is not a profit. Alice paid 10.00 for shares that are worth 10.00 at the close. What matters is what the price does *next*.

Running balances: Alice **100.00** (her opening balance), Bob **0.00**, Carol **0.00**.

21.3.2 Day 2 — Tuesday 2026-11-03

Carol buys 50 `AAPL` from Bob at **10.20**. That is the last trade of the day, so the closing price is **10.20**: the price has risen by 0.20.

Trader	Position at close	Average cost	Unrealized P&L at 10.20	Equity today	Equity yesterday	**Settlement**
Alice	+100	10.00	100 × (10.20 − 10.00) = +20.00	+20.00	0.00	**+20.00** (exchange owes her)

Trader	Position at close	Average cost	Unrealized P&L at 10.20	Equity today	Equity yesterday	**Settlement**
Bob	−150	10.0667	−150 × (10.20 − 10.0667) = −20.00	−20.00	0.00	**−20.00** (he owes the exchange)
Carol	+50	10.20	50 × (10.20 − 10.20) = 0.00	0.00	0	**0.00**

Zero-sum check: +20.00 − 20.00 + 0.00 = **0.00**.

Alice did not trade today, yet she is owed 20.00: she *holds* 100 shares and their price rose (that is *marking to market*). Bob is short, so the same rise costs him. His average sale price is (100 × 10.00 + 50 × 10.20) ÷ 150 = **10.0667**, and with 150 shares short at a closing price of 10.20 he is 20.00 down on paper. Carol traded but her settlement is zero, because she bought at the closing price itself. Settlement follows the *change in value*, not trading activity.

Running balances: Alice 100.00 + 20.00 = **120.00**, Bob **−20.00**, Carol **0.00**. These are exactly the numbers `pm-eod balances` printed when this example was run (see below).

21.3.3 Day 3 — Wednesday 2026-11-04 (Alice cashes in)

Alice sells all her 100 `AAPL` to Carol at **10.30**, the last trade of the day, so the closing price is **10.30**.

Trader	Position at close	Average cost	P&L at 10.30	Equity today	Equity yesterday	**Settlement**
Alice	0	–	realized 100 × (10.30 − 10.00) = +30.00	+30.00	+20.00	**+10.00**
Bob	−150	10.0667	−150 × (10.30 − 10.0667) = −35.00	−35.00	−20.00	**−15.00**
Carol	+150	10.2667	150 × (10.30 − 10.2667) = +5.00	+5.00	0.00	**+5.00**

Zero-sum check: +10.00 − 15.00 + 5.00 = **0.00**. Running balances: Alice **130.00**, Bob **−35.00**, Carol **5.00**. (Carol's average cost is (50 × 10.20 + 100 × 10.30) ÷ 150 = 10.2667.)

Alice is flat now, so she has no unrealized P&L left; her profit is **realized**. Look at her amounts over the three days: 0.00, +20.00, +10.00 = **+30.00**, which is exactly her locked-in profit. *This is the reassuring property of settling changes:* once you are flat, the sum of everything settled equals the profit you actually made. (Her 130.00 balance is that 30.00 plus the 100.00 she started with.)

21.3.4 The same example, run for real

This is the actual output of `pm-eod` for days 1 and 2 above (with a roster giving Alice an opening balance of 100.00), run after each day's close:

```
2026-11-02:
  [ok] gate: closed, snapshot present, reconciled
  [ok] settle: settled
  [ok] report: wrote /srv/exchange/data/settlement/2026-11-02
  [ok] audit: nothing pending
  [ok] logs: no log directory
  [ok] backup: backed up clearing.db, eod.db (132.0 KB -> 3.3 KB); not present: stats.db
2026-11-03:
  [ok] gate: closed, snapshot present, reconciled
  [ok] settle: settled
  [ok] report: wrote /srv/exchange/data/settlement/2026-11-03
  [ok] audit: nothing pending
  [ok] logs: no log directory
  [ok] backup: backed up clearing.db, eod.db (132.0 KB -> 4.1 KB); not present: stats.db
```

```
account  | balance
---------+--------
GW_ALICE | 120.00
GW_BOB   | -20.00
GW_CAROL | 0.00
```

Running day 3 as well gives 130.00, −35.00 and 5.00, as in the table above.

The "not present: stats.db" part is normal in this demonstration, which ran without `pm-stats`; on a real exchange `stats.db` is backed up too.

21.4 Setting it up

21.4.1 What must be running

Requirement	Why
`pm-clearing` running all day	It writes the close snapshot `pm-eod` settles from. It is part of the `default` profile of `pm-opctl-cli`.
Sessions enabled (`sessions_enabled: true`) and `pm-scheduler --daily`	The scheduler drives the market to `CLOSED`, which is what triggers `pm-eod`. Without it see Running without the scheduler.
`pm-eod` installed	It is installed with the other `pm-*` commands. If you run from a source checkout and updated it, run `poetry install` once so the new command exists.

Not in the mini profile

The `mini` profile has no `pm-clearing`, so there is nothing to settle and its scheduler is deliberately started *without* the hook.

21.4.2 The roster file

The roster tells `pm-eod` who is behind each gateway. It is **optional**: with no roster every statement is addressed to the bare gateway identifier. But a roster makes the statements readable and gives you the email addresses for sending them.

Create `ref_data/roster.yaml` inside your data directory (next to `engine_config.yaml`):

```
traders:
  - gateway: GW_ALICE
    name: Alice Andersson
    email: alice@example.org
    opening_balance: 100.00
  - gateway: GW_BOB
    name: Bob Berg
    email: bob@example.org
  - gateway: GW_CAROL
    name: Carol Carlsson
    email: carol@example.org
```

Field	Required	Meaning
`gateway`	**Yes**	The gateway identifier exactly as the trader logs in with it. Letters are upper-cased automatically (`gw_alice` becomes `GW_ALICE`). Must be unique. May not be `EXCHANGE`, which is reserved.
`name`	No	Printed on the statement. Defaults to the gateway identifier.
`email`	No	Printed on the statement and written to `instructions.csv`. **Never used to send anything.**
`opening_balance`	No (default 0)	The trader's starting balance in the fictional currency, a plain number with **at most two decimals**. Negative numbers are allowed.

What happens in the corner cases:

- **A trader in the roster who never traded** still gets a statement every day (amount 0.00, balance = opening balance), so everybody enrolled receives something every day.
- **A gateway that traded but is not in the roster** still gets a statement, addressed to the gateway identifier and marked *"This gateway is not in the roster."* The run and the summary also list a warning (`gateway GW_X is not in the roster`), so you notice and add them.
- **A mistake in the file** stops the *report* step with a message that names the line, for example `roster.yaml: line 7: opening_balance has more than 2 decimals`. The settlement itself is kept (the day's status is `report-failed`); fix the file and run `pm-eod run --date <date>` again — *without* `--only`, so that the day is recorded as `ok` (a run with `--only` is never recorded). The next `pm-eod run --pending` does the same by itself.

Opening balances are applied when displaying, not when settling

The ledger stores only each day's *amounts*. Opening balances come from the roster every time a balance is shown. If you change an `opening_balance` later, every balance shown from then on changes — but statements already written keep the old numbers until you regenerate them with `pm-eod run --date <date> --only report`.

21.4.3 Starting the hook

The simplest way is to let the scheduler start `pm-eod` for you after each close:

```
# checkdocs: ignore  (the quoted command has its own flags)
pm-scheduler --daily --on-close "pm-eod run --pending"
```

This is already in the `default` profile of `pm-opctl-cli`, so `pm-opctl-cli start` does it for you. What happens at each close:

1. The scheduler moves the engine to `CLOSED` and waits for the engine to confirm.
2. **It waits 30 seconds**, then starts the command. The pause gives `pm-clearing` time to save the close snapshot. Change it with `--on-close-delay SECONDS`.
3. The command runs in the background. The scheduler does **not** wait for it and goes on with its timetable.

4. Its output appears in the scheduler's log, and the exit code is logged (`--on-close command exited 1` is a warning that something failed).

Also:

- The command runs **once more when the scheduler starts**. That is how a day that closed while the scheduler was down gets settled.
- Only **one** `pm-eod` runs at a time. A trigger that finds one still running is skipped (with a warning); that is safe because `--pending` always picks up every date that is waiting.
- In `--now` test mode the hook is ignored.
- `--on-close` takes any command, so you may add your own mailing script, for example `--on-close "sh /srv/exchange/close.sh"` where the script runs `pm-eod run --pending` and then sends the mail.

21.4.4 Running without the scheduler

If `sessions_enabled` is `false` (an always-open market) nothing ever "closes", so no event can trigger `pm-eod`, and you decide when a "day" is over:

1. **Stop the engine gracefully** at the end of your session (`Ctrl-C` or `SIGTERM`, not `kill -9`). On a graceful shutdown the engine publishes its end-of-day message; `pm-clearing` then marks every position at the last price and saves the close snapshot.
2. Settle that date yourself, telling `pm-eod` to skip the "has the day closed?" test with `--force`:

```
pm-eod run --date 2026-11-02 --force
```

`--force` skips *only* the closed-day test. The other checks (a close snapshot exists; the books reconcile) still apply. The snapshot comes from `pm-clearing` only on the end-of-day message or on `CLOSED`, so a market that is simply left running has nothing to settle from.

A crontab line that settles yesterday every morning at 06:00, for an exchange you shut down every weekday evening, looks like this:

```
0 6 * * 2-6  pm-eod run --date "$(date -d yesterday +\%F)" --force
```

`2-6` means Tuesday to Saturday, so it settles Monday to Friday. Run it on a morning after a day without trading and it fails with `no close snapshot for <date>` (exit code 1): harmless, but cron mails you about it.

If you skip a day (you forgot to run it, or its run failed and you moved on), nothing is lost: the next date you settle counts from the close of the last date that *was* settled, so it covers both days in one amount.

Stopping the engine more than once a day

Each graceful shutdown replaces that date's snapshot, so if you stop the engine at lunch and again in the evening, the evening one is settled and the afternoon's trading counts for that day. (On an exchange *with* a scheduler, the snapshot taken at the real market close is final: a later shutdown or restart that day never replaces it.)

21.4.5 Where everything lives

Everything is under the **data directory** (`EDUMATCHER_DATA_DIR`, see Installation). Pass `--datapath DIR` to `pm-eod` to use another one.

Path (under the data directory)	What it is	Who writes it
`clearing.db`	Trades, positions and the close snapshots (the input)	`pm-clearing`
`stats.db`	Market statistics — only used to fill the "Market" table of the summary	`pm-stats`
`ref_data/roster.yaml`	Who is behind which gateway	**You**
`eod.db`	The settlement **ledger**: each day's amounts and the log of runs	`pm-eod`
`settlement/<date>/`	The day's statements and summary	`pm-eod`

Path (under the data directory)	What it is	Who writes it
`backup/<date>/`	Compressed copies of the databases	`pm-eod`
`audit/`, `audit.log`	The audit trail (day parts and `.zst` archives)	`pm-audit`
`logs/`	Fallback log files (written when the log server is unreachable)	every `pm-*` process
`eod/<date>/`	**The engine's own** end-of-day files — not touched by `pm-eod`	`pm-engine`
`.pm-eod.lock`	Stops two `pm-eod` runs overlapping	`pm-eod`

21.5 A normal day

Nothing for you to do. For example, with the market closing at 17:30:

Time	What happens
17:30:00	The engine closes. `pm-clearing` saves the close snapshot.
17:30:30	The scheduler starts `pm-eod run --pending`.
17:30:31	`pm-eod` settles, writes the statements, does the housekeeping.
a few seconds later	It exits. `settlement/<date>/` is ready.

To check on it:

```
pm-eod status
```

```
trade_date | status | started_at                       | finished_at                      | steps
-----------+--------+----------------------------------+----------------------------------+-----------------------------------------
2026-11-03 | ok     | 2026-11-03T16:30:30.007000+00:00 | 2026-11-03T16:30:30.014000+00:00 | gate:ok settle:ok report:ok audit:ok logs:
2026-11-02 | ok     | 2026-11-02T16:30:30.007000+00:00 | 2026-11-02T16:30:30.014000+00:00 | gate:ok settle:ok report:ok audit:ok logs:
```

The times are in UTC: 16:30:30 UTC is 17:30:30 in Stockholm, thirty seconds after the close.

The `status` column is one of:

`status`	Meaning
`ok`	Settled **and** reported. (Housekeeping warnings do not change this; they are listed in `steps`.)
`report-failed`	Settled, but the statements could not be written. The money is safe in the ledger; the next `pm-eod run --pending` writes the report.
`failed`	The gate or the settlement failed. Nothing was recorded for the day.

`pm-eod status` also prints `pending: <dates>` (on the error stream) when there are closed days that have not been done yet.

For monitoring, `pm-eod status -q` prints nothing and returns **exit code 0** when no closed day is waiting to be done, and **1** otherwise. A closed day whose run failed still counts as waiting, so the check stays red until it is fixed. (A failed run for a date that never closed, such as a mistyped date, does not.) Use it from a health check or a cron alert.

21.5.1 The balances

```
pm-eod balances
```

```
account  | balance
---------+--------
GW_ALICE | 120.00
GW_BOB   | -20.00
GW_CAROL | 0.00
```

Useful variations (note that `--format` is a global option and goes *before* the word `balances`):

```
pm-eod --format json balances --gateway gw_alice   # one trader, as JSON
pm-eod balances --date 2026-11-02                  # as they stood after that day
pm-eod balances --check                            # verify the books
```

`--check` re-adds every settled day and reports any day that does not sum to zero. Healthy output is `OK - every settled date sums to zero`.

21.5.2 One day in detail

`pm-eod show` reads one date straight from the ledger (`eod.db`):

```
pm-eod show --date 2026-11-02               # every account's amount and per-symbol P&L
pm-eod show --date 2026-11-02 --gateway gw_alice
pm-eod show --date 2026-11-02 --steps       # the run's steps with their full messages
```

```
trade_date | account  | amount  | detail
-----------+----------+---------+----------------------
2026-11-02 | GW_ALICE | 120.00  | ACME=100.00 BOLT=20.00
2026-11-02 | GW_BOB   | -20.00  | ACME=-20.00
2026-11-02 | GW_CAROL | -100.00 | ACME=-80.00 BOLT=-20.00
```

`amount` is the day's settlement in display-currency units: **positive means the exchange owes the trader, negative means the trader owes the exchange**. The `detail` column is the same result split by symbol before rounding. The amounts of one date always add up to zero; a rounding residual, if any, sits on the `EXCHANGE` account. With `--steps`, a date with no recorded run shows no rows.

21.5.3 Exit codes of `pm-eod run`

Code	Meaning	What to do
0	Clean — everything succeeded	Nothing
1	The day failed (gate, settle or report)	Read the `[FAIL]` line; see Troubleshooting
2	Settled and reported, but a housekeeping step warned	Read the `[warn]` line; the statements are fine

21.6 The files `pm-eod` writes

For each settled date a folder `settlement/<date>/` is created:

```
settlement/2026-11-03/
├── instructions.csv        who pays / receives what (for mail-merge)
├── leaderboard.csv         traders ranked by balance
├── summary.md              the instructor's one-page summary
├── summary.json            the same, machine-readable
└── statements/
    ├── GW_ALICE.md  GW_ALICE.json  GW_ALICE.csv
    ├── GW_BOB.md    GW_BOB.json    GW_BOB.csv
    └── GW_CAROL.md  GW_CAROL.json  GW_CAROL.csv
```

The folder is built in a hidden work directory and renamed into place when complete. You never see a half-written day, and re-running a date replaces it cleanly.

21.6.1 The statement (`statements/<gateway>.md`)

This is what a trader reads. Here is Bob's statement for day 2 of the example (see also Your End-of-Day Statement, which explains it line by line to the trader):

```
# Statement for 2026-11-03

Bob Berg (GW_BOB) <bob@example.org>

**You owe the exchange 20.00.**

- Today's result: -20.00
- Balance after today: -20.00

## Result by symbol

| Symbol | Result |
|---|---:|
| AAPL | -20.00 |

## Open positions at the close

| Symbol | Quantity | Closing price |
|---|---:|---:|
| AAPL | -150 | 10.20 |

## Trades

| Time (UTC) | Symbol | Side | Quantity | Price |
|---|---|---|---:|---:|
| 10:30:01 | AAPL | SELL | 50 | 10.20 |
```

The headline sentence is always one of three:

Sentence	When
The exchange owes you X.	Today's amount is positive
You owe the exchange X.	Today's amount is negative
Nothing to pay or receive today.	The amount is zero

Three details worth knowing. **Each line of "Result by symbol" is rounded on its own**, so with several symbols the lines can add up to 0.01 more or less than "Today's result"; the total is the amount that counts (it is rounded so that all traders together sum to exactly zero). **Trade times are in UTC**, not the exchange's timezone. And the **trade list comes from `trade_events`**, which `pm-clearing` prunes after 90 days by default. Settling a very old date later still gives the correct amounts (those come from the close snapshots, which are never pruned) but the trade list may be empty.

21.6.2 The same statement as JSON and CSV

`statements/GW_BOB.json` carries the same information for programs. Amounts appear twice: as whole minor units (`amount_minor`: `-2000`) for exact arithmetic, and as text (`amount`: `"-20.00"`) for display.

JSON field	Meaning
`trade_date`, `gateway`, `name`, `email`	Who and which day
`listed`	`true` if the gateway is in the roster
`amount_minor` / `amount`	Today's settlement (positive = exchange owes the trader)
`balance_minor` / `balance`	Running balance after today, including the opening balance

JSON field	Meaning
`symbol_pnl`	Today's amount broken down by symbol. A symbol the trader no longer holds, did not trade today and made nothing on today is left out (otherwise every symbol ever traded would show 0.00 forever)
`positions`	Open positions at the close: `symbol`, `net_qty`, `mark` (closing price)
`trades`	Today's trades: `time_utc`, `symbol`, `side` (`BUY`, `SELL`, or `SELF`), `quantity`, `price`

`statements/GW_BOB.csv` has one row per symbol, handy for a spreadsheet:

```
symbol,day_pnl,net_qty,mark
AAPL,-20.00,-150,10.20
```

21.6.3 `instructions.csv` — the payment instructions

One line per trader. This is the file to use for sending the statements or for a classroom "who pays whom" board:

```
gateway,name,email,amount,direction,balance
GW_ALICE,Alice Andersson,alice@example.org,20.00,receive,120.00
GW_BOB,Bob Berg,bob@example.org,-20.00,pay,-20.00
GW_CAROL,Carol Carlsson,carol@example.org,0.00,none,0.00
```

Column	Meaning
`amount`	Today's settlement. Positive = the exchange owes the trader.
`direction`	`receive` (amount > 0), `pay` (amount < 0) or `none` (0)
`balance`	Running balance after today

The `direction` column is from the *trader's* point of view: `pay` means *this trader pays the exchange.*

21.6.4 `leaderboard.csv`

Traders ranked by running balance, highest first. Traders with an equal balance share a rank (and the next rank is skipped, as in sports tables):

```
rank,gateway,name,balance
1,GW_ALICE,Alice Andersson,120.00
2,GW_CAROL,Carol Carlsson,0.00
3,GW_BOB,Bob Berg,-20.00
```

The leaderboard includes opening balances

If you gave some traders a head start with `opening_balance`, the ranking includes it. For a fair competition give everyone the same opening balance (or none).

21.6.5 `summary.md` / `summary.json` — for the instructor

```
# Exchange summary for 2026-11-03

- Accounts settled: 3
- Sum of all accounts: 0.00
- Rounding booked to the exchange: 0.00

## Biggest winners

- GW_ALICE: 20.00

## Biggest losers

- GW_BOB: -20.00

## Market

Market statistics unavailable.
```

- **Sum of all accounts** is the zero-sum proof. It should always read `0.00`.
- **Rounding booked to the exchange** is normally `0.00`; a non-zero value of a cent or two is the harmless rounding leftover described above.
- **Market** lists volume, number of trades and the closing price per symbol, taken from `stats.db`. If `pm-stats` is not running (as in this demo) the summary says *"Market statistics unavailable."*
- **Warnings** (for example unlisted gateways) are listed at the end.

21.6.6 Sending the statements

`pm-eod` stops at files, on purpose: sending mail needs credentials and a decision about wording that belong to you. `instructions.csv` is the ready-made input. A minimal script that prints one draft message per trader (replace the `print` with `smtplib` or your mail tool when you are ready):

```
import csv
import sys
from pathlib import Path

day = sys.argv[1]  # for example 2026-11-03
base = Path("settlement") / day
with (base / "instructions.csv").open() as fh:
    for row in csv.DictReader(fh):
        if not row["email"]:
            continue
        statement = (base / "statements" / f"{row['gateway']}.md").read_text()
        print(f"To: {row['email']}")
        print(f"Subject: Exchange statement {day}")
        print()
        print(statement)
        print("-" * 40)
```

Or simply hand each trader their own `statements/<gateway>.md` file, or post `leaderboard.csv` on the classroom screen.

21.7 The housekeeping steps in detail

After the money work, `pm-eod` tidies up. All three steps are **safe to repeat** and report problems as warnings.

21.7.1 `audit` — the audit-trail safety net

`pm-audit` already compresses each audit-trail day into `audit/audit-<date>.log.zst` shortly after midnight. The `audit` step is a **safety net**: it compresses any earlier day that was missed (for instance because `pm-audit` was stopped over midnight). It never touches today's data.

The message `nothing pending` is the normal outcome. Note that the day that has *just* closed cannot be archived yet — a day is only archivable once it is over — so it is `pm-audit` that archives it after midnight.

21.7.2 `logs` — compressing and tidying the fallback logs

When a process cannot reach the log server for 30 seconds it falls back to writing its own file `logs/<name>.log` (see Log Server). These files only exist after an outage, but they can grow. The step:

1. Looks at each `logs/*.log` that has **not been written to for at least one hour** and is not empty.
2. Writes its content, compressed, to `<name>.<YYYYMMDD>.log.zst` (a counter is added if that name exists).
3. Checks the compressed copy by decompressing it again.
4. **Empties the original file in place — it does not delete it.** A running process keeps its log file open; deleting it would silently lose everything it writes afterwards. If the file changed while being compressed, the step leaves it alone, throws the copy away and warns (`1 changed while compressing`); the next run tries again.
5. Deletes compressed logs older than `--log-keep-days` (default **14**).

21.7.3 `backup` — verified copies of the databases

For `clearing.db`, `stats.db` and `eod.db` — the files you cannot recreate — the step:

1. Asks SQLite for a **consistent online copy** (SQLite's backup mechanism). This is safe while the exchange is running; you do not have to stop anything.
2. Runs the database's **integrity check** on the copy.
3. Compresses it to `backup/<date>/<name>.db.zst` and **decompresses it again to compare** — only a copy proven identical replaces anything.
4. Deletes backup folders older than `--backup-keep-days` (default **7**).

A run that catches up several days takes one backup at the end, under the last day's date (see The scheduler was down).

To leave a database out — for example a very large `stats.db` that you can do without — use `--backup-skip`, which may be repeated: `pm-eod run --pending --backup-skip stats.db`. The step's message then ends with `skipped by request: stats.db`, so a missing backup is never a surprise. Keep `clearing.db` and `eod.db`: they hold the trades and the money record.

If the backup fails, the *previous* backup of that date is left untouched. A database that does not exist is simply listed (`not present: stats.db`).

Compression typically makes these databases many times smaller (the small demonstration above went from 132 KB to 3.3 KB; a real one shrinks less, because real data is less repetitive).

Restoring a backup

You need the `zstd` command (or Python 3.14, which has `compression.zstd` built in). **Stop the processes that use the database first**, then:

```
# 1. Decompress the backup to a new file
zstd -d backup/2026-11-03/clearing.db.zst -o /tmp/clearing-restored.db

# 2. Look at it before trusting it
sqlite3 /tmp/clearing-restored.db "PRAGMA integrity_check; SELECT COUNT(*) FROM trade_events;"

# 3. Move the broken files aside and put the restored one in place
mv clearing.db clearing.db.broken
rm -f clearing.db-wal clearing.db-shm
cp /tmp/clearing-restored.db clearing.db
```

Without the `zstd` tool, Python can do step 1:

```
from compression import zstd
from pathlib import Path

src = Path("backup/2026-11-03/clearing.db.zst")
Path("/tmp/clearing-restored.db").write_bytes(zstd.decompress(src.read_bytes()))
```

Delete the old `-wal` and `-shm` files (step 3). They belong to the broken database and SQLite would otherwise try to apply

them to the restored one.

21.7.4 Disk space

The housekeeping *saves* space overall, but keep an eye on three things: `settlement/` (small — a few kilobytes per trader per day), `backup/` (daily compressed copies, kept 7 days by default), and the audit archives (`pm-audit --keep-days`, default: keep everything). Change the retention with `pm-eod run --log-keep-days N --backup-keep-days N` and leave a large database out of the backup with `--backup-skip`; in the scheduler hook write it into the command: `--on-close "pm-eod run --pending --backup-keep-days 14"`.

21.8 Correcting mistakes and special situations

21.8.1 Run a day again

`pm-eod run --date 2026-11-03` is always safe. A day that is already settled says `already settled` and is not counted twice; the report and housekeeping run again (replacing the folder). To run only one step of a settled day:

```
pm-eod run --date 2026-11-03 --only report     # rewrite the statements
pm-eod run --date 2026-11-03 --only backup     # take the backup again
```

`--only` accepts `report`, `audit`, `logs` or `backup`. A step run with `--only` is not recorded in `pm-eod status`, so it can never mark a good day as bad.

21.8.2 Look before you leap: `--dry-run`

```
pm-eod run --date 2026-11-03 --dry-run
```

runs the checks and prints the amounts that *would* be settled, writing nothing at all:

```
2026-11-03:
  [ok] gate: closed, snapshot present, reconciled
  [ok] settle: would settle: GW_ALICE +20.00, GW_BOB -20.00, GW_CAROL +0.00
```

21.8.3 The scheduler was down, or the machine was off

Nothing special: the next `pm-eod run --pending` (the scheduler runs it when it starts) settles every closed day that is still waiting, **oldest first**. If one of them fails, it stops there and says `stopping: later dates wait for <date>`, because a later day must never be settled before an earlier one.

Catching up several days takes **one** backup, not one per day: every day would copy the same current databases, only under an older date. Each day's `backup` line says `deferred: one backup after the run's last date`, and the run ends with the backup itself, filed under the last day it got through. The worked example, with the scheduler down for all three days and `--pending` run once afterwards, ends like this:

```
2026-11-04:
  [ok] gate: closed, snapshot present, reconciled
  [ok] settle: settled
  [ok] report: wrote /srv/exchange/data/settlement/2026-11-04
  [ok] audit: nothing pending
  [ok] logs: no log directory
  [ok] backup: deferred: one backup after the run's last date
2026-11-04:
  [ok] backup: backed up clearing.db, eod.db (132.0 KB -> 4.5 KB); not present: stats.db
```

If a day fails and the run stops, the backup is still taken, under the last day before it.

21.8.4 The engine was restarted during the day

Restarting the engine announces `CLOSED` for a moment before the scheduler moves it back to the right phase, so `pm-clearing` saves a *provisional* close snapshot. `pm-eod` treats a day as closed only when its **latest** session phase is `CLOSED` *and the market was open earlier that day*, so a day that is still trading is never settled early. The real close overwrites the provisional snapshot.

If the day never closes properly (the scheduler was stopped, say), it is never settled on its own: its gains and losses are included in the next day that is settled, because each settlement counts from the close of the last day that *was* settled.

Restarting the engine **after** the close (for maintenance, say) also announces `CLOSED`, but changes nothing: once a day has really closed its snapshot is final, and only the next trading day takes a new one.

The same goes for starting the exchange **in the morning**, before the open. The engine announces `CLOSED` when it starts and the scheduler runs `pm-eod run --pending` a moment later, but a day on which the market has not yet opened has not closed: `pm-eod` settles only earlier days, and today waits for the real close.

21.8.5 A trader joins or leaves mid-way

Add them to `roster.yaml`. If they trade, they are in the clearing data and settle normally even before you add them (they just appear without a name until you do). To fix up earlier statements run `--only report` for those dates.

21.8.6 Recomputing a day (`--redo`)

If something upstream was wrong and you want to recompute, use `--redo`:

```
pm-eod run --date 2026-11-03 --redo
```

`--redo` drops later days too

Balances are a running total, so correcting one day changes everything after it. `--redo` therefore **deletes the settlement of that date *and every later date*** from the ledger, recomputes the first one, and leaves the later ones to the next `pm-eod run --pending`. Statement folders of the dropped dates stay on disk until they are overwritten. If the recomputation fails, nothing is deleted. Run `--dry-run` first and make sure you have a backup (the `backup` step makes one every day).

21.8.7 After `pm-opctl-cli clear`

`pm-opctl-cli clear --state` wipes `clearing.db` — but it deliberately **keeps** `eod.db`, `settlement/` and `backup/`, because the balances should survive a reset of the working data. Only `clear --all` removes them. One consequence: if you reset and then trade again on the *same calendar date*, that date is already settled in the ledger and `pm-eod` will refuse to settle it again. Use `pm-eod run --date <date> --redo` to replace it.

21.8.8 Rehearsing with a fast clock

You can rehearse with `pm-scheduler --daily --speed 600`, which runs the timetable ten simulated minutes per second. The trading date comes from the *real* clock, though, so several fast simulated days all land on **today's** date and share one close snapshot. You will get one settlement per real calendar date, not one per simulated day. That is fine for checking the whole pipeline works end to end; use `--redo` to run it again on the same date.

21.8.9 Weekends and holidays

On a day with no schedule the market never opens, so there is no `CLOSED` event, no snapshot and nothing to settle. A Friday-to-Monday gap is simply one longer interval: Monday settles the change in equity since *Friday's* close. (Precisely: since the close of the last settled day, which is normally Friday.)

21.9 Troubleshooting

Every message below is printed on a `[FAIL]` or `[warn]` line.

21.9.1 Gate failures (exit code 1)

Message	What it means	What to do
`<path>/clearing.db not found`	There is no clearing database in the data directory `pm-eod` is looking at	Check `--datapath` / `EDUMATCHER_DATA_DIR`; check that `pm-clearing` ran

Message	What it means	What to do
`no session phase recorded for <date>`	`pm-clearing` saw no session change on that date	Was `pm-clearing` running and the scheduler enabled? For a no-scheduler setup add `--force`
`<date> has not closed yet`	The day has not opened *and* closed: trading is still going on, the close was never announced, or the market has not opened yet today (the `CLOSED` an engine announces when it starts is not a close)	Wait for the close, or use `--force` if you have no scheduler
`no close snapshot for <date>`	The closing positions were never saved — `pm-clearing` was down at the close, or the date is from before this feature existed	`pm-clearing-cli snapshots --date <date>` shows what exists. A missing snapshot cannot be made afterwards. Nothing is lost by leaving the day unsettled: the next day that is settled includes its gains and losses. Make sure `pm-clearing` runs at the next close
`clearing does not reconcile for <date> (N discrepancies; see pm-clearing-cli reconcile)`	The raw trade records and the daily totals disagree. Settling on inconsistent books would spread the error	Run `pm-clearing-cli reconcile --from <date> --to <date>` to see exactly which trader and symbol; fix the cause (a crash mid-write is the usual one) before settling

21.9.2 Settlement failures (exit code 1)

Message	What it means	What to do
`the day does not sum to zero: exact total X over N position(s)`	The amounts do not cancel out by more than rounding noise could explain — a real accounting inconsistency	Nothing was written; do **not** try to force it. Run `pm-clearing-cli reconcile --from <date> --to <date>` and look at `pm-clearing-cli snapshots --date <date>` for a trader whose numbers are far off. This points at a bug or a damaged `clearing.db`: keep the files (the daily backup has a copy) and report it
`a gateway may not be called EXCHANGE`	A trader used the reserved name	Rename the gateway in the configuration
`<date> has closed but is not settled; settle the dates in order (pm-eod run --pending)`	You asked for a date while an earlier closed day is still waiting. Days are settled oldest first, so the earlier one comes first	Run `pm-eod run --pending`, which takes them in order. If that fails on the earlier day, fix what its message says
`<date> is settled, which is later than <date>`	A later date was settled before this earlier one (for example you ran Friday and then Thursday by hand)	Run `pm-eod run --date <the earlier date> --redo`: it drops the later settlements and redoes them in order with the next `--pending`

21.9.3 Report failures (exit code 1, settlement kept)

Message	What it means	What to do
`<date> is not settled`	You ran `--only report` for a date that has no settlement	Run the date without `--only` first
`roster.yaml: line N: …`	The roster file has an error on that line	Fix it; run the date again without `--only` (or wait for the next `--pending`)
An `OSError` (disk full, permission denied)	The folder could not be written	Free space or fix permissions; the next run retries

21.9.4 Other

Message	Meaning
`pm-eod: another pm-eod run is active`	The lock file is held. Wait, or check for a stuck process (`ps aux | grep pm-eod`).
`--on-close command is still running; trigger skipped` (scheduler log)	A previous run has not finished; harmless — the next run catches up.
`could not start --on-close command` (scheduler log)	The command was not found. Is `pm-eod` on the scheduler's `PATH`?

21.9.5 Housekeeping warnings (exit code 2)

Message	Meaning
`another archive run holds the lock`	A `pm-audit-cli archive` is running. Harmless; the next run retries.
`archive failed: …`	The audit archive could not be made (often not enough free disk). The day's files are untouched.
`N changed while compressing`	A log was written while it was being compressed; it was left alone and will be retried.
`failed: clearing.db: …`	That database could not be backed up (full disk, damaged database). The previous backup is kept. A damaged database is serious: investigate now.

21.10 Questions beginners ask

Is any of this real money? No. The currency is fictional and `pm-eod` only writes numbers and files.

If I buy a share and the price goes up, do I get paid even though I haven't sold? In this system, yes — in the form of a positive settlement for that day. That is the *mark-to-market* idea. If the price falls back the next day, the next settlement is negative and takes it back. Over the whole life of a position the sum of the daily amounts equals the real profit or loss.

Why is my statement amount for today different from my P&L on the clearing tool? Clearing shows your **total** P&L (everything since the beginning). The statement shows today's **change**. Today's amount = total equity now minus total equity at the previous close.

Why does a trader who did nothing sometimes get a non-zero amount? Because they *hold* a position and the market moved. Settlement follows price moves on what you hold, not just your own trading activity.

Why do the amounts add to exactly zero? Every trade has a buyer and a seller. One's gain is the other's loss.

What if two traders disagree with the numbers? Everything is reproducible. `pm-clearing-cli snapshots --date <date>` shows the closing positions used, `pm-clearing-cli trades` shows the trades, and `pm-eod run --date <date> --dry-run` recomputes without writing.

Can a trader go below zero? Yes. A negative balance is allowed and has no effect on trading.

Why didn't `pm-eod` run today? Check `pm-eod status` (and `pending:`), the scheduler log for `--on-close`, and that the market actually closed (holidays have no close).

Where do I see the current day, before the close? `pm-clearing-cli pnl` and `pm-clearing-cli gateways` show live totals. Settlement only happens after the close.

21.11 Checklist for your first real day

1. `pm-clearing` is running (check with `pm-clearing-cli health`).
2. `sessions_enabled: true` and `pm-scheduler --daily` is running with `--on-close`.
3. You wrote `ref_data/roster.yaml` and every trader is in it.
4. You rehearsed once (see Rehearsing with a fast clock) and read the statement a trader would receive.
5. You know where `settlement/<date>/instructions.csv` appears and how you will send it.
6. You decided how long to keep backups and logs.
7. The next morning: `pm-eod status` shows `ok`, and `pm-eod balances --check` prints `OK`.

21.12 Command summary

Command	What it does
`pm-eod run --date D`	Settle, report and tidy up one trading date
`pm-eod run --pending`	Do every closed date that is waiting, oldest first
`pm-eod run --date D --dry-run`	Check and show the amounts, write nothing
`pm-eod run --date D --only STEP`	Re-run `report`, `audit`, `logs` or `backup` for a settled date
`pm-eod run --date D --force`	Skip the "has the day closed?" test (no scheduler)
`pm-eod run --date D --redo`	Recompute D and drop every later settlement
`pm-eod run … --backup-skip NAME`	Leave `clearing.db`, `stats.db` or `eod.db` out of the backup
`pm-eod status [-q]`	Recent runs; `-q` is a health check (0 = no closed day waiting)
`pm-eod balances [--date D] [--gateway G] [--check]`	Running balances, or verify the books

The complete option reference is in the Reference Manual and `pm-help pm-eod`. The words used here are collected in the Glossary.

21.13 See also

- Training Guide, chapter 20: Closing the Day — hands-on exercises: settle a day, read the statements, recover from a missed day, a broken roster and trading after the close
- Sessions and Scheduling — what drives the market to `CLOSED`, and the `--on-close` option
- Statistics and Reporting — the other daily figures
- Positions and P&L — how positions, average cost and profit are calculated
- Persistence — every file the exchange writes
- Audit Trail — the day-by-day archive the `audit` step safeguards
- Health Monitor — watching the exchange's processes

CHAPTER 22

ALF Gateway — Order Entry

Learning objectives

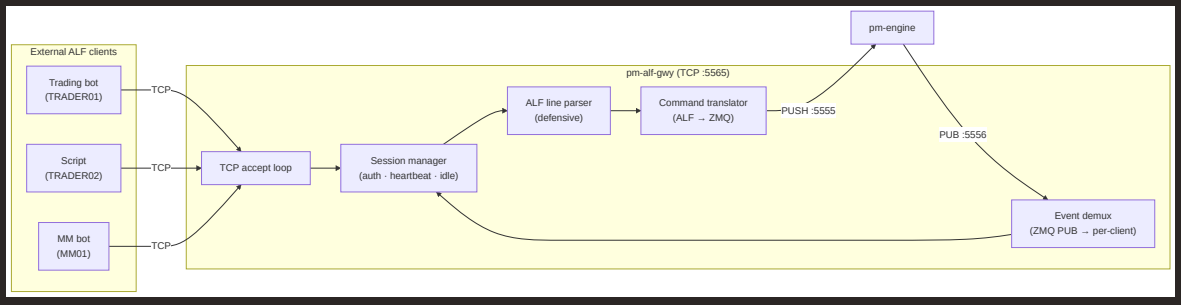
After reading this page you will understand:

- what `pm-alf-gwy` does and why it exists alongside `pm-alf-console`
- how to configure it in `engine_config.yaml`
- how to start it and verify connectivity from a terminal
- the session lifecycle: HELLO → WELCOME → commands → EXIT
- what commands are accepted and what responses to expect
- how multi-line responses (SYMBOLS, ORDERS, QBOOT, QLEGS, POS|GW=) are framed
- which broadcast events arrive unsolicited on every authenticated session
- how heartbeats, idle timeouts, and rate limiting work
- the error codes your client must handle
- how to write a minimal Python ALF client

22.1 What this process is

`pm-alf-gwy` is the **ALF TCP gateway**. The interactive `pm-alf-console` terminal is designed for a human at a keyboard — it reads stdin, prints to stdout, and connects to the engine ZMQ sockets directly. That architecture cannot serve an external bot or a process on another host.

`pm-alf-gwy` fills that gap. It binds a TCP port, accepts multiple simultaneous connections, validates each line defensively, and translates ALF commands into the same engine ZMQ messages `pm-alf-console` uses today. Engine responses are translated back to ALF-formatted lines and delivered over TCP.



22.1.1 What this is not

`pm-alf-gwy` accepts the same ALF command vocabulary as `pm-alf-console` but **does not support** interactive-terminal features:

Unsupported command	Reason
`STATUS`	Console session summary display — use `ORDERS` + `SYMBOLS` instead
`HELP`	Interactive terminal reference text

A bare `POS` (no `GW=`) is also rejected — `pm-alf-console`'s bare `POS` renders from its own local, session-scoped fill ledger, which `pm-alf-gwy` has no equivalent of. `POS|GW=<gateway_id>` **is** supported (see the command reference below) since that form is always a genuine engine round trip, never a local ledger — asking about your own connected gateway_id or any other gateway's (such as a running `pm-mm-bot`'s) works identically.

`QLEGS` **is** supported — see the command reference below. Unlike `pm-alf-console`'s `QLEGS` (which renders from its own local, session-scoped cache), `pm-alf-gwy` forwards the request to the engine over `system.quote_legs_request` and renders the engine's reply, including real `RECENT`/`ALL` history.

For interactive operator use, `pm-alf-console` remains the right tool. `pm-alf-gwy` is for programmatic clients and remote bots.

22.2 Prerequisites

- `pm-engine` running.
- Gateway IDs that will connect must be configured in `engine_config.yaml` under `participants`.
- Optional: add the `alf_gateway:` config section to customise port and limits.

22.3 Configuration

Add an `alf_gateway:` section to `engine_config.yaml`:

```
alf_gateway:
  enabled: true
  name: "alf-gwy01"
  bind_address: "0.0.0.0"
  port: 5565
  heartbeat_interval_sec: 5
  handshake_timeout_sec: 10
  idle_timeout_sec: 30
  max_connections: 1024
  max_client_queue: 10000
  max_commands_per_second: 100
  max_errors_before_disconnect: 50
  error_window_sec: 60
```

The gateway reads gateway roles from the existing `participants` list — no separate credentials block is needed. Any gateway ID listed in `participants` can connect to `pm-alf-gwy`.

Field	Default	Description
`enabled`	`true`	Master switch
`name`	`alf-gwy01`	Process name echoed in `WELCOME`
`bind_address`	`0.0.0.0`	Network interface to listen on (`127.0.0.1` for local-only)
`port`	`5565`	TCP listen port
`heartbeat_interval_sec`	`5`	Seconds between `HB` lines when no other outbound traffic
`handshake_timeout_sec`	`10`	Disconnect a connection that hasn't sent `HELLO` within this many seconds

Field	Default	Description
`idle_timeout_sec`	`30`	Disconnect after this many seconds of inbound silence. This default applies only when an `alf_gateway:` section is present but omits the key; if the section is absent entirely, the effective default is `3600`
`max_connections`	`1024`	Maximum simultaneous TCP connections. At start the gateway raises its open-file limit to `max_connections + 64` (up to the hard limit) and refuses to start if the hard limit is lower
`max_client_queue`	`10000`	Per-client outbound line buffer capacity
`max_commands_per_second`	`100`	Token-bucket rate limit per client
`max_errors_before_disconnect`	`50`	Error threshold in a sliding window before forced disconnect
`error_window_sec`	`60`	Width of the sliding window used to count errors toward `max_errors_before_disconnect`

22.3.1 One live connection per ID, and the engine heartbeat

A participant ID can have only **one** live connection to the exchange at a time, whichever process holds it — `pm-alf-gwy`, `pm-balf-gwy`, `pm-alf-console` or `pm-api-gwy`. If the ID is already connected, the engine refuses the second connect and `WELCOME` never arrives: the gateway answers `ERR|CODE=AUTH_FAILED|DETAIL=Gateway already connected: TRADER01` and closes. (Two processes sharing one ID would share one session, so either one dropping would cancel the other's orders.)

A process that dies without saying goodbye (`kill -9`, a crashed host, a pulled network cable) must not keep its ID locked for ever. So every process that holds an ID sends the engine a `system.gateway_heartbeat` (see Message Reference). `pm-alf-gwy` sends one per authenticated session every **5 seconds**, starting right after `WELCOME`, and a session that closes stops being reported. After **3 consecutive missed beats** (15 s) the engine disconnects the session as if it had said goodbye: the participant's `disconnect_behaviour` is applied and the ID can be connected again. So if `pm-alf-gwy` itself dies, its clients can log back on through a restarted gateway within about 15 seconds. (Single-ID processes such as `pm-alf-console` beat every 60 seconds, so their ID stays locked for up to three minutes.)

This engine heartbeat is **separate from** the `HB` / `PING` / `idle_timeout_sec` mechanism above, which is between your TCP client and the gateway and is much quicker (30 s by default). Your ALF client needs to do nothing extra for the engine heartbeat — the gateway sends it on the client's behalf. Its debug log line is `engine heartbeat sent gateway_id=<ID>` (run the gateway with `--log-level DEBUG`).

TLS

`pm-alf-gwy` does not terminate TLS. For remote deployments, put it behind a reverse proxy (nginx, stunnel, or similar).

22.4 Start the gateway

Installed mode:

```
pm-engine --verbose
pm-alf-gwy
```

Developer mode:

```
poetry run pm-engine --verbose
poetry run pm-alf-gwy
```

CLI override options:

Option	Default	Description
`--bind ADDR`	from config / `0.0.0.0`	Override TCP bind address
`--port PORT`	from config / `5565`	Override TCP listen port
`--engine-host HOST`	from config	Override engine host (sets `tcp://HOST:5555`, `tcp://HOST:5556`, and `tcp://HOST:5557` for drop copy)
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	Reduce output to warnings/errors

Config file location

The engine configuration is read from `<EDUMATCHER_DATA_DIR>/ref_data/engine_config.json`. There is no `--config` flag: every process reads that one file, so none of them can be started against a configuration the others have not seen. Install one with `pm-config-deploy`.

22.5 Quick connect test

Use `nc` or `telnet` to validate the session lifecycle before writing any code:

```
nc 127.0.0.1 5565
```

Type the following lines, pressing Enter after each:

```
HELLO|CLIENT=test|PROTO=ALF1|ID=TRADER01
SYMBOLS
EXIT
```

Expected response pattern:

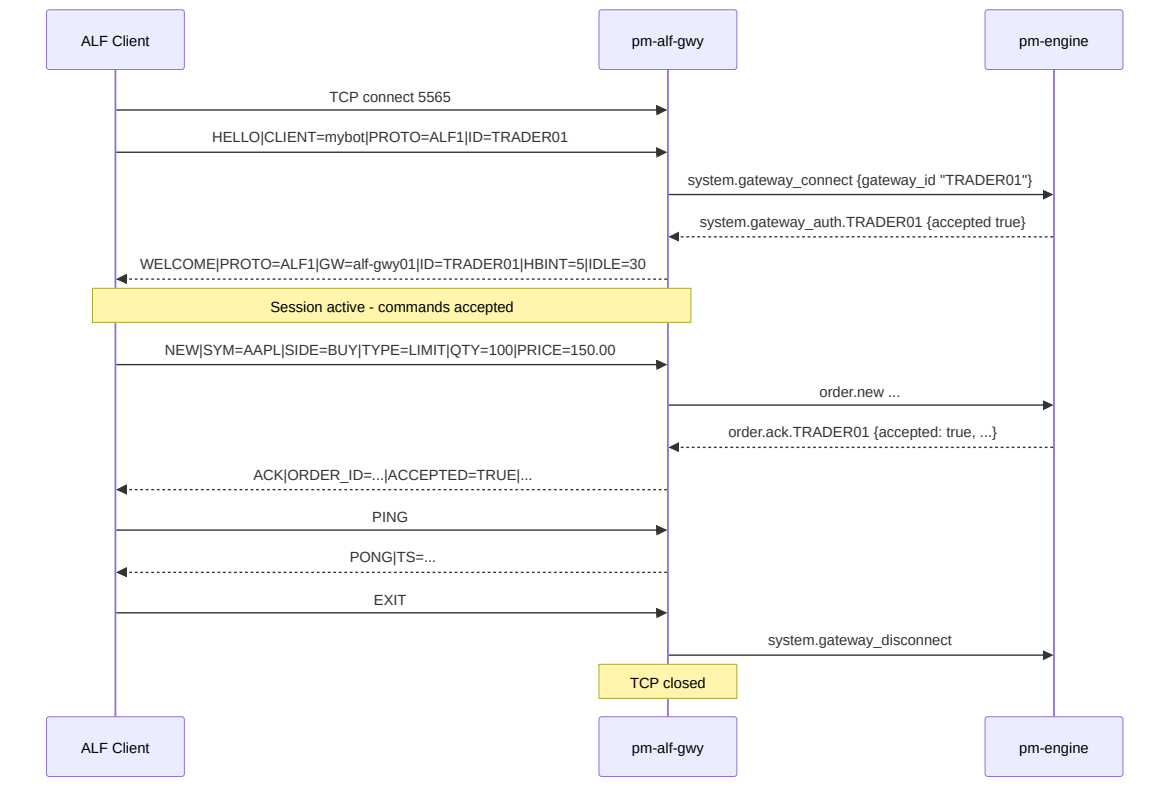
1. `WELCOME|PROTO=ALF1|GW=alf-gwy01|ID=TRADER01|HBINT=5|IDLE=30`
2. `SYMBOLS|COUNT=N` followed by one `SYMBOL|SYM=...|TICK=...` per instrument
3. `END|TYPE=SYMBOLS`
4. `HB|TS=...` every 5 seconds of quiet

TCP is a byte stream

Never assume one `recv()` equals one line. Always buffer and split on `\n` as shown in the Python example below.

22.6 Session lifecycle

Every ALF gateway session follows this exact sequence:



22.6.1 Step 1 — Send `HELLO`

The **first line** must be a `HELLO`:

```
HELLO|CLIENT=mybot|PROTO=ALF1|ID=TRADER01
```

Field	Required	Notes
`CLIENT`	Yes	Free-text label for logging (max 32 chars)
`PROTO`	Yes	Must be exactly `ALF1`
`ID`	Yes	Gateway ID that must be in `participants` in config: a letter, then up to 31 letters, digits or `_` (connection closed with `INVALID_VALUE` otherwise)
`FEED`	No	`ALL` (default) or `ORDERS`. `ORDERS` turns off the public broadcast events (`SESSION`, `HALT`, `RESUME`, `TRADE`) for this session, for clients that take market data elsewhere — such as the AI traders, which would otherwise each parse every trade on the exchange. Replies to your own commands and your own order events are unaffected

On any other first line the gateway sends `ERR|CODE=AUTH_REQUIRED|...` and closes the connection.

22.6.2 Step 2 — Authentication round trip

The gateway sends `system.gateway_connect` to the engine and waits for the `system.gateway_auth.<ID>` reply. The engine is authoritative: if the ID is not in the allowlist the gateway sends `ERR|CODE=AUTH_FAILED|DETAIL=...` and closes.

22.6.3 Step 3 — Receive `WELCOME`

```
WELCOME|PROTO=ALF1|GW=alf-gwy01|ID=TRADER01|HBINT=5|IDLE=30
```

The gateway immediately follows the `WELCOME` with a `SYMBOLS` multi-line response so the client knows which instruments are configured.

22.6.4 Step 4 — Send commands

After `WELCOME`, any supported command may be sent. Responses and broadcast events arrive asynchronously.

22.6.5 Step 5 — Disconnect

Send `EXIT` (or `QUIT`) for a graceful close. The gateway notifies the engine and shuts down the TCP connection after flushing pending output.

22.7 Command reference

All commands follow the ALF line format: `VERB|FIELD=VALUE|FIELD=VALUE\n`. Field names are **case-insensitive** — the gateway normalises names and values to uppercase before parsing. The exception is the order `ID` on `AMEND` and `CANCEL`, which is passed through unchanged, since order IDs are lowercase hexadecimal.

22.7.1 `NEW` — submit order

Single-leg order:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TAG=ORDER-001
NEW|SYM=AAPL|SIDE=BUY|TYPE=MARKET|QTY=50
NEW|SYM=AAPL|SIDE=BUY|TYPE=STOP|QTY=100|STOP=148.00
NEW|SYM=AAPL|SIDE=BUY|TYPE=STOP_LIMIT|QTY=100|STOP=148.00|PRICE=147.50
NEW|SYM=AAPL|SIDE=BUY|TYPE=FOK|QTY=100|PRICE=150.00
NEW|SYM=AAPL|SIDE=BUY|TYPE=IOC|QTY=100|PRICE=150.00
NEW|SYM=AAPL|SIDE=BUY|TYPE=ICEBERG|QTY=1000|PRICE=150.00|VISIBLE=100
NEW|SYM=AAPL|SIDE=BUY|TYPE=TRAILING_STOP|QTY=100|TRAIL=0.50
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=GTC
```

OCO pair:

```
NEW|TYPE=OCO|OCO_ID=tp-sl|SYM=AAPL|QTY=100|TIF=DAY|LEG1_SIDE=SELL|LEG1_TYPE=LIMIT|LEG1_PRICE=152.00|LEG2_SIDE=SELL|LEG2_TYPE=STOP|LEG
```

Multi-leg combo:

```
NEW|TYPE=COMBO|COMBO_ID=spread-1|COMBO_TYPE=AON|TIF=DAY|LEG_COUNT=2|LEG0.SYM=AAPL|LEG0.SIDE=BUY|LEG0.QTY=100|LEG0.PRICE=150.00|LEG1.S
```

Field	NEW (single)	Notes
`SYM`	required	Instrument symbol
`SIDE`	required	`BUY` or `SELL`
`TYPE`	required	`LIMIT`, `MARKET`, `STOP`, `STOP_LIMIT`, `FOK`, `IOC`, `ICEBERG`, `TRAILING_STOP`, `OCO`, `COMBO`
`QTY`	required	Positive integer
`PRICE`	conditional	Required for `LIMIT`, `FOK`, `IOC`, `ICEBERG`, `STOP_LIMIT`
`STOP`	conditional	Required for `STOP`, `STOP_LIMIT`

Field	NEW (single)	Notes
VISIBLE	conditional	Required for ICEBERG; must be < QTY
TRAIL	conditional	Required for TRAILING_STOP
TIF	optional	DAY (default), GTC, ATO, ATC
SMP	optional	NONE (default), CANCEL_AGGRESSOR, CANCEL_RESTING, CANCEL_BOTH
TAG	optional	Client order tag, max 64 chars, characters limited to A-Z0-9-_.; echoed on order lifecycle responses

Responses: ACK|ORDER_ID=...|ACCEPTED=TRUE|... or ACK|ORDER_ID=...|ACCEPTED=FALSE|REJECT_CODE=...|REASON=... followed asynchronously by FILL|..., CANCELLED|..., or EXPIRED|....

FILL includes TRADE_IDS, a comma-separated list of the durable public trade IDs that composed the fill. It normally contains one ID; a swept order can contain several in execution order. An empty value means no public trade ID was available for that fill.

FILL also includes LIQUIDITY=<MAKER|TAKER>: MAKER if this order was resting on the book and got matched into, TAKER if it crossed the spread and matched immediately. It is the same maker/taker attribution the drop-copy DC_FILL line has always carried, now on the ordinary session fill too — no need to enable DC or cross-reference the drop-copy feed just to know which side of a trade you were on.

FILL also names the order's SYMBOL and SIDE, so a client can book a fill into the right position even for an order it is not (or no longer) tracking — an OCO leg that fills before its OCO_ACK arrives, or an order placed before a reconnect.

On rejected order ACKs, REJECT_CODE is the stable machine-readable rejection classification. REASON remains the human-readable explanation. Gateway-local validation errors use ERR|CODE=...|REJECT_CODE=...|DETAIL=... and echo TAG=... when the rejected command carried a correlating tag.

PRICE, STOP and TRAIL must be on the tick grid

Every price must be an exact multiple of the instrument's tick size — SYMBOLS reports it per symbol as TICK. One that is not is rejected by the gateway before it reaches the engine:

```
ERR|CODE=INVALID_VALUE|REJECT_CODE=TICK_VIOLATION|DETAIL=PRICE: 100.005 is not a multiple of AAPL's tick size 0.01|TAG=ORDER-001
```

The gateway refuses rather than rounds, because rounding would rest your order at a price you never sent and never tell you. Ordinary floating-point arithmetic is safe: a price your code computed as 100.00 - 3 * 0.01 is accepted as 99.97. SYMBOLS_NOT_READY covers the startup window — the gateway will not price an order for a symbol whose tick precision it has not yet received from the engine, because converting against a default precision would move the price. Retry once the symbol list has arrived.

22.7.2 AMEND — amend resting order

```
AMEND|ID=<order-id>|PRICE=151.00[|RTAG=<request-tag>]
AMEND|ID=<order-id>|QTY=200[|RTAG=<request-tag>]
AMEND|ID=<order-id>|PRICE=151.00|QTY=200[|RTAG=<request-tag>]
```

At least one of PRICE or QTY is required. RTAG is optional and identifies this amend request; when present, the gateway echoes it on AMENDED or the rejected ACK.

Response: AMENDED|ORDER_ID=...|PRICE=...|QTY=...|REMAINING=...|PRIORITY_RESET=TRUE|FALSE[|RTAG=...]

22.7.3 `CANCEL` — cancel order / OCO / combo

```
CANCEL|ID=<order-id>[|RTAG=<request-tag>]
CANCEL|OCO_ID=<oco-id>
CANCEL|COMBO_ID=<combo-id>
```

`RTAG` is accepted on single-order cancels only. It identifies this cancel request and is echoed on `CANCELLED` or the rejected `ACK`.

Response for single order: `CANCELLED|ORDER_ID=...[|RTAG=...]` **Response for OCO:** `OCO_CANCELLED|OCO_ID=...|CANCELLED_ID=`
Response for combo: `COMBO_STATUS|COMBO_ID=...|STATUS=CANCELLED|REASON=...`

22.7.4 `QUOTE` — submit/replace two-sided quote (MARKET_MAKER role)

```
QUOTE|SYM=AAPL|BID=150.00|ASK=150.10|BID_QTY=500|ASK_QTY=500
QUOTE|SYM=AAPL|BID=150.00|ASK=150.10|BID_QTY=500|ASK_QTY=500|QUOTE_ID=my-q1
```

Requires gateway role `MARKET_MAKER`. `BID` must be strictly less than `ASK`.

Response: `QUOTE_ACK|QUOTE_ID=...|SYM=AAPL|ACCEPTED=TRUE|BID_ID=...|ASK_ID=...`

22.7.5 `QUOTE_CANCEL` — cancel active quote (MARKET_MAKER role)

```
QUOTE_CANCEL|SYM=AAPL
```

Response: `QUOTE_ACK|QUOTE_ID=...|SYM=AAPL|ACCEPTED=TRUE|...`

22.7.6 `KILL` — gateway kill-switch

```
KILL
KILL|SYM=AAPL
```

Cancels all resting orders and active quotes for this gateway, optionally scoped to one symbol.

Response: `KILL_ACK|ACCEPTED=TRUE|ORDERS=N|QUOTES=N`

22.7.7 `DC` — toggle drop-copy relay

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

Subscribes (or unsubscribes) this session to the engine's drop-copy feed (`DropCopyPublisher`, ZMQ PUB `:5557` — a separate socket from the main event bus on `:5556`, see Drop Copy), scoped to this session's own `gateway_id`. Unlike CALF/RALF, the drop-copy feed has no handshake or role model of its own — `pm-alf-gwy` maintains a single refcounted subscription per gateway ID on its drop-copy SUB socket and fans it out only to the session(s) that requested it.

Disabled by default per session. There is no gateway-wide config flag to force it on — each connecting client opts in independently with `DC|STATE=ON`, mirroring how a real drop-copy relay is provisioned per participant, not per venue.

Response: `DC_ACK|STATE=ON` or `DC_ACK|STATE=OFF`

Live events (while enabled): `DC_FILL|SEQ=..|ORDER_ID=..|SYMBOL=..|FILL_QTY=..|FILL_PRICE=..|LIQUIDITY=..` — one per fill, delivered asynchronously in addition to (not instead of) the ordinary `FILL` message. `SEQ` and `LIQUIDITY` (`MAKER`/`TAKER`) come from the drop-copy envelope and are not present on `FILL`; `DC_FILL` does not carry `REMAINING`/`STATUS` the way `FILL` does. See Drop Copy — order.fill event for the full source payload.

```
> 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
```

22.7.8 `SYMBOLS` — instrument list

```
SYMBOLS
```

Multi-line response:

```
SYMBOLS|COUNT=3
SYMBOL|SYM=AAPL|TICK=0.01
SYMBOL|SYM=MSFT|TICK=0.01
SYMBOL|SYM=TSLA|TICK=0.01
END|TYPE=SYMBOLS
```

22.7.9 `ORDERS` — resting order list

```
ORDERS
```

Multi-line response:

```
ORDERS|COUNT=2|GW=TRADER01
ORDER|ID=abc123|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|REMAINING=60|PRICE=150.00|STATUS=PARTIAL
ORDER|ID=def456|SYM=MSFT|SIDE=SELL|TYPE=LIMIT|QTY=200|REMAINING=200|PRICE=415.00|STATUS=NEW
END|TYPE=ORDERS
```

22.7.10 `QBOOT` — quote bootstrap state

```
QBOOT
QBOOT|SYM=AAPL
```

Multi-line response:

```
QBOOT|COUNT=1
QUOTE|QUOTE_ID=...|SYM=AAPL|BID=150.00|ASK=150.10|BID_QTY=500|ASK_QTY=500|STATUS=ACTIVE
END|TYPE=QBOOT
```

22.7.11 `QLEGS` — quote leg snapshot (active + recent)

```
QLEGS
QLEGS|SYM=AAPL
QLEGS|SHOW=RECENT
QLEGS|SYM=AAPL|SHOW=ALL
```

Field	Required	Default	Description
`SYM`	No	all symbols	Restrict output to one symbol
`SHOW`	No	`ACTIVE`	`ACTIVE` = currently live legs, `RECENT` = recently-inactivated quotes, `ALL` = both

`pm-alf-gwy` forwards this straight to the engine's `system.quote_legs_request` message and renders the reply. `ACTIVE` legs (`LEG` lines) carry each leg's live price/qty/remaining/status. `RECENT` rows (`RECENT_LEG` lines) are quote-level summaries drawn from the engine's bounded, in-memory, per-gateway history of recently-inactivated quotes. Each `RECENT_LEG` line is optionally followed by `RECENT_BID_LEG` and/or `RECENT_ASK_LEG` lines carrying that leg's final price/qty/remaining/filled/status snapshot at the moment it was cancelled — these are emitted only when the engine had that leg's final order state available at removal time, which is the common case for every normal inactivation path (see `system.quote_legs_request` for when a leg's snapshot can be absent). See ALF Console → QLEGS for the column semantics, which `pm-alf-console` and `pm-alf-gwy` now share.

PRICE on leg lines

`PRICE` is display money — the leg's limit price, the same number the `QUOTE` command was given. A quote leg is always a priced limit order, so it is normally present; an empty value means the engine had no price for that order, not that the leg was unpriced.

On `RECENT_BID_LEG`/`RECENT_ASK_LEG` it is the price **as it stood at removal**. The engine records it at that moment because once an order leaves the book nothing in the engine can recover its price, which is the same reason `QTY`/`REMAINING`/`FILLED` on those lines are final values rather than live ones.

Multi-line response:

```
QLEGS|COUNT=2|RECENT_COUNT=1|SHOW=ALL
LEG|QUOTE_ID=Q123|SYM=AAPL|SIDE=BUY|ORDER_ID=7c4a91e2|PRICE=150.00|QTY=500|REMAINING=400|FILLED=100|STATUS=PARTIAL|QUOTE_STATUS=ACTIV
LEG|QUOTE_ID=Q123|SYM=AAPL|SIDE=SELL|ORDER_ID=be2170fd|PRICE=150.10|QTY=500|REMAINING=500|FILLED=0|STATUS=RESTING|QUOTE_STATUS=ACTIVE
RECENT_LEG|QUOTE_ID=Q100|SYM=AAPL|QUOTE_STATUS=CANCELLED|REASON=Cancelled by participant|REMOVED_AT_NS=1784468999030221878
RECENT_BID_LEG|QUOTE_ID=Q100|SIDE=BUY|ORDER_ID=3f9a2b71|PRICE=149.90|QTY=500|REMAINING=500|FILLED=0|STATUS=CANCELLED
RECENT_ASK_LEG|QUOTE_ID=Q100|SIDE=SELL|ORDER_ID=8d1c4e05|PRICE=150.20|QTY=500|REMAINING=200|FILLED=300|STATUS=CANCELLED
END|TYPE=QLEGS
```

An unconnected/unknown gateway still gets a well-formed, empty reply (`QLEGS|COUNT=0|RECENT_COUNT=0|SHOW=...` followed immediately by `END|TYPE=QLEGS`) rather than an error — `QLEGS` never fails on a bad gateway ID, it simply has nothing to report.

22.7.12 `POS|GW=<gateway_id>` — query another gateway's position

```
POS|GW=MM_AAPL_01
```

Field	Required	Default	Description
`GW`	Yes	—	The `gateway_id` to ask about — any connected gateway, not just your own

A bare `POS` (no `GW=`) is rejected with `ERR|CODE=MISSING_FIELD` (`POS requires GW=<gateway_id>`).

`POS|GW=` forwards `system.position_request` for the given `gateway_id` to the engine and renders the engine's `system.position_snapshot` reply: net quantity and average cost, per symbol, for every symbol that gateway has a non-zero position in. Unlike every other command on this page, the reply's topic is scoped to the *queried* `gateway_id`, not to your own session's gateway_id, so `pm-alf-gwy` routes it back to every session with a query outstanding for that `gateway_id`. A session may have queries for several gateways outstanding at once; each is answered with its own `POSITION` block. `GW` must be a valid gateway ID (a letter, then up to 31 letters, digits or _), or the query is refused with `INVALID_VALUE`.

Multi-line response:

```
POSITION|GW=MM_AAPL_01|COUNT=1
POS_ENTRY|SYM=AAPL|NET_QTY=-300|AVG_COST=150.05
END|TYPE=POSITION
```

A flat gateway (or one with no position in any symbol) still gets a well-formed, empty reply (`POSITION|GW=<gateway_id>|COUNT=0` followed immediately by `END|TYPE=POSITION`) rather than an error — same convention as `QLEGS` and `QBOOT` for an unconnected/unknown/flat gateway.

22.7.13 `SESSION` — query current trading session state

```
SESSION
```

Requests the engine's current trading session state (`PRE_OPEN`, `OPENING_AUCTION`, `CONTINUOUS`, `CLOSING_AUCTION`, `CLOSED`) for this gateway. `pm-alf-gwy` forwards this straight to the engine's `system.session_state_request` message and routes the reply back to the querying session only.

Response:

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

This reuses the same `SESSION` message type as the unsolicited session-state broadcast below — a queried response is simply a `SESSION` line with `PREV_STATE` left empty (a query has no "previous state" to report) and an additional `SESSIONS_ENABLED` field indicating whether session-state gating is active for this engine. Existing clients that already parse the broadcast `SESSION` line need no changes to also handle the queried response.

22.7.14 `PING` / `EXIT`

`PING` is also your keepalive: any inbound line resets the gateway's `idle_timeout_sec` clock, so a client that may sit idle (waiting for a human, or for the market) should send `PING` every `IDLE / 3` seconds, where `IDLE` is the value announced in `WELCOME`. Both example clients do this and hide the `PONG` replies to their own pings (`--debug` logs each one sent).

```
PING        → PONG|TS=2026-07-02T09:30:00.123Z
EXIT        → (connection closed)
QUIT        → (connection closed)
```

22.8 Broadcast events

These messages arrive **unsolicited** on every authenticated session that did not log on with `FEED=ORDERS`.

Message type	Key fields	Trigger
`SESSION`	`STATE`, `PREV_STATE`	Session phase change (e.g. `CONTINUOUS`, `CLOSED`). Also sent (with `PREV_STATE` empty and an added `SESSIONS_ENABLED` field) as a direct reply to the `SESSION` command
`HALT`	`SYMBOL`, `LEVEL`	Circuit-breaker halt on a symbol
`RESUME`	`SYMBOL`, `SRC`	Circuit-breaker resume; `SRC` is the halt source that triggered the original halt
`TRADE`	`SYMBOL`, `PRICE`, `QTY`, `SIDE`	Any matched trade on any symbol
`HB`	`TS`	Periodic heartbeat when no other outbound activity

Your client does not need to subscribe to anything. Broadcast events are delivered automatically after `WELCOME`.

22.9 Engine-scoped events (per-gateway)

These messages are addressed to your gateway ID and arrive on your session only.

Message type	Key fields
`ACK`	`ORDER_ID`, `ACCEPTED`, `REASON`, `REJECT_CODE`, `SYMBOL`, `SIDE`, `TYPE`, `TAG`, `RTAG`
`FILL`	`ORDER_ID`, `FILL_QTY`, `FILL_PRICE`, `REMAINING`, `STATUS`, `SYMBOL`, `SIDE`, `TRADE_IDS`, `LIQUIDITY`, `TAG`
`AMENDED`	`ORDER_ID`, `PRICE`, `QTY`, `REMAINING`, `PRIORITY_RESET`, `TAG`, `RTAG`
`TRIGGERED`	`ORDER_ID`, `SYMBOL`, `TYPE`, `STOP`, `BECOMES`, `TRADE_ID`, `TRADE_PRICE`, `TAG` — a stop fired, sent before its `FILL`s; see TRAILING_STOP
`CANCELLED`	`ORDER_ID`, `TAG`, `RTAG`, `CANCEL_REASON` — see Unsolicited cancels
`EXPIRED`	`ORDER_ID`, `TAG`
`QUOTE_ACK`	`QUOTE_ID`, `SYM`, `ACCEPTED`, `REASON`, `BID_ID`, `ASK_ID`
`QUOTE_STATUS`	`QUOTE_ID`, `STATUS`, `REASON`
`COMBO_ACK`	`COMBO_ID`, `ACCEPTED`, `REASON`
`COMBO_STATUS`	`COMBO_ID`, `STATUS`, `REASON`
`OCO_ACK`	`OCO_ID`, `ACCEPTED`, `LEG1_ID`, `LEG2_ID`, `REASON`
`OCO_CANCELLED`	`OCO_ID`, `CANCELLED_ID`, `REASON`
`KILL_ACK`	`ACCEPTED`, `REASON`, `ORDERS`, `QUOTES`
`DC_ACK`	`STATE` (`ON`/`OFF`) — reply to `DC`, not unsolicited
`DC_FILL`	`SEQ`, `ORDER_ID`, `SYMBOL`, `FILL_QTY`, `FILL_PRICE`, `LIQUIDITY` — only while `DC|STATE=ON` is active, see `DC`

SYM on QUOTE_ACK

Quote events are addressed to a *gateway*, not to a symbol, so a client quoting several instruments would otherwise have to remember which quote it sent last and match replies by send order. That breaks the moment one reply is missed. `SYM` names the instrument on the ack itself, so no client-side bookkeeping is needed. It is empty only when the quote was rejected before its symbol was known — a malformed payload, or a missing `SYM=` on the request.

`QUOTE_ID` remains the correlation key for the *specific quote*: supply your own `QUOTE_ID=` on the `QUOTE` command and it is echoed here, on `QUOTE_STATUS`, and stamped on both leg orders. If you omit it the engine generates one.

Every one of these carries `TAG` when the order was submitted with one, so a client correlates an event to its own order without matching on arrival order. That includes events you did not ask for — an expiry hours after you last touched the order, or a cancel the exchange decided on.

22.9.1 Unsolicited cancels

A `CANCELLED` line does not always answer a `CANCEL` you sent. The exchange cancels orders on its own: self-match prevention, the unfillable remainder of a `MARKET` or `IOC` order, a kill switch, an OCO sibling, a `DAY` order at the close. Two fields tell you which happened:

Field	Meaning
`RTAG`	Present only when *you* asked, echoing the `RTAG` from your `CANCEL`. Absent means the exchange decided
`CANCEL_REASON`	Why the exchange decided. Absent on your own cancels

```
CANCELLED|ORDER_ID=ORD-7f3a|TAG=ORDER-001|CANCEL_REASON=SELF_MATCH_PREVENTED
```

`CANCEL_REASON`	Meaning
`SELF_MATCH_PREVENTED`	The order would have traded against another order from your own gateway. Which order is cancelled — the aggressor, the resting one, or both — follows the `SMP` action in force
`INSUFFICIENT_LIQUIDITY`	A `MARKET` or `IOC` order ran out of book. Whatever traded is reported by the preceding `FILL` lines; this cancels the remainder, which never rests
`KILL_SWITCH`	A gateway kill switch cancelled this order
`CIRCUIT_BREAKER_HALT`	Cancelled as part of a circuit-breaker halt
`GATEWAY_DISCONNECT`	Cancelled because the owning gateway session disconnected
`ADMIN_CANCEL_SYMBOL`	An administrator cancelled all resting orders on this symbol
`QUOTE_REPLACED`	A market-maker quote leg was superseded by a newer quote
`QUOTE_LEG_FILLED`	The sibling leg of a two-sided quote was auto-cancelled because this leg filled

The field is **omitted, not empty**, when you requested the cancel yourself. Treat a value you do not recognise the same way `RTAG`/`CANCEL_REASON` both absent is treated: new members may be added, but existing ones are never removed or renamed.

`CANCEL_REASON` deliberately does not share the `REJECT_CODE` vocabulary. A cancel is not a rejection — the order was accepted, and may well have traded.

22.10 Error codes

Every error arrives as `ERR|CODE=<CODE>|REJECT_CODE=<canonical-code>|DETAIL=<message>`. When the rejected command carried `TAG` or `RTAG`, the gateway also echoes it as `TAG` so the client can correlate the error with the submitted line.

Code	When it occurs	Connection kept?
AUTH_REQUIRED	Any command before HELLO completes	No — closed immediately
AUTH_TIMEOUT	No HELLO received within handshake_timeout_sec (default 10s) of connecting	No
AUTH_FAILED	Engine rejected the gateway ID	No
PROTO_MISMATCH	HELLO with wrong PROTO value	No
GATEWAY_ALREADY_CONNECTED	Same gateway ID already has an active session	No
HELLO_ALREADY_PENDING	A second HELLO arrives while an earlier one is still awaiting the engine's auth result	Yes
BAD_MESSAGE	Empty line, non-UTF-8, or line > 4096 bytes	Yes
UNKNOWN_COMMAND	Unrecognised command verb	Yes
MISSING_FIELD	Required field absent	Yes
INVALID_VALUE	Field value fails validation (e.g. PRICE=NaN, or a price off the instrument's tick grid — the latter carries REJECT_CODE=TICK_VIOLATION)	Yes
SYMBOLS_NOT_READY	Any command naming a symbol before the engine's symbol list has reached the gateway (REJECT_CODE=SYMBOL_NOT_READY) — transient, retry	Yes
SYMBOL_NOT_CONFIGURED	Unknown symbol (after symbols are loaded)	Yes
ROLE_DENIED	Command not allowed for this gateway's role (e.g. QUOTE for non-MM)	Yes
RATE_LIMITED	Commands arriving faster than max_commands_per_second	Yes
SLOW_CLIENT	Outbound queue full (max_client_queue). Queued TRADE prints and heartbeats are dropped; the client's own events are still sent ahead of this line	No
GATEWAY_SHUTDOWN	The gateway is stopping. Sent after the session's remaining events (flushed for up to 2 seconds)	No
IDLE_TIMEOUT	No inbound traffic for idle_timeout_sec	No
MAX_ERRORS	Too many errors in the sliding error window	No
SESSION_LOST	The engine no longer holds the session: it restarted, or it expired the session. The gateway closes every session when it sees the engine's startup broadcast, and every 5 seconds asks the engine which participants it holds and closes the ones it no longer does. Log on again (after an engine restart the engine's positions start flat)	No
EVENTS_LOST	The engine's event stream dropped events meant for this session (the gateway fell behind and the bus discarded them). Unsolicited and untagged; TOPIC names the event family (e.g. ORDER.FILL) and COUNT how many were missed. Reconcile with ORDERS. Counted in gwy.events_lost	Yes

Code	When it occurs	Connection kept?
`ENGINE_UNAVAILABLE`	Command could not be forwarded to the engine (engine unreachable); retry shortly	Yes
`INTERNAL_ERROR`	Unexpected gateway-internal exception	Yes

Error escalation

If a client accumulates `max_errors_before_disconnect` (default 50) errors in the `error_window_sec` sliding window, the gateway disconnects it. This protects the gateway from runaway or malicious clients.

22.11 Example libraries and interactive clients

The `docs/examples/alf/` directory contains ready-to-run Python and C libraries that replicate the workflow of `pm-alf-console` as an **external TCP client** — no ZeroMQ, no `edumatcher` package import, only a plain socket.

```
docs/examples/alf/
├── python/
│   ├── alf_parser.py       # Protocol library: parse, build, AlfSession
│   └── alf_client.py       # Interactive client (tab-completion, event display, P&L, keepalive PING)
└── c/
    ├── alf_parser.h / .c   # C library
    ├── alf_client.c        # Interactive C client (readline + select)
    └── Makefile
```

22.11.1 Python

Library — `alf_parser.py`

```
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", ...}

# Build a line to send
line = 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"
```

Response values preserve case; use the complete order ID unchanged in `AMEND`/`CANCEL`. ALF prices remain decimal display prices, not internal ticks. For a complete `AlfSession` query loop with keepalives and cleanup, see Building your own application below.

Interactive client — `alf_client.py`

```
cd docs/examples/alf/python

# Connect to a local gateway
python3 alf_client.py --id TRADER01

# Connect to a remote gateway
python3 alf_client.py --host 10.0.0.5 --port 5565 --id TRADER01
```

At the prompt the client behaves like `pm-alf-console`: Tab completes command verbs, field names, and enum values. Background receive thread displays fills, acks, and broadcast events while you type. `POS` shows positions and realized P&L from fills seen by this connection; `POS|GW=TRADER01` queries the engine's position snapshot. `STATUS` is local session info. Both interactive clients send keepalive `PING` automatically. `DC|STATE=ON` enables drop-copy display; `DC_FILL` is not booked into local positions again. Fill displays include `LIQUIDITY` and `TRADE_IDS` when supplied. History is saved to `~/.alf_client_history`.

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TAG=ORDER-001
[09:30:01.234] ACK      xxxxxxxx tag=ORDER-001  order accepted
[TRADER01]> ORDERS
[TRADER01]> POS
[TRADER01]> HELP
[TRADER01]> EXIT
```

22.11.2 C

Build:

```
# macOS: brew install readline (Homebrew readline for full callback support)
# Linux: sudo apt install libreadline-dev

cd docs/examples/alf/c
make
```

Run:

```
./alf_client --id TRADER01
./alf_client --host 10.0.0.5 --port 5565 --id TRADER01
./alf_client --id TRADER01 --no-color
```

The C client uses `select()` to multiplex the TCP socket and stdin, so gateway events display immediately while you are typing. Readline provides tab completion and history.

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TAG=ORDER-002
[09:30:01.234] ACK      xxxxxxxx tag=ORDER-002  order accepted
[TRADER01]> ORDERS
[TRADER01]> POS
[TRADER01]> HELP
[TRADER01]> EXIT
```

Library usage:

```
#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;
}
const char *accepted = alf_get_field(&msg, "ACCEPTED");
if (accepted) puts(accepted);          /* "TRUE" */

/* Build */
const char *kv[] = {"SYM", "AAPL", "SIDE", "BUY",
                    "TYPE", "LIMIT", "QTY", "100", "PRICE", "150.00", "TAG", "ORDER-003", 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;
}
```

This fragment belongs inside your application's function; include `<stdio.h>`. Send all `len` bytes with a partial-write loop such as the client's `gwy_send`. The parser returns negative errors for invalid/over-capacity input; field lookups can return `NULL`.

22.11.3 Building your own application

Copy `python/alf_parser.py` into a Python project and import `AlfSession`. It uses only the standard library: no engine package, ZeroMQ, or readline is required. Reuse its validated, timeout-bounded handshake and buffered line reader instead of maintaining a second ad-hoc socket implementation. After connecting, reads are blocking unless your application sets a timeout; the helper does not schedule keepalives or reconnect automatically.

For C, copy `alf_parser.h` and `alf_parser.c` and compile them with your application. Readline is required only by the interactive client, not the parser. Adapt `do_handshake`, `gwy_send`, `process_socket_data`, and the `select()` loop in `alf_client.c` for transport and lifecycle handling; they are internal example functions, not exported C library APIs.

The application integration walkthrough provides a runnable Python query and a C integration checklist. It explains matching multiline query terminators, unsolicited events, periodic `PING` even while receiving broadcasts, request correlation, explicit failures, and reconciling orders/positions after reconnect. The C demo has fixed capacities; review them before adapting it to a larger workload.

TCP is a byte stream

Never assume one `recv()` equals one line. Always buffer and split on `\n`.

22.12 When to use `pm-alf-gwy` vs. the alternatives

Scenario	Best choice
Human operator on the same machine	`pm-alf-console` (tab completion, history, P&L display)
External bot in Python / any language on the same or a remote host	`pm-alf-gwy`
Browser UI / REST-native stack	`pm-api-gwy`
Read-only market-data consumer	`pm-md-gwy` (CALF)
Post-trade / clearing / audit consumer	`pm-ralf-gwy` (RALF)

22.13 Troubleshooting

22.13.1 Check whether the port is in use

Before starting `pm-alf-gwy`, or when a client cannot connect, verify that something is actually listening on port 5565.

macOS:

```
# lsof — shows the process name and PID holding the port
sudo lsof -iTCP:5565 -sTCP:LISTEN

# BSD netstat (ships with macOS)
netstat -an | grep LISTEN | grep 5565
```

Linux:

```
# ss — preferred on modern Linux
ss -tlnp 'sport = :5565'

# lsof
sudo lsof -iTCP:5565 -sTCP:LISTEN

# netstat (older distributions)
netstat -tlnp | grep 5565
```

If no output appears, the gateway is not running or is bound to a different port. Check the `alf_gateway.port` value in `engine_config.yaml`.

22.13.2 Test the TCP connection from the command line

Use `nc` (netcat) to open a raw TCP connection and type ALF lines by hand. This bypasses any client library and proves the gateway is reachable end-to-end.

macOS / Linux:

```
nc 127.0.0.1 5565
```

For a remote host:

```
nc 10.0.0.5 5565
```

Type the following lines (press Enter after each):

```
HELLO|CLIENT=test|PROTO=ALF1|ID=TRADER01
SYMBOLS
EXIT
```

Expected output: `WELCOME|...`, then a `SYMBOLS|COUNT=N` block, then `END|TYPE=SYMBOLS`, then the connection closes.

`telnet` (macOS / Linux):

```
telnet 127.0.0.1 5565
```

Type `HELLO|CLIENT=test|PROTO=ALF1|ID=TRADER01` and press Enter. `telnet` echoes characters locally so the line appears duplicated in the terminal — the `WELCOME` response confirms the gateway accepted it. Press `Ctrl-]`, then type `quit` to close.

Non-interactive test (useful in scripts or CI):

```
printf 'HELLO|CLIENT=test|PROTO=ALF1|ID=TRADER01\nEXIT\n' | nc 127.0.0.1 5565
```

Expected output ends with `BYE` or a clean connection close immediately after `WELCOME`.

22.13.3 Common problems

Symptom	Likely cause	Fix
`Connection refused`	Gateway not started or wrong port	Confirm `pm-alf-gwy` is running; check `alf_gateway.port` in config
Connection hangs with no output	Firewall blocking port 5565	Test on loopback (`127.0.0.1`) first; open port in firewall for remote access
`ERR|CODE=AUTH_REQUIRED` immediately	First line was not `HELLO`	Ensure the very first line is a valid `HELLO`
`ERR|CODE=AUTH_FAILED`	Gateway ID not in `participants`	Add the ID under `participants` in `engine_config.yaml` and restart engine
`ERR|CODE=PROTO_MISMATCH`	`PROTO` field value is not `ALF1`	Fix the `HELLO` line: `HELLO|CLIENT=...|PROTO=ALF1|ID=...`
`ERR|CODE=GATEWAY_ALREADY_CONNECTED`	Same gateway ID connected elsewhere	Disconnect the other session, or use a different gateway ID
`ERR|CODE=AUTH_FAILED` with `Gateway already connected`	Another process (console, API gateway, another gateway) holds the ID at the engine	Stop that process. If it was killed, wait up to 3 minutes for the engine to time it out (3 missed heartbeats)
`WELCOME` arrives but then silence	Engine not running or ZMQ link lost	Start `pm-engine`; check gateway logs for ZMQ errors
Gateway closes after ~30 s of silence	`idle_timeout_sec` elapsed	Send `PING` periodically (the example clients do so every `IDLE/3` seconds); reduce `idle_timeout_sec` in config if needed
`ERR|CODE=RATE_LIMITED`	Commands arriving faster than `max_commands_per_second`	Throttle the client; increase `max_commands_per_second` in config
Gateway not reachable from another host	`bind_address: 127.0.0.1`	Change `bind_address` to `0.0.0.0` (or the specific interface IP)

22.14 See also

In this book:

- The Configuration Workflow — `alf_gateway:` section and `participants` allowlist

Participant Guide:

- ALF Console — interactive command reference for `pm-alf-console`

Reference Manual:

- Processes — process topology and ZMQ message tables

Protocols and Clients:

- ALF Protocol Reference — formal wire syntax and full field/enum definitions
- Drop Copy — the engine's drop-copy feed (`:5557`) that `DC|STATE=ON` relays
- External Protocols Overview — protocol comparison and selection guide

CHAPTER 23

BALF Gateway — Binary Order Entry

Learning objectives

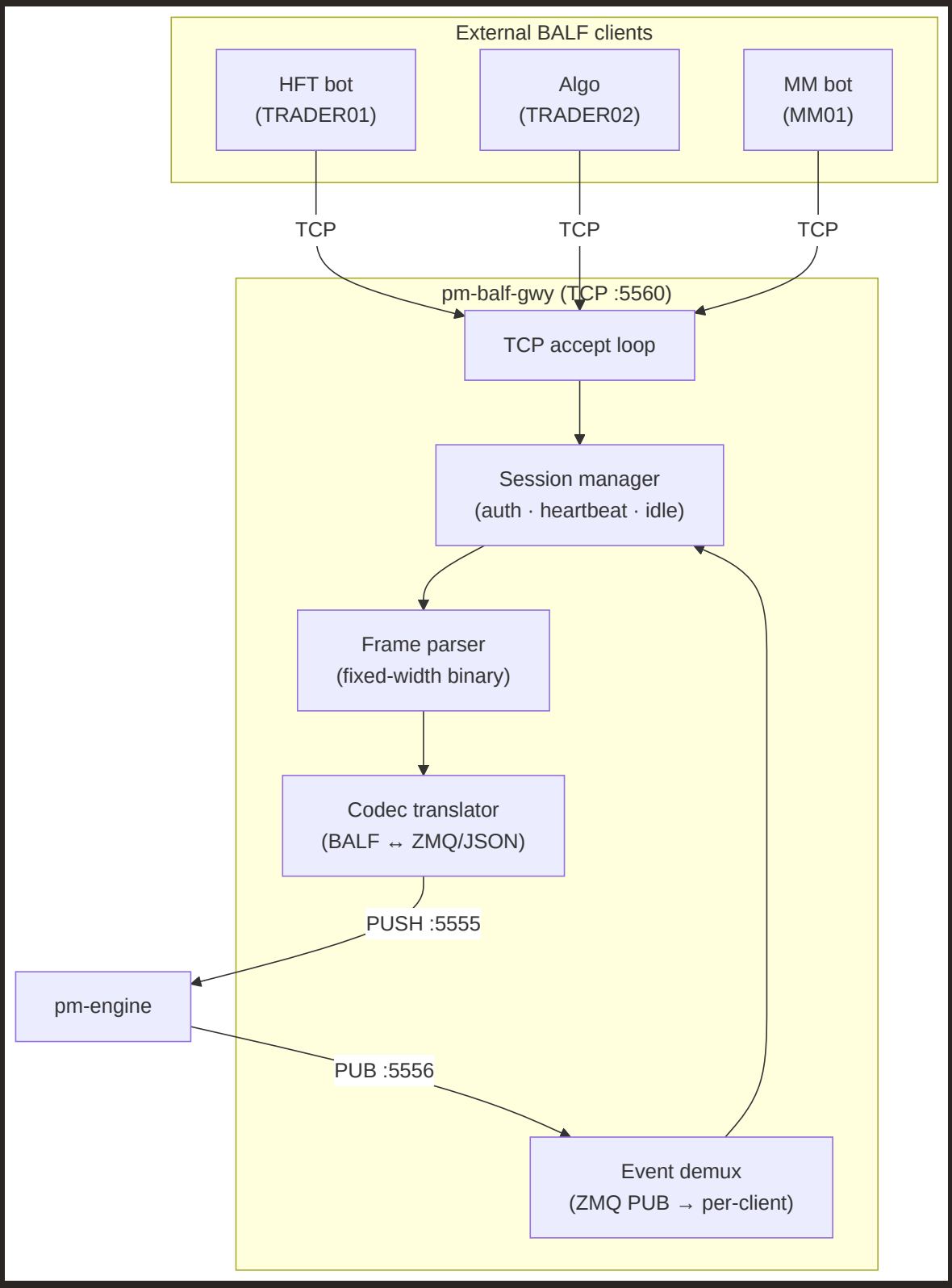
After reading this page you will understand:

- what `pm-balf-gwy` does and when to prefer it over `pm-alf-gwy`
- how to configure it in `engine_config.yaml`
- how to start it and verify connectivity from a Python script
- the session lifecycle: LOGON → LOGON_ACK → messages → LOGOUT
- which client messages are accepted and which server messages to expect
- how heartbeats and liveness timeouts work
- how to write a minimal Python BALF client
- the error conditions that close the connection

See BALF Protocol Reference for exact frame layouts, byte offsets, and encoding rules.

23.1 What this process is

`pm-balf-gwy` is the **BALF TCP gateway**. BALF (Binary ALF) is a fixed-width binary protocol that carries the same trading semantics as ALF — identical order types, TIF values, SMP rules, and gateway authentication — but is designed for low-latency programmatic clients.



23.1.1 Why BALF instead of ALF?

Property	ALF (`pm-alf-gwy`)	BALF (`pm-balf-gwy`)
Wire format	Human-readable text lines	Fixed-width binary frames
Parsing	`|`-delimiter split + float parsing	Known byte offsets, no scanning

Property	ALF (`pm-alf-gwy`)	BALF (`pm-balf-gwy`)
Frame size known upfront	No — scan for `\n`	Yes — `msg_type` byte determines size
Sequence numbers	None	Explicit `u32` per direction
Price encoding	ASCII decimal string	`i64` scaled by 10
Best for	Python scripts, operators, simple bots	Latency-sensitive algos, C/Rust/C++ clients

Same engine semantics

BALF and ALF both authenticate against the same `participants` allowlist and translate into the identical engine ZMQ/JSON messages. The engine sees no difference between an ALF order and a BALF order.

23.1.2 What BALF `1.0.0` does not support

Unsupported feature	Notes
OCO (one-cancels-other) orders	Reserved for a future BALF version
Multi-leg combo orders	Reserved for a future BALF version
Market-data broadcast (SESSION, TRADE, HALT)	ALF gateway only in `1.0.0`
Sequence recovery / resend protocol	Reconnect on gap
Encryption at the protocol level	Use TLS or a tunnel

For interactive operator use, `pm-alf-console` remains the right tool. For external market-data consumers, use `pm-md-gwy` (CALF) or `pm-api-gwy`.

23.2 Prerequisites

- `pm-engine` running and accessible.
- Gateway IDs that will connect must be present in `engine_config.yaml` under `participants` (BALF `1.0.0` reuses the ALF allowlist).
- Optional: add a `balf_gateway:` section to customise port and limits.

23.3 Configuration

Add a `balf_gateway:` section to `engine_config.yaml`:

```
balf_gateway:
  enabled: true
  name: "balf-gwy01"
  bind_address: "0.0.0.0"
  port: 5560
  heartbeat_interval_sec: 1
  heartbeat_timeout_sec: 5
  idle_timeout_sec: 30
  auth_timeout_sec: 10
  max_connections: 64
  max_client_queue: 10000
  max_messages_per_second: 100
  max_errors_before_disconnect: 10
  error_window_sec: 60
  duplicate_session_policy: REJECT_NEW
```

The gateway reads gateway roles and disconnect behaviour from the existing `participants` list — no separate credentials block is needed. Any gateway ID listed in `participants` can connect to `pm-balf-gwy`.

Field	Default	Description
`enabled`	`true`	Master switch
`name`	`balf-gwy01`	Gateway name echoed in `LOGON_ACK.msg`
`bind_address`	`0.0.0.0`	Network interface to listen on (`127.0.0.1` for local-only)
`port`	`5560`	TCP listen port
`heartbeat_interval_sec`	`1`	Seconds between server-initiated `HEARTBEAT` frames
`heartbeat_timeout_sec`	`5`	Disconnect if no inbound traffic for this many seconds
`idle_timeout_sec`	`30`	Accepted but currently unused by the gateway runtime (only `heartbeat_timeout_sec` drives idle-session disconnects); reserved for future use
`auth_timeout_sec`	`10`	Hard-close if `LOGON` is not answered within this window
`max_connections`	`64`	Maximum simultaneous TCP connections
`max_client_queue`	`10000`	Per-client outbound frame buffer capacity
`max_messages_per_second`	`100`	Token-bucket rate limit per client
`max_errors_before_disconnect`	`10`	Error threshold in a sliding window before forced disconnect
`error_window_sec`	`60`	Sliding window length for the error counter
`duplicate_session_policy`	`REJECT_NEW`	`REJECT_NEW` or `EVICT_OLD`

TLS

`pm-balf-gwy` does not terminate TLS. For remote deployments, place it behind a reverse proxy (nginx, stunnel, or similar).

23.4 Start the gateway

Installed mode:

```
pm-engine --verbose
pm-balf-gwy
```

Developer mode:

```
poetry run pm-engine --verbose
poetry run pm-balf-gwy
```

CLI override options:

Option	Default	Description
`--bind ADDR`	from config / `0.0.0.0`	Override TCP bind address
`--port PORT`	from config / `5560`	Override TCP listen port
`--engine-host HOST`	from config	Override engine host (sets `tcp://HOST:5555` and `tcp://HOST:5556`)
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)

Option	Default	Description
`-q` / `--quiet`	off	Reduce output to warnings/errors — currently a no-op: the default log level is already `WARNING`, so `-q` changes nothing observable (a codebase-wide quirk, not specific to this gateway)
`--version`	—	Print the installed `pm-balf-gwy` version and exit
`--log-target`	auto	Where operational log records go: `server` (default, auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file PATH`	—	Operational log file path — required when `--log-target file`
`--log-failover-timeout SECONDS`	`30`	Grace window before falling back to a local log file once `pm-log-srv` becomes unreachable

Config file location

The engine configuration is read from `<EDUMATCHER_DATA_DIR>/ref_data/engine_config.json`. There is no `--config` flag: every process reads that one file, so none of them can be started against a configuration the others have not seen. Install one with `pm-config-deploy`.

23.5 Quick connectivity test

BALF is a binary protocol, so `nc` and `telnet` cannot be used to type frames manually. The minimal test is a short Python script:

```
#!/usr/bin/env python3
"""Minimal BALF session test — sends LOGON and prints the LOGON_ACK result."""
import socket
import struct

BALF_MAGIC   = 0xBA
BALF_VERSION = 0x01
MSG_LOGON     = 0x01
MSG_LOGON_ACK = 0x02
FRAME_SIZE = {MSG_LOGON: 32, MSG_LOGON_ACK: 92}

def recv_exact(sock, n):
    buf = bytearray()
    while len(buf) < n:
        chunk = sock.recv(n - len(buf))
        if not chunk:
            raise RuntimeError("connection closed")
        buf.extend(chunk)
    return bytes(buf)

def build_logon(gateway_id: str) -> bytes:
    header = struct.pack("<BBBBI", BALF_MAGIC, BALF_VERSION, MSG_LOGON, 0, 0)
    body   = struct.pack("<16sB7s", gateway_id.encode("ascii"), BALF_VERSION, b"\x00" * 7)
    return header + body

def parse_logon_ack(frame: bytes):
    body = frame[8:]                  # skip 8-byte header
    gw_raw, accepted, reject_code, msg_len, _, msg = struct.unpack("<16sBBBB64s", body)
    gw_id = gw_raw.rstrip(b"\x00").decode("ascii")
    reason = msg[:msg_len].decode("ascii", errors="replace")
    return gw_id, bool(accepted), reject_code, reason

sock = socket.create_connection(("127.0.0.1", 5560), timeout=5)
sock.sendall(build_logon("TRADER01"))

ack_bytes = recv_exact(sock, FRAME_SIZE[MSG_LOGON_ACK])
gw_id, ok, code, msg = parse_logon_ack(ack_bytes)

if ok:
    print(f"LOGON_ACK accepted — gateway={gw_id!r}  msg={msg!r}")
else:
    print(f"LOGON_ACK rejected — code=0x{code:02X}  msg={msg!r}")

sock.close()
```

Expected output on success:

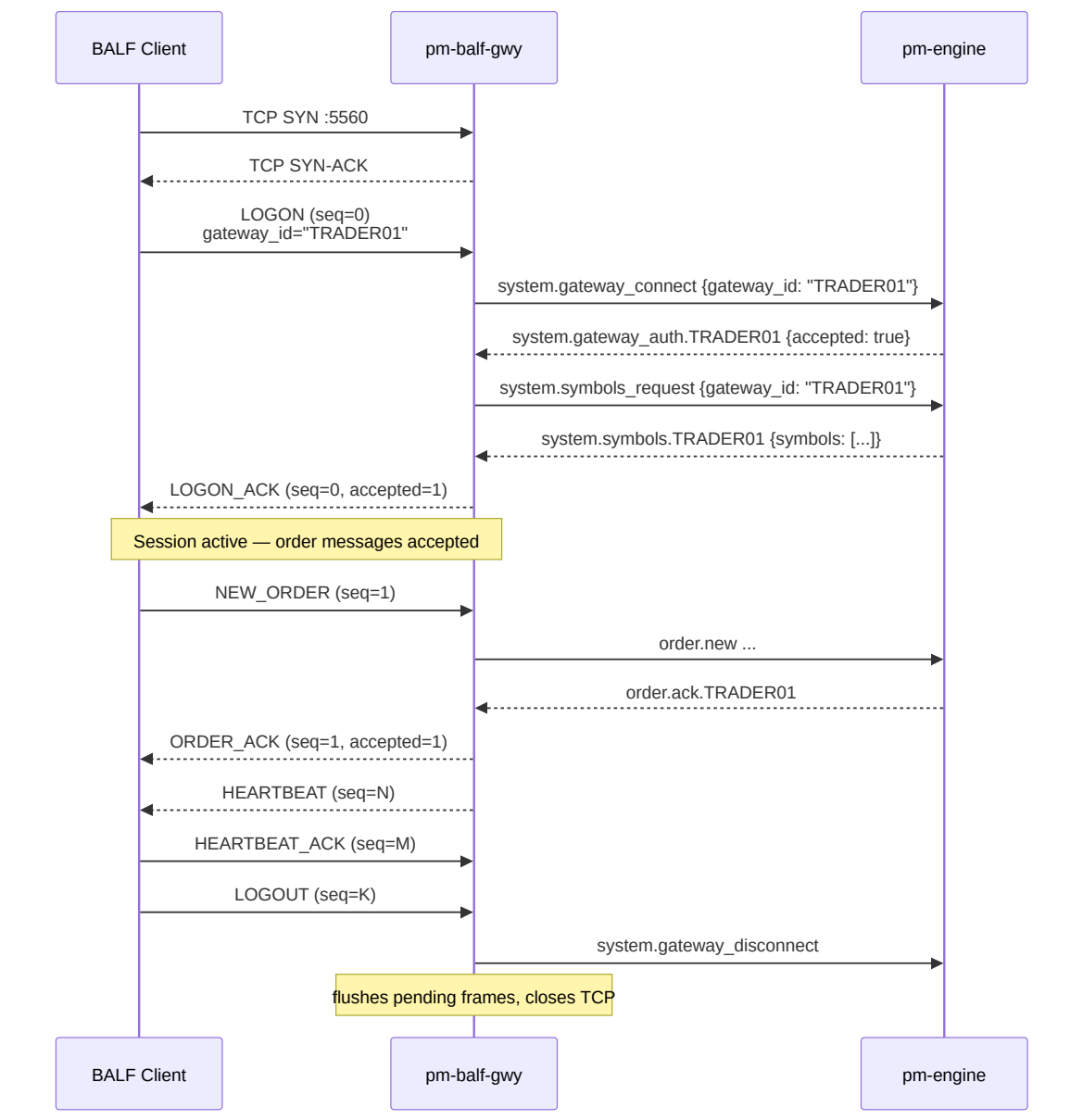
```
LOGON_ACK accepted — gateway='TRADER01'  msg='gateway=balf-gwy01 hbint=1s'
```

TCP is a byte stream

Never assume one `recv()` call delivers a complete frame. Always read exactly the expected byte count using a loop, as shown above.

23.6 Session lifecycle

Every BALF session follows this exact sequence:



23.6.1 Step 1 — Send LOGON

The **first frame** must be a `LOGON` (msg_type `0x01`, 32 bytes total):

- `gateway_id`: 16-byte zero-padded ASCII, matching an entry in `participants`
- `proto_version`: must be `1`
- reserved bytes: must be zero

If the first frame is not a valid `LOGON`, the gateway hard-closes the TCP connection immediately.

gateway_id and symbol are silently uppercased

The gateway uppercases both `gateway_id` (decoded from `LOGON`) and `symbol` (decoded from `NEW_ORDER`) after reading them off the wire, so a lowercase value is normalized rather than rejected or matched case-sensitively.
See LOGON frame layout in the protocol reference.

23.6.2 Step 2 — Receive LOGON_ACK

The gateway authenticates with the engine, asks it for the symbol list, and only then replies with `LOGON_ACK` (msg_type `0x02`, 92 bytes). The accepted ack is deliberately withheld until the symbol metadata is in: a price is meaningless without its symbol's tick scale, so no order may be accepted before the gateway knows it.

- `accepted = 1`: session is open; order messages may be sent
- `accepted = 0`: session rejected; close the TCP connection

Reject code	Meaning
`0x01`	Gateway ID not configured in engine
`0x02`	Gateway ID already connected
`0x03`	Protocol version mismatch
`0xFF`	Other — see `msg` field

`0xFF` with `msg = SYMBOLS_UNAVAILABLE` means the engine accepted the connection but never answered `system.symbols_request` within `auth_timeout_sec`. Retry the logon; if it repeats, the engine is unhealthy.

The `msg` field carries a human-readable description on success (e.g. the gateway name and configured heartbeat interval) and the engine rejection reason on failure.

See LOGON_ACK frame layout for the exact body structure.

23.6.3 Step 3 — Send order messages

After `accepted = 1`, the client may send `NEW_ORDER`, `CANCEL_ORDER`, `AMEND_ORDER`, `HEARTBEAT`, `HEARTBEAT_ACK`, and `LOGOUT`.

Sequence numbers start at `1` for the first non-LOGON message in each direction.

23.6.4 Step 4 — Receive server messages

Server-initiated messages arrive asynchronously and out-of-band relative to client messages. The client must always be able to read them. See Server-initiated messages below.

23.6.5 Step 5 — Disconnect

Send `LOGOUT` (msg_type `0x40`, 8 bytes, header only) for a graceful close. The gateway notifies the engine and closes the TCP connection after flushing any pending outbound frames.

23.7 Message reference

See the BALF Protocol Reference appendix for exact byte layouts, offset tables, and encoding rules for all messages.

23.7.1 Client → server messages

Code	Name	Bytes	Purpose
`0x01`	`LOGON`	32	Open the session; must be the first frame
`0x10`	`NEW_ORDER`	60	Submit a new single-leg order
`0x12`	`CANCEL_ORDER`	24	Cancel a resting order
`0x14`	`AMEND_ORDER`	44	Amend price and/or quantity of a resting order
`0x30`	`HEARTBEAT`	16	Liveness probe; carries `send_time_ns`
`0x31`	`HEARTBEAT_ACK`	16	Reply to a server `HEARTBEAT`
`0x40`	`LOGOUT`	8	Graceful disconnect

23.7.2 Server → client messages

Code	Name	Bytes	Trigger
0x02	LOGON_ACK	92	Response to LOGON
0x11	ORDER_ACK	60	Response to NEW_ORDER; accepted or rejected
0x13	CANCEL_ACK	32	Cancel confirmed, rejected, or system-originated
0x15	AMEND_ACK	48	Amend confirmed or rejected
0x20	EXECUTION_REPORT	64	Partial or full fill
0x30	HEARTBEAT	16	Server-initiated liveness probe
0x31	HEARTBEAT_ACK	16	Reply to a client HEARTBEAT

Reading frame size

All frame sizes are fixed and known from the header `msg_type` byte. Read the 8-byte header first, use `msg_type` to look up the total frame size, then read exactly that many additional bytes. This is all that a BALF receive loop needs to do. See Message reference summary table.

23.8 Order types and field encoding

23.8.1 Supported order types

order_type 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

See NEW_ORDER message layout for TIF, SMP, and all field offset tables.

23.8.2 Price encoding

All prices and price-like offsets are `i64`, little-endian, scaled by 10 :

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

  $150.25   →  15_025_000_000
  $0.0001   →          10_000
  $0.00     →               0  (field not used for this order type)
```

See Price encoding in the protocol reference.

23.8.3 ORDER_ACK reject codes

When `ORDER_ACK.accepted = 0`, the `reject_code` byte classifies the rejection.

Code	Meaning
`0x01`	Gateway not configured
`0x02`	Gateway not connected
`0x03`	Symbol not configured
`0x04`	Market is closed
`0x05`	ATO order outside opening auction
`0x06`	ATC order outside closing auction
`0x07`	Halt rejection (MARKET / FOK / IOC while the symbol is halted, by a circuit breaker or by an administrator — BALF has one code for both)
`0x08`	Session-phase rejection
`0x09`	Trailing stop — no prior trade price and no explicit STOP=
`0x0A`	Insufficient liquidity
`0x0B`	Price collar rejection
`0x0C`	Invalid field / gateway validation failure
`0xFF`	Other — read the `reason` field for the engine detail text

The `reason` field (25 bytes, ASCII) always contains the authoritative engine text, regardless of `reject_code`. Use `reason` for logging and `reject_code` for programmatic branching.

See ORDER_ACK reject codes in the protocol reference.

23.9 Annotated wire-frame examples

These byte-level examples show exactly how messages look on the wire. Every frame starts with a 4-byte header — magic `BA`, version `01`, message type, flags — then a little-endian `seq_no`, then the fixed-width body. (For the normative frame layouts and field offsets, see the BALF Protocol Reference.)

NEW_ORDER — LIMIT BUY 100 AAPL @ 150.25 (60 bytes):

```
Header:
  BA 01 10 00  |  magic=0xBA, version=1, msg_type=NEW_ORDER(0x10), flags=0
  01 00 00 00  |  seq_no=1 (LE)
Body:
  01 00 00 00  00 00 00 00  |  client_order_id = 1
  41 50 50 4C  00 00 00 00  |  symbol = "AAPL\0\0\0\0"
  40 4e 8f 7f  03 00 00 00  |  price = 15_025_000_000
  00 00 00 00  00 00 00 00  |  stop_price = 0
  00 00 00 00  00 00 00 00  |  trail_offset = 0
  64 00 00 00               |  quantity = 100
  00 00 00 00               |  visible_qty = 0
  01                        |  side = BUY
  02                        |  order_type = LIMIT
  01                        |  tif = DAY
  00                        |  smp = NONE
```

CANCEL_ORDER (24 bytes):

```
Header:
  BA 01 12 00  |  msg_type=CANCEL_ORDER(0x12)
  02 00 00 00  |  seq_no=2
Body:
  02 00 00 00  00 00 00 00  |  client_order_id = 2
  01 00 00 00  00 00 00 00  |  order_id = 1
```

HEARTBEAT (16 bytes):

```
Header:
  BA 01 30 00  |  msg_type=HEARTBEAT(0x30)
  03 00 00 00  |  seq_no=3
Body:
  78 56 34 12  00 00 00 00  |  send_time_ns (example)
```

23.10 Server-initiated messages

These frames arrive **unsolicited** at any point during the session. The client's receive loop must handle them without waiting for a prior request.

23.10.1 HEARTBEAT

The server sends `HEARTBEAT` when no other outbound traffic has been sent for `heartbeat_interval_sec` (default 1 s).

The client **must** reply with `HEARTBEAT_ACK`, echoing `send_time_ns` in the `orig_send_time_ns` field.

If no inbound traffic of any kind arrives within `heartbeat_timeout_sec` (default 5 s), the gateway disconnects the session.

```
HEARTBEAT      →  fill orig_send_time_ns with send_time_ns
HEARTBEAT_ACK  →  echo back orig_send_time_ns
```

See HEARTBEAT and HEARTBEAT_ACK in the protocol reference.

Engine heartbeat

The exchange allows only **one live connection per participant ID**, whichever process holds it. If another process already holds the ID at the engine, the engine refuses the connect and the `LOGON` is answered with `LOGON_ACK(accepted=0, reject_code=0x01)` carrying the message `Gateway already connected: <ID>`.

To make sure a process that dies silently cannot keep its ID locked, `pm-balf-gwy` sends the engine a `system.gateway_heartbeat` for every authenticated session every **60 seconds**, starting right after `LOGON_ACK` (see Message Reference). After **3 consecutive missed beats** (180 s) the engine disconnects the session and applies the participant's `disconnect_behaviour`, so the ID can be connected again. Your BALF client does nothing extra: this is separate from the `HEARTBEAT` / `HEARTBEAT_ACK` exchange above, which is between your client and the gateway (5 s by default). The gateway logs `engine heartbeat sent gateway_id=<ID>` at debug level.

23.10.2 EXECUTION_REPORT

Sent when a resting order is partially or fully filled. Multiple `EXECUTION_REPORT` frames may arrive for a single `NEW_ORDER` (one per fill event).

Field	Description
`client_order_id`	Echoed from the original `NEW_ORDER`
`order_id`	Gateway-assigned session-scoped ID
`fill_price`	Execution price × 10 (i64 LE)
`fill_qty`	Quantity matched in this event
`remaining_qty`	Unfilled quantity after this fill
`status`	`1` = PARTIAL, `2` = FILLED

23.10.3 CANCEL_ACK

A `CANCEL_ACK` may arrive because:

1. **Client cancel confirmed** — the engine processed a `CANCEL_ORDER` request
2. **Cancel rejected** — `CANCEL_ORDER` failed (e.g. order not found); `accepted = 0`
3. **System-originated cancel** — SMP, session-end policy, or expiry; `cancel_reason = 255`

cancel_reason	Meaning
0	Explicit client CANCEL_ORDER confirmed
255	System-originated (SMP, session-end, expiry)

23.10.4 AMEND_ACK

A AMEND_ACK may arrive because:

1. **Amend confirmed** — accepted = 1; new_price, new_quantity, remaining_qty, and priority_reset reflect the post-amend state
2. **Amend rejected** — accepted = 0; AMEND_ORDER failed (e.g. unknown order, or the amended price fails the tick grid or a price collar)

Cancel and amend correlation

The gateway correlates CANCEL_ORDER and AMEND_ORDER requests with the engine ack/reject cycle internally. If the engine rejects the cancel or amend via order.ack, the gateway emits the appropriate CANCEL_ACK or AMEND_ACK with accepted = 0.

23.11 Disconnect behaviour and passive-order policy

On every disconnect trigger the gateway sends system.gateway_disconnect to the engine. The engine applies the disconnect_behaviou configured for that gateway identity in participants:

disconnect_behaviour	Engine action
LEAVE_ALL	Leave all resting orders and active quotes
CANCEL_QUOTES_ONLY	Cancel active quotes; leave non-quote orders
CANCEL_ALL	Cancel all active quotes and all resting orders

Disconnect trigger classes:

- Graceful client LOGOUT
- Heartbeat timeout (heartbeat_timeout_sec without inbound traffic)
- Transport break (TCP FIN / RST, peer closed)
- Protocol safety disconnect (bad framing, unknown msg_type, outbound queue overflow, repeated errors). A non-zero flags byte is currently tolerated (logged, not rejected), not a disconnect trigger

23.12 Duplicate session policy

If a second client sends LOGON with the same gateway ID as an already-connected session, the gateway's response depends on duplicate_session_policy:

Policy	Behaviour
REJECT_NEW (default)	Second LOGON receives LOGON_ACK(accepted=0, reject_code=0x02)
EVICT_OLD	First session is disconnected; second session proceeds to auth

23.13 Example: minimal Python BALF client

The following zero-dependency script demonstrates a complete BALF session: connect, authenticate, submit a LIMIT BUY, read the ORDER_ACK, and gracefully disconnect.

```
#!/usr/bin/env python3
"""Minimal zero-dependency BALF client.

Usage:
    python3 balf_minimal.py --id TRADER01
    python3 balf_minimal.py --host 10.0.0.5 --port 5560 --id TRADER01
"""
import argparse
import socket
import struct
import time

# ── Protocol constants ──────────────────────────────────────────────
BALF_MAGIC   = 0xBA
BALF_VERSION = 0x01
PRICE_SCALE  = 100_000_000  # 10^8

MSG_LOGON              = 0x01
MSG_LOGON_ACK          = 0x02
MSG_NEW_ORDER          = 0x10
MSG_ORDER_ACK          = 0x11
MSG_HEARTBEAT          = 0x30
MSG_HEARTBEAT_ACK      = 0x31
MSG_LOGOUT             = 0x40

SIDE_BUY   = 0x01
ORDER_LIMIT = 0x02
TIF_DAY    = 0x01

FRAME_SIZES = {
    MSG_LOGON:         32,
    MSG_LOGON_ACK:     92,
    MSG_NEW_ORDER:     60,
    MSG_ORDER_ACK:     60,
    MSG_HEARTBEAT:     16,
    MSG_HEARTBEAT_ACK: 16,
    MSG_LOGOUT:         8,
}

# ── Wire helpers ────────────────────────────────────────────────────
def encode_price(display: float) -> int:
    return round(display * PRICE_SCALE)

def build_header(msg_type: int, seq_no: int) -> bytes:
    return struct.pack("<BBBBI", BALF_MAGIC, BALF_VERSION, msg_type, 0, seq_no)

def recv_frame(sock: socket.socket) -> tuple[int, bytes]:
    """Read one complete BALF frame; return (msg_type, body)."""
    def read_exact(n: int) -> bytes:
        buf = bytearray()
        while len(buf) < n:
            chunk = sock.recv(n - len(buf))
            if not chunk:
                raise RuntimeError("connection closed by gateway")
            buf.extend(chunk)
        return bytes(buf)

    header  = read_exact(8)
    msg_type = header[2]
    total   = FRAME_SIZES[msg_type]
    body    = read_exact(total - 8)
    return msg_type, body

# ── Message builders ────────────────────────────────────────────────
def build_logon(gateway_id: str) -> bytes:
    body = struct.pack(
```

Run it:

```
python3 balf_minimal.py --id TRADER01
python3 balf_minimal.py --host 10.0.0.5 --port 5560 --id TRADER01
```

Expected output on success:

```
Connecting to 127.0.0.1:5560 as TRADER01 ...
LOGON accepted — gateway='TRADER01' (gateway=balf-gwy01 hbint=1s)
Sent NEW_ORDER: symbol=AAPL side=BUY type=LIMIT qty=100 price=150.0000 seq=1
ORDER_ACK accepted — order_id=1 timestamp_ns=1751447400000000000
LOGOUT sent — closing connection
```

23.14 Comparing `pm-balf-gwy` with the alternatives

Scenario	Best choice
Human operator on the same machine	`pm-alf-console` (tab completion, history, P&L display)
Python / any-language bot, simple scripts	`pm-alf-gwy` (text lines, easy to debug with nc/telnet)
Latency-sensitive algo in C, C++, or Rust	`pm-balf-gwy` (fixed frames, no text parsing)
Browser UI / REST stack	`pm-api-gwy`
Read-only market-data consumer	`pm-md-gwy` (CALF)
Post-trade / clearing / audit consumer	`pm-ralf-gwy` (RALF)

23.15 Troubleshooting

23.15.1 Check whether the port is in use

macOS:

```
sudo lsof -iTCP:5560 -sTCP:LISTEN
netstat -an | grep LISTEN | grep 5560
```

Linux:

```
ss -tlnp 'sport = :5560'
sudo lsof -iTCP:5560 -sTCP:LISTEN
```

If no output appears, the gateway is not running or is bound to a different port. Check `balf_gateway.port` in `engine_config.yaml`.

23.15.2 Non-interactive TCP reachability test

Because BALF is binary, `nc` cannot be used interactively. Use the quick connectivity test script from the Quick connectivity test section above, or send a minimal LOGON with a one-liner:

```
python3 - <<'EOF'
import socket, struct
sock = socket.create_connection(("127.0.0.1", 5560), timeout=3)
header = struct.pack("<BBBBI", 0xBA, 0x01, 0x01, 0, 0)
body   = struct.pack("<16sB7s", b"PROBE\x00"*1, 0x01, b"\x00"*7)
sock.sendall(header + body)
import sys; ack = bytearray()
while len(ack) < 92:
    ack.extend(sock.recv(92 - len(ack)))
print("accepted" if ack[24] else "rejected (code=0x{:02X})".format(ack[25]))
EOF
```

23.15.3 Common problems

Symptom	Likely cause	Fix
`Connection refused`	Gateway not started or wrong port	Confirm `pm-balf-gwy` is running; check `balf_gateway.port`
Connection accepted but LOGON_ACK never arrives	Engine not running or ZMQ link lost	Start `pm-engine`; check gateway logs
`LOGON_ACK accepted=0, code=0x01`	Gateway ID not in `participants` — or, when the message reads `Gateway already connected`, another process holds the ID at the engine	Add the ID under `participants` and restart engine; or stop the other process (if it was killed, the engine frees the ID within 3 minutes)
`LOGON_ACK accepted=0, code=0x02`	Same gateway ID already connected	Disconnect the other session, or use `duplicate_session_policy: EVICT_OLD`
`LOGON_ACK accepted=0, code=0x03`	`proto_version` byte is not `1`	Fix the LOGON frame builder
`LOGON_ACK accepted=0, code=0xFF, msg=SYMBOLS_UNAVAILABLE`	Engine accepted the connection but never answered `system.symbols_request`	Check `pm-engine` health, then log on again
Session receives `LOGOUT` and the TCP connection closes	`pm-engine` restarted, so it no longer holds this session	Log on again; the gateway re-requests symbol metadata
Connection closes ~5 s after last message	`heartbeat_timeout_sec` elapsed	Send `HEARTBEAT` frames and reply to server `HEARTBEAT` with `HEARTBEAT_ACK`
`ORDER_ACK accepted=0`	Field validation failure or engine rejection	Inspect `reason` field; check reject code table above
Unknown `msg_type` → connection closed	Client sent a server-direction frame	Only send client-direction frames: `LOGON`, `NEW_ORDER`, `CANCEL_ORDER`, `AMEND_ORDER`, `HEARTBEAT`, `HEARTBEAT_ACK`, `LOGOUT`
Corrupt response / partial reads	Client reads wrong number of bytes	Always read header first (8 bytes), look up `FRAME_SIZES[msg_type]`, then read exactly that many additional bytes
Gateway not reachable from another host	`bind_address: 127.0.0.1`	Change `bind_address` to `0.0.0.0` or the specific interface IP

23.16 See also

In this book:

- ALF TCP Gateway — text-protocol alternative for Python/any-language clients where binary parsing is not required
- The Configuration Workflow — `balf_gateway:` section and `participants` allowlist

Reference Manual:

- Processes — process topology and ZMQ message tables

Protocols and Clients:

- Appendix: BALF Protocol Reference — normative frame layouts, byte offsets, encoding rules, and worked hex examples
- External Protocols Overview — protocol comparison and selection guide

CHAPTER 24

CALF Gateway — Market Data

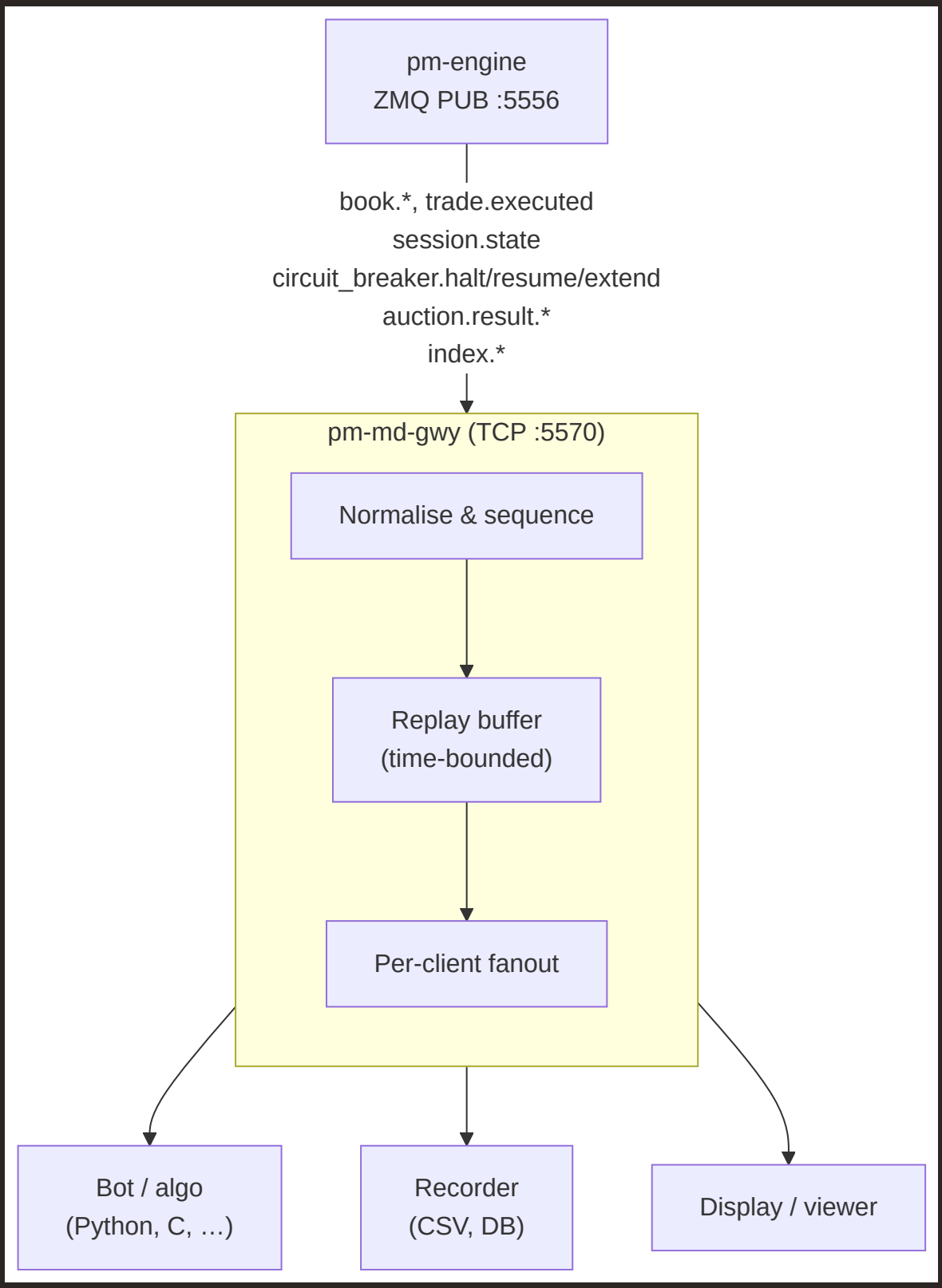
Learning objectives

After reading this page you will understand:

- what `pm-md-gwy` does and why CALF exists as an external feed
- what data is available on each channel (`TOP`, `TRADE`, `STATE`, `INDEX`, `DEPTH`, `AUCTION`, `CB`)
- which channels accept the `SYM=*` wildcard (`TOP`, `TRADE`, `STATE`, `AUCTION` — not `INDEX`/`DEPTH`/`CB`)
- how to detect gateway capability via `WELCOME|CH_SUPPORTED=` before relying on `DEPTH`
- when to choose CALF over the other available protocols
- how to start the gateway and verify connectivity from a terminal
- how to subscribe to a filtered subset of symbols and channels
- how snapshot delivery works and when to expect a `SNAP`
- how to detect sequence gaps and recover with `RESUME`
- how to write a working Python subscriber using the library in `docs/examples/calf/`, or the production `edumatcher.calf_client` package
- the exact fields carried by every message on every channel
- what kinds of tools you can build on the feed, and how a browser client reaches it through a server-side WebSocket bridge
- which operational checks to use when debugging connectivity problems

24.1 What this process is

`pm-md-gwy` is the CALF market-data gateway. The matching engine publishes market events on an internal ZeroMQ PUB socket (`:5556`) that is not accessible to external clients. `pm-md-gwy` bridges that gap: it subscribes to the engine PUB socket, normalises raw engine events into CALF lines, and re-publishes them over TCP (default port `5570`) in a format any language can consume with a plain socket and line-split logic.



Responsibilities of `pm-md-gwy`:

- translates engine events into CALF lines
- assigns **per-stream sequence numbers** on each `(channel, symbol)` pair
- keeps a **time-bounded replay buffer** so reconnecting clients can recover missed messages without a full snapshot
- sends an automatic **baseline snapshot** (`SNAP`) when a client first subscribes to a `TOP`, `STATE`, `INDEX`, `DEPTH`, or `CB` stream (a wildcard `TOP` subscription gets one real `SNAP` per known symbol, not a single `SYM=*` snapshot); `TRADE` and `AUCTION` never send one
- advertises which channels a given gateway build supports via `WELCOME|CH_SUPPORTED=`, so clients can detect `DEPTH`/`INDEX`/wildcard

availability without a protocol version bump
- enforces per-client subscription limits and disconnects slow clients
- asks the engine for its session phase every two seconds, over the engine's PULL socket (`:5555`). This is the gateway's only traffic towards the engine. The engine announces its phase only on a transition, so without the query a gateway started after the last transition, or an exchange started outside trading hours, would never learn it. Until the first answer arrives, `STATE` reports `SESSION=UNKNOWN`.

24.2 The feed at a glance

Everything a client needs to know in one place. Read this section first; the rest of the chapter is detail.

Property	Value
Process	`pm-md-gwy`
Transport	plain TCP, newline-delimited UTF-8 text
Default port	`5570` (`market_data_gateway.port`)
Protocol token	`CALF1` (in `HELLO` / `WELCOME`) — unchanged across gateway builds
Line format	`MSGTYPE|KEY=VALUE|KEY=VALUE\n`
Max line length	4096 bytes (longer → `ERR|CODE=BAD_MESSAGE`)
Channels	`TOP`, `TRADE`, `STATE`, `INDEX`, `DEPTH`, `AUCTION`, `CB`
Heartbeat	every `heartbeat_interval_sec` (default 1 s) when a stream is quiet
Idle timeout	`EXIT|REASON=idle_timeout`, then disconnect, after `idle_timeout_sec` (default 5 s) with no traffic in either direction
Replay window	`replay_window_sec` (default 30 s), tracked per `(channel, symbol)` stream
Values	prices as decimal text, sizes as integers, timestamps ISO-8601 UTC

Because it is line-based text over TCP, a client needs nothing more than a socket and a newline split — no ZeroMQ binding, no schema compiler, no dependencies. That is the whole point of CALF: **any language can read the market in a few lines of code.**

24.2.1 Message catalogue

Client → gateway

Message	Purpose
`HELLO|CLIENT=..|PROTO=CALF1`	Open a session
`RESUME|CH=..|SYM=..|LASTSEQ=..`	Replay one stream from a known sequence; send one per stream on reconnect
`SUB|CH=..|SYM=..`	Subscribe. Channels × symbols, comma-separated; `SYM=*` where allowed. Cumulative across lines
`SYMBOLS`	Ask which instruments the gateway knows; replies `SYMBOLS|COUNT=n|SYMBOLS=..|REF=...` Repeatable — use this rather than relying on `WELCOME|SYMBOLS=`, which is optional and sent once
`UNSUB|CH=..|SYM=..`	Cancel subscriptions (idempotent)

Message	Purpose
`PING`	Liveness probe — gateway replies `PONG`
`EXIT`	Close the session

Gateway → client

Message	Channel(s)	Meaning
`WELCOME`	—	Session accepted; carries `GW`, `HBINT`, `REPLAY`, `SYMBOLS`, `REF`, `CH_SUPPORTED`
`SNAP`	TOP, STATE, INDEX, DEPTH, CB	Baseline snapshot for a stream — the `SEQ` you anchor on
`MD`	TOP	Incremental top-of-book change (only the fields that changed)
`TRADE`	TRADE	One executed trade
`STATE`	STATE	Session-phase transition, or a symbol halt/resume
`IDX`	INDEX	Index level recalculation
`DEPTH`	DEPTH	Full top-N ladder for the changed side(s)
`AUCTION`	AUCTION	Auction uncross result (equilibrium price/qty, imbalance)
`INDIC`	AUCTION	Indicative uncross price/qty while a call phase is still open — see CALF protocol → `INDIC`
`CB`	CB	Circuit-breaker halt or resume detail for one symbol
`HB`	—	Heartbeat while the stream is quiet
`PONG`	`TS`	Reply to `PING`
`ERR|CODE=..`	—	Protocol or subscription error (see Common errors)

24.2.2 The envelope on every market-data line

`SNAP`, `MD`, `TRADE`, `STATE`, `IDX`, `DEPTH`, `AUCTION`, and `CB` all begin with the same four envelope fields, followed by channel-specific payload fields:

Field	Meaning
`CH`	Channel: `TOP`, `TRADE`, `STATE`, `INDEX`, `DEPTH`, `AUCTION`, or `CB`
`SYM`	Symbol; the index id on `INDEX`; or `*` on a session-wide `STATE` line
`SEQ`	Per-`(CH, SYM)` sequence number — starts at 1, +1 per message. Your gap detector
`TS`	Event time, ISO-8601 UTC with milliseconds (e.g. `2026-06-30T09:30:00.000Z`)

24.2.3 The seven channels

Channel	Incremental msg	Baseline SNAP?	SYM=*?	Payload fields	Primary use
TOP	MD	Yes (one per symbol when SYM=*)	Yes (1.0.0+)	BID BIDSZ ASK ASKSZ LAST LASTSZ	best bid/ask/last, price widgets, algos
TRADE	TRADE	No	Yes (1.0.0+)	TRADE_ID RUN_SEQ PX QTY SIDE	time-and-sales tape, VWAP/OHLCV
STATE	STATE	Yes	Yes	SESSION PREV	halt gating, session-phase display
INDEX	IDX	Yes (1.0.0+)	No	LEVEL SESSION OPEN CHG PCTCHG HIGH LOW AGGCAP	index trackers, benchmarks
DEPTH	DEPTH	Yes	No	LEVELS BIDS ASKS	order-book (DOM) widgets, Level-2 teaching
AUCTION	AUCTION	No	Yes	EQPX EQQTY TRADES IMBSIDE IMBQTY REASON	auction uncross results, Terminal-style auction views
CB	CB	Yes	No	STATUS LEVEL TRIGGERPX REFPX RESUMEAT CORRLO CORRHI EXP SRC INDICPX INDICQTY IMB REASON CLAMPED PRINTPX	circuit-breaker detail beyond STATE's halt/resume flag, including the ACE reopening corridor

Each channel is detailed under What information is available; the exact field meanings are in the per-channel field tables there.

24.3 Prerequisites

- pm-engine running
- pm-md-gwy running
- symbols configured in engine_config.yaml

Optional but recommended in config:

```
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_symbols_per_client: 200
  max_client_queue: 10000
  max_connections: 64
  max_messages_per_second: 200
  depth_levels: 10
```

All keys are optional; the defaults below apply when the block (or a key) is omitted. pm-md-gwy reads only this market_data_gateway block — see Configuration → Which Process Reads What.

Key	Default	Purpose
`enabled`	`true`	When `false`, `pm-md-gwy` logs a warning and exits immediately without binding the TCP port
`name`	`md-gwy01`	Gateway id reported in `WELCOME|GW=`
`bind_address`	`0.0.0.0`	TCP listen address
`port`	`5570`	TCP listen port
`heartbeat_interval_sec`	`1`	`HB` cadence when idle; advertised as `WELCOME|HBINT=`
`idle_timeout_sec`	`5`	Disconnect a client after this many seconds with no traffic in **either** direction. Outbound market data and heartbeats reset the timer, so a purely passive consumer stays connected without sending anything; a client is aged out only once writes to it stop succeeding
`replay_window_sec`	`30`	Per-stream replay retention; advertised as `WELCOME|REPLAY=`
`max_symbols_per_client`	`200`	Cap on the number of *unique symbol strings* a client has subscribed to, counted once across all channels — `SYM=*` counts as a single entry; exceeding it returns `ERR|CODE=SUB_LIMIT`
`max_client_queue`	`10000`	Outbound backlog limit; exceeding it drops the backlog, sends `ERR|CODE=SLOW_CLIENT` and disconnects the client (within 5 s if it has stopped reading)
`max_connections`	`64`	Max simultaneous TCP connections; an over-limit connection is accepted at the socket level and then closed immediately with no `HELLO`/`WELCOME` exchange at all
`max_messages_per_second`	`200`	Inbound token-bucket rate limit per client; exceeding it returns `ERR|CODE=RATE_LIMITED` (connection stays open)
`depth_levels`	`10`	Price levels per side on the `DEPTH` channel; surfaced as `LEVELS=`

All values must be positive integers (`port`, the intervals, the limits); the gateway refuses to start otherwise.

If engine_config.yaml fails to load

If `pm-md-gwy` cannot load the engine config's symbol list at startup (missing file, parse error, etc.), it silently falls back to an empty known-symbol set, which disables known-symbol validation gateway-wide — any non-wildcard symbol is then accepted on `SUB`, not just configured ones. Treat this as a misconfiguration to fix, not a supported mode.

24.4 Start the gateway

Installed mode:

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

Developer mode:

```
poetry run pm-engine --verbose
poetry run pm-md-gwy
```

CLI override options:

Option	Default	Description
`--bind ADDR`	from config / `0.0.0.0`	Override TCP bind address
`--port PORT`	from config / `5570`	Override TCP listen port
`--engine-pub ADDR`	`tcp://127.0.0.1:5556`	Override engine PUB socket address — always the fixed engine-side default; not configurable via the `market_data_gateway` YAML block
`--index-pub ADDR`	`tcp://127.0.0.1:5558`	Override index PUB socket address
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	Reduce output to warnings/errors
`--log-target`	auto	Where operational log records go: `server` (default, auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file PATH`	—	Operational log file path — required when `--log-target file`
`--log-failover-timeout SECONDS`	`30`	Grace window before falling back to a local log file once `pm-log-srv` becomes unreachable
`--version`	—	Print the installed `pm-md-gwy` version and exit

The `--engine-pub`/`--index-pub` defaults themselves can be shifted for the whole installation via two environment variables (useful when the engine runs on another host): `EDUMATCHER_ENGINE_HOST` (default `127.0.0.1`) and `EDUMATCHER_INDEX_PUB_PORT` (default `5558`). The engine configuration itself is not overridable: like every other `pm-*` process, `pm-md-gwy` reads `<EDUMATCHER_DATA_DIR>/re` — see Reference Manual → Environment variables.

This matters more for `pm-md-gwy` than for most processes. Its symbol universe comes from that file, and it is what `WELCOME|SYMBOLS=` and the `SYMBOLS` reply are built from; a gateway reading a different configuration from the engine would advertise an instrument list no client could trade.

Each symbol's `tick_decimals` comes from the same file and is advertised in `REF=` (see CALF protocol → `SYMBOLS`). It is the only route a market data client has to an instrument's display precision, so a gateway started without a readable engine config leaves every client rendering prices at the default of two decimals.

24.5 Quick connect test

Use `nc` (or `telnet`) to validate the line protocol from the command line before writing any code:

Prefer a ready-made client?

`pm-calf-spy` does everything below for you — handshake, subscribe, and pretty-print every line — without hand-typing protocol messages: `pm-calf-spy --channels TOP,TRADE --symbols AAPL`.

```
nc 127.0.0.1 5570
```

Then type:

```
HELLO|CLIENT=demo01|PROTO=CALF1
SUB|CH=TOP,TRADE|SYM=AAPL
```

Expected response pattern:

1. `WELCOME|...` — session open
2. `SNAP|CH=TOP|SYM=AAPL|...` — baseline snapshot
3. `MD|...` when top of book changes
4. `TRADE|...` when a trade executes
5. `HB|...` when the stream is quiet

To verify the STATE channel and the wildcard subscription:

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

You should receive an immediate `SNAP|CH=STATE|SYM=*|...` followed by live `STATE|...` lines on session-phase and halt/resume events.

Since CALF `1.0.0`, `SYM=*` also works for `TOP` and `TRADE` — useful for a market-wide trade tape or "watch everything" bot:

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

Unlike `STATE`'s wildcard, a wildcard `TOP` subscription does **not** send a single `SNAP|SYM=*`. It sends one real `SNAP` per symbol the gateway currently knows about, then live `MD` for any symbol — including ones added later — through that same subscription.

To verify the `DEPTH` channel (check `WELCOME|CH_SUPPORTED=` first — see below):

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

Expect an immediate `SNAP|CH=DEPTH|SYM=AAPL|LEVELS=...|BIDS=...|ASKS=...`, then a `DEPTH|...` line whenever the top price levels change.

To verify the `AUCTION` channel (fires around auction uncrosses — try it around the open/close auction or a re-opening after a halt):

```
SUB|CH=AUCTION|SYM=AAPL
```

There is no `SNAP` — you'll see an `AUCTION|...` line the next time that symbol's auction uncrosses.

To verify the `CB` channel:

```
SUB|CH=CB|SYM=AAPL
```

Expect an immediate `SNAP|CH=CB|SYM=AAPL|STATUS=ACTIVE|...` (or `STATUS=HALTED` plus detail fields if the symbol happens to already be halted), then a `CB|...` line on the next halt or resume for that symbol.

24.6 What information is available

The gateway exposes seven logical channels. Each represents a different view of market activity.

24.6.1 Channel `TOP` — best bid, ask, and last trade

`TOP` carries incremental updates to the top-of-book for a single symbol. When the best bid price, bid size, ask price, ask size, or last-trade price/size changes, the gateway emits one `MD` line containing only the fields that changed.

When you get it: Subscribe with `CH=TOP|SYM=<symbol>`. The gateway immediately sends a `SNAP` baseline, then streams incremental `MD` events. Since CALF `1.0.0`, `SYM=*` is also valid on `TOP` — it produces one `SNAP` per known symbol rather than a single wildcard snapshot (see Step 3 — Receive snapshots for the full walkthrough of this burst behaviour).

Typical use cases: algo trading, live bid/ask/last widgets, real-time price tracking.

Wire example:

```
SNAP|CH=TOP|SYM=AAPL|SEQ=1|TS=2026-06-30T09:30:00.000Z|BID=150.10|BIDSZ=1200|ASK=150.12|ASKSZ=900|LAST=150.11|LASTSZ=300
MD|CH=TOP|SYM=AAPL|SEQ=2|TS=2026-06-30T09:30:00.500Z|BID=150.11|BIDSZ=1400
MD|CH=TOP|SYM=AAPL|SEQ=3|TS=2026-06-30T09:30:01.100Z|ASK=150.13|ASKSZ=700|LAST=150.12|LASTSZ=200
```

Fields (payload after the `CH SYM SEQ TS` envelope):

Field	Type	Meaning
`BID`	decimal	Best (highest) bid price

Field	Type	Meaning
BIDSZ	integer	Total resting size at the best bid
ASK	decimal	Best (lowest) ask price
ASKSZ	integer	Total resting size at the best ask
LAST	decimal	Last trade price
LASTSZ	integer	Last trade size

A SNAP carries every field currently known; an MD carries **only the fields that changed**. Fields omitted from an MD are unchanged — merge each MD into a per-symbol state object seeded from the SNAP.

24.6.2 Channel TRADE — every trade print

TRADE carries one line per matched trade: its durable trade identity, price, quantity, and aggressor side. There is **no baseline SNAP** — the stream starts from events that occur after the subscription becomes active.

When you get it: Subscribe with CH=TRADE|SYM=<symbol>, or SYM=* (CALF 1.0.0+) to receive every trade across every symbol on one subscription — handy for a market-wide tape without enumerating tickers.

Typical use cases: time-and-sales tape, VWAP/OHLCV calculation, fill attribution.

Wire example:

```
TRADE|CH=TRADE|SYM=AAPL|SEQ=44|TS=2026-06-30T09:30:01.100Z|TRADE_ID=000042-000000123|RUN_SEQ=42|PX=150.12|QTY=200|SIDE=BUY
```

Fields:

Field	Type	Meaning
TRADE_ID	string	Durable public trade ID; unique across engine restarts
RUN_SEQ	integer	Persisted engine-run sequence encoded in TRADE_ID's prefix
PX	decimal	Execution price
QTY	integer	Executed quantity
SIDE	enum	Aggressor side — BUY or SELL (empty when the engine did not report one)

TRADE_ID is the reconciliation key for the public tape. It also makes reconnect overlap safe: consumers should retain a seen-ID set and ignore a replayed print whose TRADE_ID was already processed. SEQ still detects gaps in the (TRADE, SYM) stream; TRADE_ID answers whether two deliveries name the same execution.

24.6.3 Channel STATE — session and symbol states

STATE carries two kinds of transitions:

- **Session-wide** (e.g. PRE_OPEN → OPENING_AUCTION → CONTINUOUS) with SYM=*
- **Symbol-level** circuit-breaker halts and resumes with SYM=<symbol>

The gateway sends an immediate SNAP for each new (STATE, symbol) stream.

STATE was the first channel to support SYM=*. Since CALF 1.0.0, TOP and TRADE also accept it (in any combination with STATE). SYM=* is never valid for INDEX or DEPTH — see "Channel summary" below.

When you get it: CH=STATE|SYM=* for everything, or CH=STATE|SYM=<symbol> for a single symbol.

Typical use cases: gating order flow on halts, session-phase displays, back-test state annotation.

Wire example:

```
SNAP|CH=STATE|SYM=*|SEQ=1|TS=2026-06-30T09:30:00.000Z|SESSION=PRE_OPEN
STATE|CH=STATE|SYM=*|SEQ=2|TS=2026-06-30T09:30:00.000Z|SESSION=OPENING_AUCTION|PREV=PRE_OPEN
STATE|CH=STATE|SYM=*|SEQ=3|TS=2026-06-30T09:30:05.000Z|SESSION=CONTINUOUS|PREV=OPENING_AUCTION
STATE|CH=STATE|SYM=AAPL|SEQ=1|TS=2026-06-30T10:02:17.000Z|SESSION=HALTED|PREV=CONTINUOUS
STATE|CH=STATE|SYM=AAPL|SEQ=2|TS=2026-06-30T10:05:00.000Z|SESSION=CONTINUOUS|PREV=HALTED
```

Fields:

Field	Type	Meaning
`SESSION`	enum	New state. On `SYM=*`: `PRE_OPEN`, `OPENING_AUCTION`, `CONTINUOUS`, `CLOSING_AUCTION`, `CLOSED`. On a single symbol: `HALTED` (circuit-breaker/operator halt) or `CONTINUOUS` (resume)
`PREV`	enum	Previous state; present on transitions and on halt/resume, omitted on the first baseline `SNAP`

24.6.4 Channel `INDEX` — index level updates

`INDEX` carries one `IDX` line every time the index level is recalculated. Since CALF `1.0.0`, the gateway sends an immediate baseline `SNAP` on subscribe — the same pattern as `TOP`/`STATE`/`DEPTH` — followed by live `IDX` updates.

When you get it: `CH=INDEX|SYM=<index_id>` (e.g. `CH=INDEX|SYM=EDU50`). `SYM=*` is not valid for `INDEX` — an explicit index id is always required.

Typical use cases: index trackers, portfolio benchmark display, monitoring day-open / day-high / day-low for the composite index.

Wire example:

```
SNAP|CH=INDEX|SYM=EDU50|SEQ=1|TS=2026-06-30T09:30:00.000Z|LEVEL=5100.00|SESSION=PRE_OPEN
IDX|CH=INDEX|SYM=EDU50|SEQ=12|TS=2026-06-30T09:30:01.100Z|LEVEL=5123.45|SESSION=CONTINUOUS|OPEN=5100.00|CHG=+23.45|PCTCHG=+0.46|HIGH=
```

Fields (on `CH=INDEX`, `SYM` carries the index id):

Field	Type	Present	Meaning
`LEVEL`	decimal	always	Current index level
`SESSION`	enum	always	Session state at this calculation
`OPEN`	decimal	when known	Day-open index level
`CHG`	signed decimal	when `OPEN` known	`LEVEL − OPEN`
`PCTCHG`	signed decimal	when `OPEN` known	Percent change vs `OPEN`
`HIGH`	decimal	when known	Day-high level
`LOW`	decimal	when known	Day-low level
`AGGCAP`	integer	when known	Aggregate market cap of constituents (Σ price × shares)

24.6.5 Channel `DEPTH` — aggregated multi-level order book

`DEPTH` carries the top N price levels per side (Level 2 — aggregated by price, never per individual order). Whenever the ladder changes on either side, the gateway emits a `DEPTH` line carrying the **complete current ladder for both sides that**

currently have resting liquidity — not a per-level diff, and not filtered down to only the side that changed. A side is omitted from the line only when it is empty, never because it happens not to have moved. A client always replaces its in-memory ladder for both sides on receipt.

When you get it: Subscribe with `CH=DEPTH|SYM=<symbol>`. The gateway immediately sends a `SNAP` baseline, then streams `DEPTH` updates whenever the top levels change. `SYM=*` is not valid for `DEPTH` — it is deliberately excluded because a wildcard depth subscription could multiply one client's bandwidth footprint by the entire symbol count.

Typical use cases: order-book visualisation (DOM/ladder widgets), simple liquidity/depth analysis, teaching Level 2 concepts.

Wire example:

```
SNAP|CH=DEPTH|SYM=AAPL|SEQ=1|TS=2026-06-30T09:30:00.000Z|LEVELS=10|BIDS=150.10:1200:3,150.09:800:2|ASKS=150.12:900:2,150.13:600:1
DEPTH|CH=DEPTH|SYM=AAPL|SEQ=2|TS=2026-06-30T09:30:00.500Z|LEVELS=10|BIDS=150.10:1400:4,150.09:800:2|ASKS=150.12:900:2,150.13:600:1
```

Fields:

Field	Type	Meaning
`LEVELS`	integer	Levels per side the gateway tracks (`market_data_gateway.depth_levels`, default 10)
`BIDS`	list	Bid ladder, best (highest) first, comma-separated `PRICE:QTY:COUNT` triples
`ASKS`	list	Ask ladder, best (lowest) first, same encoding

Each level in `BIDS`/`ASKS` is `PRICE:QTY:COUNT`, comma-separated, best price first. `QTY` is the total resting quantity at that price; `COUNT` is how many individual orders were aggregated into it. A side with no liquidity omits its field entirely — not because it wasn't the side that changed. Every `DEPTH` line (and the `SNAP`) carries the **complete** current ladder for both sides that have any resting liquidity — replace, don't merge.

The number of levels per side (`LEVELS`, default 10) is a gateway-wide setting (`market_data_gateway.depth_levels`) — there is no per-client override in CALF `1.0.0`. Not every gateway build supports `DEPTH` yet; check `WELCOME|CH_SUPPORTED=` before relying on it (see "Connecting and subscribing" below).

24.6.6 Channel `AUCTION` — auction uncross results

`AUCTION` carries one line every time a symbol's auction uncrosses: the equilibrium price, matched quantity, number of trades generated, and any remaining imbalance. There is **no baseline `SNAP`** — like `TRADE`, the stream starts from events that occur after the subscription becomes active.

A subscription to `AUCTION` will also deliver `INDIC` lines during a call phase — indicative uncross updates before the phase actually closes. See CALF protocol → `INDIC`.

When you get it: Subscribe with `CH=AUCTION|SYM=<symbol>`, or `SYM=*` to receive every symbol's auction results on one subscription — useful for a market-wide auction monitor without enumerating tickers. Expect events around the opening auction, closing auction, and any re-opening auction after a circuit-breaker halt.

Typical use cases: auction monitors, Terminal-style auction panels, open/close imbalance dashboards, halt re-opening confirmation.

Wire example:

```
AUCTION|CH=AUCTION|SYM=AAPL|SEQ=1|TS=2026-07-20T09:30:00.000Z|EQPX=150.10|EQQTY=48200|TRADES=37|IMBSIDE=BUY|IMBQTY=1400
```

A no-cross auction (no matched quantity) omits `EQPX` and `IMBSIDE`:

```
AUCTION|CH=AUCTION|SYM=MSFT|SEQ=1|TS=2026-07-20T09:30:00.000Z|EQQTY=0|TRADES=0|IMBQTY=0
```

Fields:

Field	Type	Present	Meaning
EQPX	decimal	when a cross occurred	Equilibrium (uncross) price
EQQTY	integer	always	Matched quantity at the equilibrium price
TRADES	integer	always	Number of trades generated by the uncross
IMBSIDE	enum	when an imbalance remains	Side of the remaining imbalance — BUY or SELL
IMBQTY	integer	always	Remaining imbalance quantity (0 when fully matched)

EQQTY, TRADES, and IMBQTY are always present (0 when there's nothing to report); EQPX and IMBSIDE are omitted rather than sent as 0/empty when there was no cross or no imbalance, so don't treat a missing EQPX as a literal zero price.

24.6.7 Channel CB — circuit-breaker detail

CB carries the same halt/resume events already visible on STATE, but with the operational detail STATE doesn't carry: trigger price, reference price, breaker ladder level, scheduled resume time, and re-opening mode. Think of it as the detail view behind STATE's simple HALTED/CONTINUOUS flag.

The gateway sends an immediate SNAP for each new (CB, symbol) stream, reflecting the symbol's current status — STATUS=ACTIVE with no other fields if it has never been halted, or the current halt detail if it has.

When you get it: Subscribe with CH=CB|SYM=<symbol>. SYM=* is **not** valid for CB — unlike AUCTION, a circuit-breaker event is rare and per-symbol/operator-relevant rather than a firehose case, so an explicit symbol is always required.

Typical use cases: halt detail panels, breaker-ladder-level display, resume-time countdowns, distinguishing an automatic trigger from an operator (ADMIN_ALL/ADMIN_SYMBOL) halt.

Wire example — automatic halt, then resume:

```
SNAP|CH=CB|SYM=AAPL|SEQ=1|TS=2026-07-20T10:02:17.000Z|STATUS=ACTIVE
CB|CH=CB|SYM=AAPL|SEQ=2|TS=2026-07-20T10:02:17.000Z|STATUS=HALTED|LEVEL=L2|TRIGGERPX=148.20|REFPX=150.10|RESUMEAT=2026-07-20T10:20:00
CB|CH=CB|SYM=AAPL|SEQ=3|TS=2026-07-20T10:20:00.000Z|STATUS=ACTIVE|SRC=CB
```

Wire example — a halt extended by ACE before it reopens:

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 one rung, and a fresh call phase begins — see Risk Controls - Automated Corridor Expansion.

```
CB|CH=CB|SYM=AAPL|SEQ=4|TS=2026-07-20T13:30:00.010Z|STATUS=HALTED|LEVEL=L1|TRIGGERPX=122.00|REFPX=100.00|RESUMEAT=2026-07-20T13:35:00
CB|CH=CB|SYM=AAPL|SEQ=5|TS=2026-07-20T13:35:00.010Z|STATUS=HALTED|LEVEL=L1|TRIGGERPX=122.00|REFPX=100.00|RESUMEAT=2026-07-20T13:37:00
CB|CH=CB|SYM=AAPL|SEQ=6|TS=2026-07-20T13:37:00.010Z|STATUS=ACTIVE|SRC=CB
```

A client that ignores EXP/CORRLO/CORRHI will show a RESUMEAT that has already passed and report the symbol as overdue to reopen. Note also that no STATE line accompanies an extension: the symbol was halted before it and is halted after it, so the coarse session state has not changed.

If the trading day ends before ACE resolves, the resume says so and reports the price it was forced to print at:

```
CB|CH=CB|SYM=AAPL|SEQ=9|TS=2026-07-20T16:05:00.010Z|STATUS=ACTIVE|SRC=CB|REASON=CLOSING_BACKSTOP|CLAMPED=1|PRINTPX=120.00
```

CLAMPED=1 means the price was imposed at the corridor boundary rather than discovered by the book.

Note that STATE also emits its own, simpler pair of lines for the same halt/resume — CB and STATE fire independently for the same event, so a client that wants both just subscribes to both channels; they are not sequenced against each other.

Fields:

Field	Type	Present	Meaning
STATUS	enum	always	ACTIVE or HALTED
LEVEL	string	on halt	Circuit-breaker ladder level (e.g. L1/L2/L3)
TRIGGERPX	decimal	on automatic halt	Price that triggered the breaker
REFPX	decimal	on automatic halt	Reference price the trigger was measured against
RESUMEAT	string	on halt, when scheduled	Scheduled resume time, ISO-8601 UTC with ms (same format as TS)
SRC	enum	on halt and resume	What halted the symbol: CB or ADMIN

An operator-initiated halt (ADMIN_ALL/ADMIN_SYMBOL) omits TRIGGERPX, REFPX, and RESUMEAT since there was no automatic trigger and often no scheduled time. SRC is carried on both the halt and the resume line, since what caused the halt is as relevant on the way out as on the way in.

There is no "re-opening mode" field, because there is no choice to make: a halt is the call phase of a reopening auction — LIMIT orders rest, matching is off — and it always ends in an uncross, published on the AUCTION channel with REASON=REOPEN just before the resume.

24.6.8 Channel summary

See the seven-channel overview table at the top of the chapter for the at-a-glance comparison (message type, snapshot behaviour, wildcard support, and payload per channel).

24.7 When to use CALF — protocol comparison

EduMatcher offers several ways to obtain market data. The right choice depends on your context.

Approach	Transport	Data available	Best for	Not suitable for
CALF (pm-md-gwy)	TCP text	TOP, TRADE, STATE, INDEX, DEPTH, AUCTION, CB	External clients; any language; snapshot + replay	Internal Python code that already imports edumatcher
Internal ZMQ PUB (:5556)	ZMQ binary	Raw engine events (book.*, trade.executed, …)	Internal Python processes (pm-stats, bots)	External clients; languages without a ZMQ binding
REST API (pm-api-gwy)	HTTP/JSON	Snapshot queries; order status	Web dashboards; one-shot queries	Low-latency streaming; high-frequency incremental data
WebSocket API (pm-api-gwy)	WebSocket/JSON	Streaming market data (JSON)	Browser-based UIs; REST-native stacks	Latency-critical paths
RALF (pm-ralf-gwy)	TCP text	Post-trade events (fills, positions, clearing)	External clearing, drop-copy, audit consumers	Pre-trade market data; top-of-book streaming

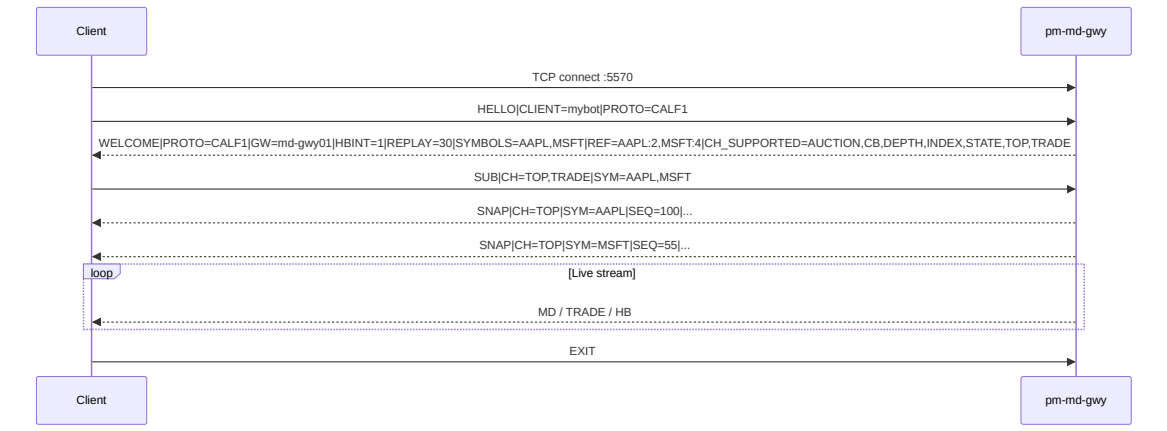
Approach	Transport	Data available	Best for	Not suitable for
Drop Copy (ZMQ `:5557`)	ZMQ binary	Fill events only	Internal fill-monitoring processes	General market data

Key rules:

- External client in any language → **use CALF**.
- Internal Python process that imports `edumatcher` → use ZMQ PUB directly via `make_subscriber()`.
- Web UI → use the REST/WebSocket API gateway.
- Post-trade / clearing / audit → use RALF.

24.8 Connecting and subscribing

Every CALF session follows this sequence:



24.8.1 Step 1 — Send `HELLO`

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

`CLIENT` is a free-text identifier (max 32 chars) used for gateway logging. `PROTO` must be exactly `CALF1` — this does **not** change between CALF `1.0.0` and earlier gateways. The gateway replies with `WELCOME` or closes the connection on protocol error. Check the `SYMBOLS` field in `WELCOME` for the list of configured symbols — useful for building a dynamic subscription list. `SYMBOLS` is **omitted entirely** if the gateway has not learned of any symbols yet, so treat a missing field the same as an empty list rather than an error. Also check `CH_SUPPORTED`: if present, it lists every channel this gateway build actually supports (e.g. `AUCTION,CB,DEPTH,INDEX,STATE,TOP,TRADE`). If `CH_SUPPORTED` is **absent**, assume a pre-`1.0.0` gateway that only supports `TOP`/`TRADE`/`STATE` and no `SYM=*` wildcard outside `STATE`; `AUCTION` and `CB` are newer still, so check for their individual presence in `CH_SUPPORTED` rather than assuming they ship together with `DEPTH`/`INDEX`.

24.8.2 Step 2 — Subscribe

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

Multiple channels and symbols are comma-separated. The subscription is the Cartesian product of all listed channels × symbols. Multiple `SUB` lines are cumulative; existing subscriptions are preserved.

Since CALF `1.0.0`, `SYM=*` also works for `TOP` and `TRADE`:

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

24.8.3 Step 3 — Receive snapshots

For each new `TOP`, `STATE`, `INDEX`, `DEPTH`, or `CB` subscription pair the gateway sends an immediate `SNAP`. Store the `SEQ` — it is your baseline sequence number for that stream. `TRADE` and `AUCTION` never get a baseline `SNAP`.

A wildcard `TOP` subscription (`SYM=*`) is the one exception to "one `SNAP` per pair": it produces **one `SNAP` per symbol the gateway currently knows about**, not a single `SNAP` with a literal `SYM=*`. Expect a burst of per-symbol `SNAP` lines, then live `MD` for any symbol — including symbols that only become known later — through that one subscription.

Build a per-symbol state dictionary seeded from the `SNAP`, then merge each subsequent `MD` into it.

24.8.4 Step 4 — Cancel subscriptions

```
UNSUB|CH=TRADE|SYM=MSFT
```

`UNSUB` is idempotent — removing a pair you are not subscribed to has no effect.

24.8.5 Step 5 — Handle heartbeats

When the stream is quiet the gateway sends periodic heartbeats (`HB|TS=...`), which keep the session alive on their own — a listener never has to send anything to stay connected. You can still probe with `PING`; the gateway replies `PONG|TS=…`. The connection is closed only when nothing has flowed in *either* direction for `idle_timeout_sec` seconds, which in practice means writes to your client have stopped succeeding; the gateway sends `EXIT|REASON=idle_timeout` first.

24.8.6 Step 6 — Disconnect

```
EXIT
```

24.9 Subscribing to a targeted subset

Subscribe only to what you need to minimise gateway-side fanout and parsing overhead.

Goal	`SUB` line
Best bid/ask for one symbol	`SUB|CH=TOP|SYM=AAPL`
Trade tape for one symbol	`SUB|CH=TRADE|SYM=AAPL`
Everything for one symbol	`SUB|CH=TOP,TRADE,STATE|SYM=AAPL`
Session state only (all symbols)	`SUB|CH=STATE|SYM=*`
Top-of-book for every symbol	`SUB|CH=TOP|SYM=*` (CALF `1.0.0`+)
Market-wide trade tape	`SUB|CH=TRADE|SYM=*` (CALF `1.0.0`+)
Top and trades for several symbols	`SUB|CH=TOP,TRADE|SYM=AAPL,MSFT,GOOG`
Index level	`SUB|CH=INDEX|SYM=EDU50`
Order book ladder for one symbol	`SUB|CH=DEPTH|SYM=AAPL` (CALF `1.0.0`+)
Auction results for one symbol	`SUB|CH=AUCTION|SYM=AAPL`
Market-wide auction monitor	`SUB|CH=AUCTION|SYM=*`
Circuit-breaker detail for one symbol	`SUB|CH=CB|SYM=AAPL`
Build up incrementally	Multiple `SUB` lines are cumulative

Symbol discovery

The `SYMBOLS` field in `WELCOME` lists all configured symbols as a comma-separated string. Use it instead of hard-coding symbol names.

Capability discovery

The `CH_SUPPORTED` field in `WELCOME` lists every channel this gateway build supports (e.g. `AUCTION,CB,DEPTH,INDEX,STATE,TOP,TRADE`). Check it before subscribing to `DEPTH`/`AUCTION`/`CB` or relying on the `TOP`/`TRADE` wildcard so your client degrades gracefully against an older gateway instead of handling an `ERR` reactively.

24.10 Gap detection and replay recovery

Every stream has an independent, monotonically increasing `SEQ` starting at 1. Track `last_seq[(CH, SYM)]` on every received message and check:

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

Recovery option 1 — replay within window

Reconnect, then send one `RESUME` per stream you were following:

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

The gateway replays all events with `SEQ` greater than each `LASTSEQ` that are still inside the replay window (`replay_window_sec`, default 30 s), then continues live.

Recovery option 2 — replay miss

If the requested `LASTSEQ` is older than the window the gateway sends `ERR|CODE=REPLAY_MISS|...`. On `TOP`, `STATE`, `INDEX`, `DEPTH` and `CB` a fresh `SNAP` follows: accept it and reset your local state for that stream.

On `TRADE` and `AUCTION` **no `SNAP` follows** — a past print has no current state to snapshot, so the missed events are gone for good. If you present `TRADE` as a time-and-sales record, mark the hole rather than closing the two sides over it; a record with an unmarked gap is worse than one that admits it.

The session stays open either way, so your other `RESUME` requests are unaffected.

Recovery option 3 — the gateway cannot place `LASTSEQ`

If the buffer has aged out everything after `LASTSEQ`, or `LASTSEQ` is ahead of the stream — the gateway was restarted and numbers from 1 again, or has never emitted that `(CH, SYM)` — the answer is also `ERR|CODE=REPLAY_MISS` (with a `SNAP` on the snapshot-backed channels). Zero replay lines with no error mean only that you were already up to date.

RESUME is per stream, and repeatable

Each `RESUME` recovers one `(CH, SYM)` stream, because `LASTSEQ` describes one stream's position. Send as many as you have streams. Earlier gateway builds carried this as a `RESUME=1` flag on `HELLO`, which — since `HELLO` is processed once per connection — could only ever recover a single stream; multi-stream clients had to fall back on re-`SUB` and lose the gap.

SYM=* is never valid on RESUME

Unlike `SUB`, where `SYM=*` is accepted for `TOP`/`TRADE`/`STATE`/`AUCTION`, a `RESUME|...|SYM=*` request is always rejected with `ERR|CODE=INVALID_SYMBOL` — even for those same channels. `RESUME` only ever replays one concrete symbol. The session stays open, so you can correct the request and try again.

24.11 Examples and libraries

`docs/examples/calf/` contains ready-to-run Python and C clients, plus a standalone gap-recovery library, and the package ships a production-grade Python client as `edumatcher.calf_client`. Pick based on what you're doing:

```
docs/examples/calf/
├── README.md              # what's here, and when to reach for edumatcher.calf_client instead
├── calf_parser.py         # minimal parser + serializer library (Python)
├── calf_subscriber.py     # full working subscriber example (Python)
├── calf_parser.h          # minimal parser + serializer library (C)
├── calf_parser.c
├── calf_recovery.h        # sequence tracking / gap detection / replay
├── calf_recovery.c        #   reconciliation — pure functions, no sockets
├── calf_recovery_test.c   # self-test for calf_recovery, no gateway needed
├── calf_subscriber.c      # full working subscriber example (C)
└── Makefile               # builds calf_subscriber and calf_recovery_test
```

Writing a real Python client? Don't start from the example.

`edumatcher.calf_client` ships with the package and already handles reconnect, gap repair, `REF` precision, and optional cached state. `calf_subscriber.py` is deliberately standalone — it exists to show the protocol itself, in a form that ports cleanly to a language with no library yet, not to be imported.
Each piece, in order of "how much does it do for you":

Piece	Language	What it is	When to reach for it
`edumatcher.calf_client`	Python	Production client library: reconnect/backoff, gap repair, `REF` precision, cached `MarketState`	Building a real Python bot, recorder, or dashboard backend
`calf_subscriber.py`	Python	Full example: `TOP/TRADE/STATE/DEPTH/INDEX`, gap detection **and** `RESUME` repair, `REF`-aware price formatting	Learning the protocol in Python; porting the pattern to a language with no library
`calf_subscriber.c`	C	Same coverage as the Python example, prices printed verbatim (no `REF` formatting)	Learning the protocol in C; a latency-sensitive or dependency-free environment
`calf_recovery.h/.c`	C	Just the sequence-tracking/gap/replay state machine, socket-free	Understanding or reusing the recovery logic in isolation, independent of I/O
`calf_parser.py` / `calf_parser.h+.c`	Python / C	Just the line parser/serializer (`MSGTYPE|KEY=VALUE`)	Bootstrapping a client in a new language; no channel semantics, no `REF`, no recovery

24.11.1 The production Python client: `edumatcher.calf_client`

For real Python usage — not learning the wire format — use this instead of `calf_subscriber.py`. It is the same package the engine's own tooling is built on, so it stays correct as the protocol grows.

```
from edumatcher.calf_client import CalfClient, CalfClientOptions

client = CalfClient(CalfClientOptions(symbols=["AAPL"]))
client.run(on_frame=lambda f: print(f.msg_type, f.fields))
```

It gives you two layers, and you pick per use case:

- **Raw frames** — every message, already de-duplicated and gap-checked, via the `on_frame` callback above.
- **Cached state** — `client.state` (a `MarketState`, present when `track_state=True`, the default) folds those frames into current top-of-book, depth ladder, session phase, and halt status, so you read "what is true now" instead of replaying deltas yourself:

```
def on_frame(frame):
    book = client.state.top("AAPL")
    if book:
        print(client.reference.format_price("AAPL", book.bid))

client.run(on_frame=on_frame, on_gap=lambda g: print("lost", g.count))
```

`CalfClientOptions` covers the same ground you would otherwise hand-roll: `host`/`port`/`client_name`, `channels`/`symbols`/`index_ids` to subscribe on connect and again after every reconnect, `extra_subscriptions` for additional `(channel, symbol)` pairs held across reconnects the same way — the route to per-symbol `DEPTH/CB` alongside a wildcard `TOP/TRADE/ STATE` subscription, since those can't ride one wildcard `SUB` together, `reconnect`/`reconnect_min_sec`/`reconnect_max_sec`/`connect_timeout_sec` for backoff, `ping_interval_sec` for keepalive, `track_state` to enable/disable the `MarketState` cache, `request_symbols` to send `SYMBOLS` after

the handshake (on by default, since `WELCOME|SYMBOLS=` is optional), and `auto_recover` to control whether gaps are repaired automatically or only reported via `on_gap` — set it `False` for a passive observer (a spy, a tap, a recorder) that must show the wire exactly as it is, with nothing sent upstream and nothing withheld.

24.11.2 Zero-dependency minimal client

For a quick smoke-test or a self-contained script that has no local imports:

```
import socket

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

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

TCP is a byte stream

Never assume one `recv()` equals one message. Always buffer and split on newlines as shown above.

24.11.3 Using the `calf_parser.py` library

`calf_parser.py` in `docs/examples/calf/` provides `parse_calf_line` and `build_calf_line`:

```
from calf_parser import parse_calf_line, build_calf_line, CalfMessage

# Parse a line received from the gateway
msg: CalfMessage = parse_calf_line("MD|CH=TOP|SYM=AAPL|SEQ=101|BID=150.11|BIDSZ=1400")
print(msg.msg_type)   # "MD"
print(msg.fields)     # {"CH": "TOP", "SYM": "AAPL", "SEQ": "101", ...}

# Build a line to send to the gateway
line: str = build_calf_line("SUB", {"CH": "TOP,TRADE", "SYM": "AAPL"})
# → "SUB|CH=TOP,TRADE|SYM=AAPL\n"
```

24.11.4 Annotated end-to-end subscriber

This snippet is a condensed version of `calf_subscriber.py` annotated to highlight the key CALF patterns.

```
import socket
from calf_parser import parse_calf_line, build_calf_line


class LineReader:
    """Buffer TCP bytes and yield complete CALF lines."""

    def __init__(self, sock: socket.socket) -> None:
        self.sock = sock
        self.buf = bytearray()

    def recv_line(self) -> str:
        while True:
            nl = self.buf.find(b"\n")
            if nl >= 0:
                line = bytes(self.buf[:nl])
                del self.buf[:nl + 1]
                return line.decode("utf-8", errors="replace")
            chunk = self.sock.recv(4096)
            if not chunk:
                raise RuntimeError("gateway closed connection")
            self.buf.extend(chunk)


def send(sock: socket.socket, msg_type: str, fields: dict[str, str]) -> None:
    sock.sendall(build_calf_line(msg_type, fields).encode())


with socket.create_connection(("127.0.0.1", 5570), timeout=5) as sock:
    reader = LineReader(sock)

    # Authenticate
    send(sock, "HELLO", {"CLIENT": "mybot", "PROTO": "CALF1"})
    welcome = parse_calf_line(reader.recv_line())
    assert welcome.msg_type == "WELCOME", f"unexpected: {welcome}"
    known_symbols = welcome.fields.get("SYMBOLS", "").split(",")
    print(f"Connected. Gateway knows: {known_symbols}")

    # Subscribe
    send(sock, "SUB", {"CH": "TOP,TRADE", "SYM": "AAPL,MSFT"})
    send(sock, "SUB", {"CH": "STATE", "SYM": "*"})

    # Per-stream state
    top: dict[str, dict[str, str]] = {}           # symbol → current top fields
    last_seq: dict[tuple[str, str], int] = {}     # (CH, SYM) → last seen SEQ

    while True:
        msg = parse_calf_line(reader.recv_line())

        if msg.msg_type in ("MD", "TRADE", "STATE", "IDX", "DEPTH", "SNAP"):
            ch  = msg.fields.get("CH", "")
            sym = msg.fields.get("SYM", "")
            seq = int(msg.fields.get("SEQ", "0"))

            # Gap check. A SNAP re-baselines and is never a gap; a SEQ at or
            # below the baseline is a replayed duplicate unless it falls in a
            # range you actually asked RESUME to backfill.
            prev = last_seq.get((ch, sym))
            if msg.msg_type == "SNAP":
                pass                                # baseline: just record it
            elif prev is not None and seq <= prev:
                continue                            # already seen; drop it
            elif prev is not None and seq != prev + 1:
                print(f"GAP on ({ch},{sym}): expected {prev + 1}, got {seq}")
                # → recover in place: RESUME|CH=..|SYM=..|LASTSEQ=<prev>
            last_seq[(ch, sym)] = seq
```

24.11.5 Run the bundled examples

`calf_subscriber.py` has no `--channels` flag — the channel set is fixed by design (it demonstrates the Cartesian-subscribe pattern, not a configurable client): it always subscribes to `TOP,TRADE,STATE,DEPTH` for the symbols you pass, filtered down to whatever `WELCOME|CH_SUPPORTED=` actually advertises, plus a separate session-wide `STATE|SYM=*` unless you pass `--no-state-wildcard`. `INDEX` is opt-in via `--index`, since it needs an index id rather than a symbol.

```
cd docs/examples/calf

# Default run: TOP/TRADE/STATE/DEPTH for one symbol, plus session-wide STATE
python3 calf_subscriber.py --host 127.0.0.1 --port 5570 --symbols AAPL

# Several symbols
python3 calf_subscriber.py --symbols AAPL,MSFT

# Also subscribe an index feed (skipped with a warning if INDEX isn't advertised)
python3 calf_subscriber.py --symbols AAPL --index EDU100

# Skip the extra session-wide STATE|SYM=* subscription
python3 calf_subscriber.py --symbols AAPL --no-state-wildcard

# Send one explicit RESUME after the handshake, to see replay in isolation
# (gaps noticed while running are always resumed automatically, regardless
# of this flag)
python3 calf_subscriber.py --resume --resume-ch TOP --resume-sym AAPL --lastseq 1042
```

Run `python3 calf_subscriber.py --help` for the full flag list.

For a C client (useful for latency-sensitive or dependency-free environments — it prints wire values verbatim and needs no `REF` handling):

```
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 and, if omitted, the `INDEX` subscription is skipped entirely. Press Ctrl-C for a clean shutdown — the client traps `SIGINT` and closes the socket instead of being killed mid-syscall.

24.11.6 The recovery library in isolation: `calf_recovery`

Both the Python and C examples detect a `SEQ` gap and repair it with `RESUME` — not just notice it. In the C example that logic is factored out into `calf_recovery.h/.c`: a small state machine with no sockets, threads, or timers, built around the same three rules documented under Gap detection and replay recovery above (replay overlaps live traffic; a `SNAP` re-baselines and is never a gap; sequence never moves backward within one connection). Feed it `(msg_type, CH, SYM, SEQ)` for each inbound line and it tells you whether to process the message and whether to send a `RESUME`:

```
calf_recovery_t rec;
calf_recovery_init(&rec);
calf_recovery_new_connection(&rec);          // after each connect

calf_gap_t gap;
calf_action_t act = calf_recovery_observe(&rec, msg.msg_type,
                                          ch, sym, seq, &gap);
if (act == CALF_DROP) continue;              // replayed duplicate
if (act == CALF_RESUME) {
    char line[CALF_RESUME_LINE_LEN];
    calf_recovery_build_resume(line, sizeof(line), &gap);
    send_line(fd, line);
}
if (act == CALF_GAP_UNREPAIRABLE) {
    // no RESUME will fill this: TRADE/AUCTION have no SNAP, or the hole is
    // outside the replay window. Surface it -- an unmarked hole is worse
    // than one that admits it -- then use the message.
}
// CALF_PROCESS, or after any of the above: use the message
```

A fourth outcome, `CALF_GAP_UNREPAIRABLE`, means the hole cannot be closed at all — either the channel has no replay path (`TRADE`/`AUCTION`) or the gap is already outside the replay window. Don't fold it silently into the same handling as `CALF_PROCESS`.

Because it takes no socket, it has its own self-test that runs without a gateway:

```
cd docs/examples/calf
make test
```

Worth reading even if you're not writing C: the header comments in `calf_recovery.h` are a second, denser statement of the same recovery rules `calf_subscriber.py`'s `SequenceTracker` class implements in Python — seeing the same logic in two languages is often what makes it click.

24.12 Common errors and fixes

Error code	Typical cause	Action
`AUTH_REQUIRED`	`SUB` sent before `HELLO`, or `HELLO` with a missing `CLIENT` or wrong/missing `PROTO`	Send `HELLO|CLIENT=…|PROTO=CALF1` first
`INVALID_CHANNEL`	Unknown `CH` value	Use `TOP`, `TRADE`, `STATE`, `INDEX`, `DEPTH`, `AUCTION`, or `CB`
`INVALID_SYMBOL`	Unknown symbol; or `SYM=*` used with `INDEX`/`DEPTH`/`CB`; or `SYM=*` used at all on `RESUME`	Use configured symbols; `SYM=*` only for `STATE`/`TOP`/`TRADE`/`AUCTION` on `SUB` — never on `RESUME`
`SUB_LIMIT`	Too many subscribed symbols	Reduce requested symbol set
`RATE_LIMITED`	Client exceeded `max_messages_per_second`	Slow down the send rate; connection stays open
`REPLAY_MISS`	Requested replay is outside buffer window	On `TOP`/`STATE`/`INDEX`/`DEPTH`/`CB`, accept the fresh `SNAP` and reset the baseline. On `TRADE`/`AUCTION` none follows — the events are gone; mark the gap
`SLOW_CLIENT`	Client cannot drain the outbound stream fast enough	Backlog is dropped, this `ERR` is sent, then the connection is closed — reconnect and process faster
`BAD_MESSAGE`	Malformed or oversized line (> 4096 bytes)	Fix line syntax/framing

Connection refused with no CALF response at all

If the gateway is already at `max_connections`, it accepts the TCP socket and closes it immediately — there is no `HELLO`/`WELCOME` exchange and no `ERR` line, so this looks like a bare connection failure, not a protocol error. Treat an immediate disconnect right after connecting (before any `WELCOME`) as a possible sign the gateway is at its connection cap.

24.13 Operational checklist

1. Confirm `pm-engine` is running and publishing (`pm-engine --verbose`)
2. Confirm `pm-md-gwy` is running
3. Confirm TCP port is reachable (`nc 127.0.0.1 5570`)
4. Confirm `HELLO` receives `WELCOME`
5. Confirm `SUB` receives expected `SNAP` and live flow
6. Track `SEQ` per stream; on reconnect send one `RESUME` per stream with `LASTSEQ`
7. On `REPLAY_MISS`: accept the recovery `SNAP` and reset local state — except on `TRADE`/`AUCTION`, where none is sent and the gap is permanent
8. Drop any replayed message at or below the `SEQ` you already recorded: `RESUME` returns everything past `LASTSEQ`, duplicates included

24.14 Building tools on top of the feed

CALF is deliberately trivial to consume, which makes it a great base for small, satisfying tools. Each of these is well under a hundred lines on top of the patterns already shown:

Tool idea	Channels	Sketch
Time-and-sales tape	`TRADE` (`SYM=*`)	print each `TRADE`, colour by `SIDE`, tally volume
Live price board	`TOP` (`SYM=*`)	keep a `symbol → top` dict, redraw a table on each `MD`
DOM / ladder widget	`DEPTH`	replace the ladder on each `DEPTH` line; render bids/asks
VWAP / OHLCV recorder	`TRADE`	accumulate $\Sigma(\text{px} \cdot \text{qty})$ and Σqty per symbol per interval
CSV / Parquet recorder	`TRADE`, `TOP`	append every line to disk for replay and back-testing
Halt-aware alert bot	`STATE`	alert on `SESSION=HALTED`, clear on resume
Index dashboard	`INDEX`	plot `LEVEL` / `PCTCHG` across the day
Auction monitor	`AUCTION` (`SYM=*`)	print each uncross; flag large `IMBQTY` remainders
Circuit-breaker panel	`CB`	show `STATUS`/`LEVEL`/`RESUMEAT` per halted symbol, clear on resume

24.14.1 Anatomy of a robust client

Whatever you build, the same six habits keep it correct:

1. **Frame properly** — buffer TCP bytes and split on `\n`; one `recv()` is *not* one message.
2. **Handshake first** — send `HELLO`, wait for `WELCOME`, then read `SYMBOLS` and `CH_SUPPORTED` before subscribing.
3. **Seed then update** — initialise state from the `SNAP`, then merge each `MD` (TOP) or replace the ladder on each `DEPTH`.
4. **Watch `SEQ`** — track it per `(CH, SYM)`; on a gap, reconnect and `RESUME` or re-`SUB` for a fresh `SNAP`.
5. **Stay alive** — treat `HB` as a liveness signal, answer nothing; treat `ERR|CODE=SLOW_CLIENT` as a sign to reconnect and consume faster, and `EXIT|REASON=…` as the gateway ending the session.
6. **Degrade gracefully** — if `CH_SUPPORTED` lacks `DEPTH`/`INDEX`/`AUCTION`/`CB`, fall back instead of subscribing blindly.

24.14.2 Browser clients need a server-side bridge

A web browser **cannot open a raw TCP socket**, so browser JavaScript cannot connect to `pm-md-gwy` on `:5570` directly — browsers only speak HTTP, WebSocket, and WebRTC. You have two options:

1. **Use the built-in WebSocket feed.** The API Gateway already exposes streaming market data as JSON over WebSocket — the browser-native path, no extra code to write.
2. **Write a small bridge** that holds one TCP CALF connection server-side and relays each line to the browser over WebSocket. This keeps the exact CALF stream and is an excellent learning project.



A minimal bridge stub (Python, `asyncio` + `websockets`) — enough to stream one subscription to any number of browser tabs:

```
import asyncio, websockets   # pip install websockets

CALF_HOST, CALF_PORT = "127.0.0.1", 5570
SUB = b"HELLO|CLIENT=bridge|PROTO=CALF1\nSUB|CH=TOP,TRADE|SYM=*\n"

async def bridge(ws):
    # One upstream CALF connection per browser client (simple 1:1 model).
    reader, writer = await asyncio.open_connection(CALF_HOST, CALF_PORT)
    writer.write(SUB)
    await writer.drain()
    try:
        buf = bytearray()
        while True:
            chunk = await reader.read(4096)
            if not chunk:
                break
            buf.extend(chunk)
            while b"\n" in buf:
                nl = buf.index(b"\n")
                line = bytes(buf[:nl]).decode("utf-8", "replace")
                del buf[:nl + 1]
                if line:
                    await ws.send(line)          # forward CALF line to the browser
    finally:
        writer.close()

async def main():
    async with websockets.serve(bridge, "127.0.0.1", 8080):
        await asyncio.Future()   # run forever

asyncio.run(main())
```

The browser then consumes it with a few lines:

```
const ws = new WebSocket("ws://127.0.0.1:8080");
ws.onmessage = (e) => {
  const [type, ...kv] = e.data.split("|");
  const f = Object.fromEntries(kv.map(p => p.split("=")));
  if (type === "TRADE") console.log(`${f.SYM} ${f.QTY}@${f.PX} ${f.SIDE}`);
};
```

Production hardening

The stub keeps one upstream connection **per** browser client for clarity. A real bridge would hold a **single** CALF connection and fan its lines out to all connected browsers, apply the same `SEQ`/`RESUME` recovery described above, and authenticate the WebSocket side. Treat it as a starting point, not a finished service.

24.15 See also

In this book:

- `docs/examples/calf/README.md` — the examples directory's own guide to what's there and which client to start from
- Risk Controls — circuit-breaker mechanics behind the `CB` channel and `STATE` halt/resume events
- API Gateway — REST and WebSocket market data alternative
- Post-Trade Dissemination (RALF) — fills and post-trade events

Reference Manual:

- Processes — where `pm-md-gwy` sits in the process model

Protocols and Clients:

- External Protocols Overview — ALF, BALF, CALF, RALF at a glance
- Appendix — CALF Protocol — normative wire format, full field tables, sequencing rules, including the `AUCTION` and `CB` message definitions
- CALF Protocol Spy (pm-calf-spy) — read-only CLI for watching the raw feed, human-readable or JSON
- Messages Reference — CALF messages in the full message catalogue

CHAPTER 25

RALF Gateway — Post-Trade Dissemination

Learning objectives

After reading this page you will understand:

- what `pm-ralf-gwy` does and where it sits in the EduMatcher architecture
- what data is available on each channel (`CLEARING`, `DROP_COPY`, `AUDIT`)
- when to choose RALF over the other available protocols
- how to start the gateway and verify connectivity from a terminal
- how to subscribe and receive post-trade events
- how replay and reconnect behavior works and when to use `LASTSEQ`
- how to write a working Python subscriber using the library in `docs/examples/ralf/`
- which operational checks to use when debugging connectivity problems

25.1 What this process does

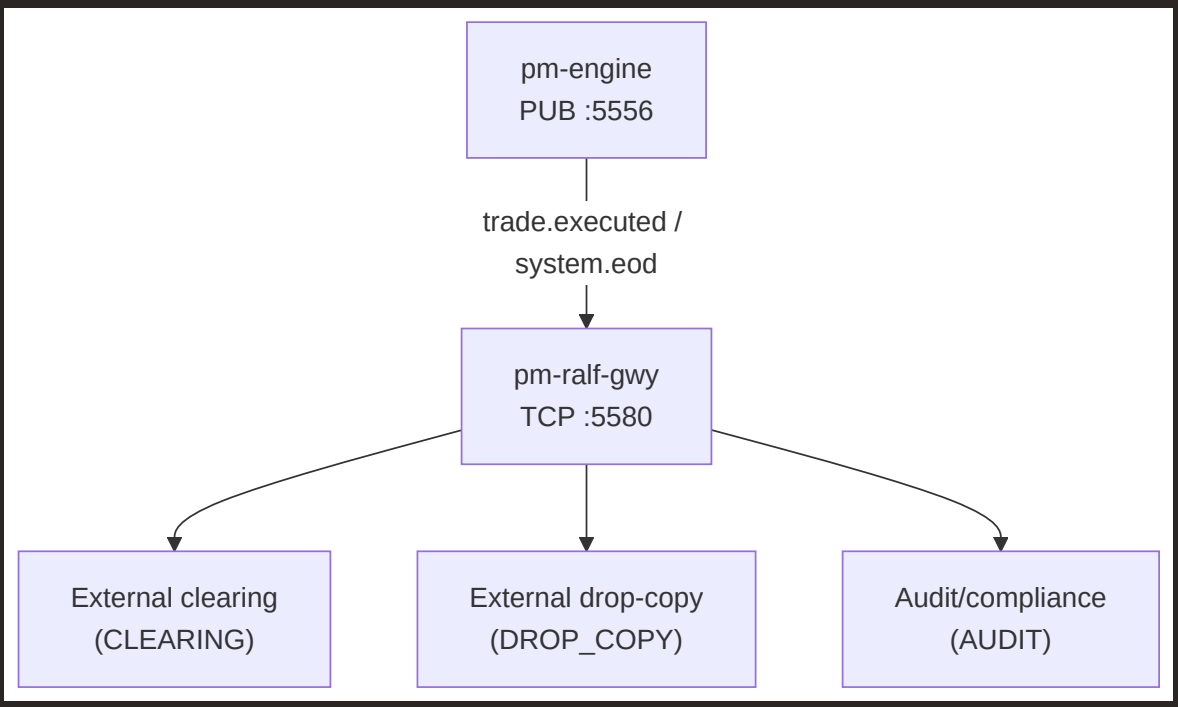
`pm-ralf-gwy` is the RALF post-trade dissemination gateway. The matching engine publishes trade and end-of-day events on an internal ZeroMQ PUB socket (`:5556`) that external systems cannot reach directly. `pm-ralf-gwy` bridges that gap: it subscribes to the engine PUB socket, translates raw events into RALF lines, sequences them, and re-publishes them over TCP (default port `5580`) to authenticated external clients.

RALF is intended for external parties such as:

- clearing and reconciliation systems
- risk and compliance drop-copy consumers
- audit and surveillance tooling

Unlike the interactive `pm-alf-console`, `pm-ralf-gwy` is not used for order entry. It is a read-only event dissemination process.

25.2 Architecture position



Responsibilities of `pm-ralf-gwy`:

- translates engine events into RALF lines per channel
- assigns **per-channel sequence numbers** — one counter per channel, shared across every symbol on that channel
- keeps a **time-bounded replay journal** so reconnecting clients can recover missed messages without reprocessing raw engine logs
- sends an automatic **baseline snapshot** (`SNAP`) when a client subscribes
- enforces role-based channel access and disconnects slow clients

25.3 Prerequisites

- `pm-engine` running and publishing events
- `pm-ralf-gwy` running
- symbols configured in `engine_config.yaml`

25.4 Quick start

25.4.1 1. Start the engine

```
pm-engine --verbose
```

25.4.2 2. Start the RALF gateway

```
pm-ralf-gwy
```

By default it reads `engine_config.yaml` from the same resolution path used by other `pm-*` processes.

25.4.3 3. Connect an external client

```
HELLO|CLIENT=ext01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=0
SUB|CH=CLEARING|SYM=*
```

The gateway responds with `WELCOME` and `SNAP`, then streams live `EXEC`/`EOD` messages according to subscriptions.

25.5 Configuration

`pm-ralf-gwy` supports an optional `post_trade_gateway` block in `engine_config.yaml`:

```
post_trade_gateway:
  name: "ralf-gwy01"
  bind_address: "0.0.0.0"
  port: 5580
  replay_retention_sec: 86400
  heartbeat_interval_sec: 2
  idle_timeout_sec: 20
  max_connections: 64
  max_client_queue: 10000
  allowed_roles:
    - CLEARING
    - DROP_COPY
    - AUDIT
```

CLI overrides:

```
pm-ralf-gwy --bind 127.0.0.1 --port 5580 --engine-pub tcp://127.0.0.1:5556
```

Option	Default	Description
`--bind ADDR`	from config / `0.0.0.0`	Override TCP bind address
`--port PORT`	from config / `5580`	Override TCP listen port
`--engine-pub ADDR`	from config / `tcp://127.0.0.1:5556`	Override engine PUB socket address
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	Reduce output to warnings/errors
`--log-target`	auto	Where operational log records go: `server` (default, auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file PATH`	—	Operational log file path — required when `--log-target file`
`--log-failover-timeout SECONDS`	`30`	Grace window before falling back to a local log file once `pm-log-srv` becomes unreachable

25.6 Generate config with `pm-config-gen`

If you do not want to write the `post_trade_gateway` block by hand, generate it alongside the main engine config with `pm-config-gen`:

```
pm-config-gen \
  --symbols AAPL MSFT \
  --participants TRADER01 TRADER02 OPS01:ADMIN \
  --sessions-enabled \
  --post-trade-gateway \
  --post-trade-bind-address 127.0.0.1 \
  --post-trade-port 5580 \
  --post-trade-replay-retention-sec 3600 \
  --post-trade-heartbeat-interval-sec 1 \
  --post-trade-idle-timeout-sec 10 \
  --post-trade-max-client-queue 2000 \
  --post-trade-allowed-roles CLEARING AUDIT \
  --output engine_config.yaml
```

This produces a normal `engine_config.yaml` for `pm-engine` plus a top-level `post_trade_gateway` block for `pm-ralf-gwy`.

Expected generated section:

```
post_trade_gateway:
  name: ralf-gwy01
  bind_address: 0.0.0.0
  port: 5580
  replay_retention_sec: 3600
  heartbeat_interval_sec: 2
  idle_timeout_sec: 20
  max_connections: 64
  max_client_queue: 2000
  allowed_roles:
    - CLEARING
    - AUDIT
```

Use `127.0.0.1` for a single-host lab. If external consumers must connect from other machines, change the bind address to a controlled network-facing value such as `0.0.0.0` and restrict access at the network boundary.

25.7 What data is available

The gateway exposes three channels. Each targets a different class of external consumer. A client's `ROLE` (declared in `HELLO`) determines which channels it may subscribe to.

25.7.1 Channel `CLEARING` — post-trade clearing data

`CLEARING` carries one `EXEC` line per matched trade plus an `EOD` summary for each symbol at end of day. It is the primary channel for clearing and reconciliation systems.

When you get it: Subscribe with `CH=CLEARING|SYM=*` (or a specific symbol). The gateway immediately sends a `SNAP` baseline, then streams `EXEC` events as trades occur and `EOD` events at end of day.

Typical use cases: trade reconciliation, position keeping, clearing obligation calculation.

Wire example:

```
SNAP|CH=CLEARING|SYM=*|SEQ=0|TS=2026-06-30T09:30:00.000Z
EXEC|CH=CLEARING|SYM=AAPL|SEQ=1|TS=2026-06-30T09:30:01.100Z|EXEC_ID=E001|MATCH_ID=E001|BUY_ORDER_ID=O100|SELL_ORDER_ID=O101|BUY_GW=gw
EXEC|CH=CLEARING|SYM=AAPL|SEQ=2|TS=2026-06-30T09:31:00.000Z|EXEC_ID=E002|MATCH_ID=E002|BUY_ORDER_ID=O102|SELL_ORDER_ID=O103|BUY_GW=gw
EOD|CH=CLEARING|SYM=AAPL|SEQ=3|TS=2026-06-30T16:00:00.000Z|TRADE_COUNT=2|EXEC_COUNT=2
```

25.7.2 Channel `DROP_COPY` — fill events for risk monitoring

`DROP_COPY` carries the same `EXEC` events as `CLEARING` but is aimed at risk and compliance subscribers who need a real-time feed of all fills across gateways.

When you get it: Subscribe with `CH=DROP_COPY|SYM=<symbol>` or a symbol list. A `SNAP` baseline is sent on subscription.

Typical use cases: real-time position risk, pre-trade limit monitoring, compliance surveillance.

Wire example:

```
SNAP|CH=DROP_COPY|SYM=AAPL|SEQ=0|TS=2026-06-30T09:30:00.000Z
EXEC|CH=DROP_COPY|SYM=AAPL|SEQ=1|TS=2026-06-30T09:30:01.100Z|EXEC_ID=E001|MATCH_ID=E001|BUY_ORDER_ID=O100|SELL_ORDER_ID=O101|BUY_GW=gw
```

25.7.3 Channel `AUDIT` — full audit trail

`AUDIT` carries all post-trade events and is intended for audit and surveillance consumers. A client with `ROLE=AUDIT` may subscribe to any channel.

When you get it: Subscribe with `CH=AUDIT|SYM=*` for the full trail.

Typical use cases: regulatory reporting, trade reconstruction, end-of-day audit file generation.

Wire example:

```
SNAP|CH=AUDIT|SYM=*|SEQ=0|TS=2026-06-30T09:30:00.000Z
EXEC|CH=AUDIT|SYM=AAPL|SEQ=1|TS=2026-06-30T09:30:01.100Z|EXEC_ID=E001|MATCH_ID=E001|BUY_ORDER_ID=O100|SELL_ORDER_ID=O101|BUY_GW=gw01|S
EOD|CH=AUDIT|SYM=AAPL|SEQ=2|TS=2026-06-30T16:00:00.000Z|TRADE_COUNT=1|EXEC_COUNT=1
```

25.7.4 Channel summary

Channel	Message types	Baseline `SNAP`?	`SYM=*` wildcard?	Primary consumer
`CLEARING`	`EXEC`, `EOD`	Yes	Yes	Clearing and reconciliation
`DROP_COPY`	`EXEC`	Yes	Yes	Risk, compliance drop-copy
`AUDIT`	`EXEC`, `EOD`	Yes	Yes	Audit, surveillance

25.8 When to use RALF — protocol comparison

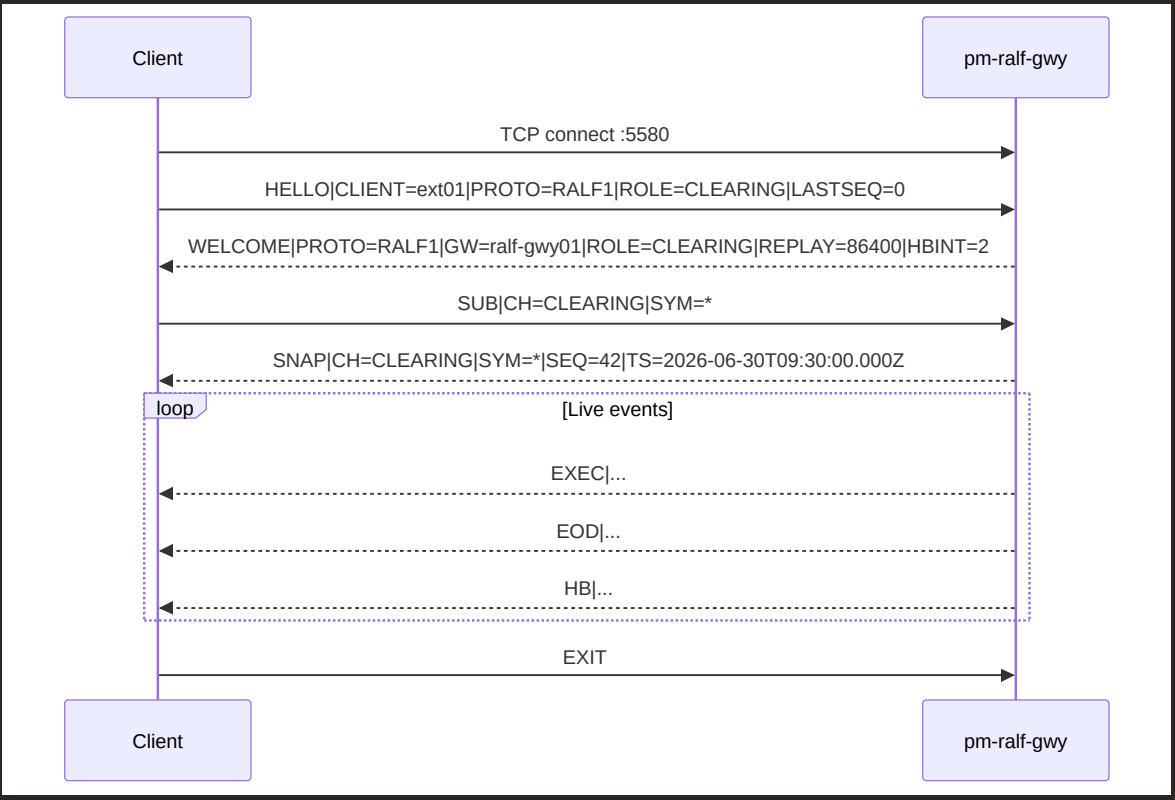
EduMatcher offers several ways to obtain market and post-trade data.

Approach	Transport	Best for	Not suitable for
RALF (`pm-ralf-gwy`)	TCP text	External clearing, drop-copy, audit consumers	Pre-trade market data; top-of-book
CALF (`pm-md-gwy`)	TCP text	External market data (TOP, TRADE, STATE, INDEX)	Post-trade events
Internal ZMQ PUB (`:5556`)	ZMQ binary	Internal Python processes	External clients
REST / WebSocket (`pm-api-gwy`)	HTTP/JSON	Web dashboards; one-shot queries	High-frequency streaming

See External Protocols Overview for the full comparison table.

25.9 Connecting and subscribing

Every RALF session follows this sequence:



25.9.1 Step 1 — Send `HELLO`

```
HELLO|CLIENT=ext01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=0
```

`CLIENT` is a free-text identifier (max 32 chars) used in gateway logs. `PROTO` must be exactly `RALF1`. `ROLE` declares your access tier — `CLEARING`, `DROP_COPY`, or `AUDIT`. `LASTSEQ=0` means start from the current live position (no replay). The gateway replies with `WELCOME` or closes the connection on protocol or entitlement error.

A connection that sends no `HELLO` within 5 seconds receives `EXIT|REASON=auth_timeout` and is closed — the same deadline in all three line gateways (`pm-md-gwy`, `pm-ralf-gwy`, `pm-dc-gwy`). Lines a client sends after the one that ended its session (a `HELLO` pipelined behind a refused first line, for example) are ignored.

The `REPLAY` field in `WELCOME` tells you the journal retention window in seconds — use it to decide how far back you can safely request on reconnect.

25.9.2 Step 2 — Subscribe

```
SUB|CH=CLEARING|SYM=*
```

Multiple channels and symbols are comma-separated:

```
SUB|CH=CLEARING,DROP_COPY|SYM=AAPL,MSFT
```

The subscription is the Cartesian product of channels × symbols. A client may only subscribe to channels that match its declared role, except `AUDIT` clients who may subscribe to any channel. Multiple `SUB` lines are cumulative.

25.9.3 Step 3 — Receive the snapshot

For each `SUB` line, the gateway sends **one** immediate `SNAP` covering the whole requested channel/symbol set (not one per `(channel, symbol)` pair). Store the `SEQ` — it is your baseline sequence number for that channel.

25.9.4 Step 4 — Cancel subscriptions

```
UNSUB|CH=DROP_COPY|SYM=AAPL
```

`UNSUB` is idempotent.

25.9.5 Step 5 — Handle heartbeats

When the stream is idle the gateway sends periodic `HB|TS=...` lines. You can probe with `PING`; the gateway replies `PONG|TS=…`. The idle timeout counts traffic in both directions, so a client that keeps reading stays connected on heartbeats alone; only when writes to it have stopped succeeding for `idle_timeout_sec` seconds does the gateway send `EXIT|REASON=idle_timeout` and close the connection.

A client that stops reading is dropped too. When its outbound queue exceeds `max_client_queue` the gateway discards the backlog (a reconnect with `LASTSEQ` replays it), sends `ERR|CODE=SLOW_CLIENT`, and closes the connection. Either final line is delivered if the client is still reading; the connection is closed after at most 5 seconds whether or not it was.

25.9.6 Step 6 — Disconnect

```
EXIT
```

25.10 Gap detection and replay

Every **channel** has an independent, monotonically increasing `SEQ` starting at 1 — shared across all symbols on that channel, not scoped per symbol. Track `last_seq[CH]` on every received message and check:

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

Recovery option 1 — replay within window

Reconnect with a non-zero `LASTSEQ` in `HELLO`:

```
HELLO|CLIENT=ext01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=1200
```

The gateway replays all journal events with `SEQ > 1200` that are still inside the retention window (`replay_retention_sec`, default 86400 s — one full trading day), then continues live.

Recovery option 2 — replay miss

If the requested `LASTSEQ` is older than the retained journal the gateway sends `ERR|CODE=REPLAY_MISS|...` followed by a fresh `SNAP`. Accept the `SNAP` and reset your local state for that stream.

Note

RALF's replay journal covers a full trading day by default (86 400 s), making intraday reconnects lossless in normal operation.

25.11 Quick connect test

Use `nc` (or `telnet`) to validate the line protocol from the command line before writing any code:

Prefer a ready-made client?

`pm-ralf-spy` does everything below for you — handshake, subscribe, and pretty-print every line — without hand-typing protocol messages: `pm-ralf-spy --role CLEARING`.

```
nc 127.0.0.1 5580
```

Then type:

```
HELLO|CLIENT=ops01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=0
SUB|CH=CLEARING|SYM=*
```

Expected response pattern:

1. `WELCOME|PROTO=RALF1|GW=ralf-gwy01|...` — session open
2. `SNAP|CH=CLEARING|SYM=*|SEQ=...|TS=...` — baseline snapshot
3. `EXEC|...` when a trade executes
4. `EOD|...` at end of day
5. `HB|...` when the stream is idle

25.12 Python subscriber example

The `docs/examples/ralf/` directory contains ready-to-run Python and C libraries.

```
docs/examples/ralf/
├── ralf_parser.py        # parser + serializer library
├── ralf_subscriber.py    # full working subscriber example
├── ralf_parser.h         # C parser library
├── ralf_parser.c
├── ralf_subscriber.c     # C subscriber example
└── Makefile
```

25.12.1 Zero-dependency minimal client

For a quick smoke-test or a self-contained script with no local imports:

```
import socket

sock = socket.create_connection(("127.0.0.1", 5580))
sock.sendall(b"HELLO|CLIENT=bot01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=0\n")
sock.sendall(b"SUB|CH=CLEARING|SYM=*\n")

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

TCP is a byte stream

Never assume one `recv()` equals one message. Always buffer and split on newlines as shown above.

25.12.2 Using the `ralf_parser.py` library

`ralf_parser.py` in `docs/examples/ralf/` provides `parse_ralf_line` and `build_ralf_line`:

```
from ralf_parser import parse_ralf_line, build_ralf_line, RalfMessage

# Parse a line received from the gateway
msg: RalfMessage = parse_ralf_line("EXEC|CH=CLEARING|SYM=AAPL|SEQ=1|PX=150.12|QTY=200|SIDE=BUY")
print(msg.msg_type)   # "EXEC"
print(msg.fields)     # {"CH": "CLEARING", "SYM": "AAPL", "SEQ": "1", ...}

# Build a line to send to the gateway
line: str = build_ralf_line("SUB", {"CH": "CLEARING", "SYM": "*"})
# → "SUB|CH=CLEARING|SYM=*\n"
```

25.12.3 Annotated end-to-end subscriber

This snippet is a condensed version of `ralf_subscriber.py` annotated to highlight the key RALF patterns.

```
import socket
from ralf_parser import parse_ralf_line, build_ralf_line


class LineReader:
    """Buffer TCP bytes and yield complete RALF lines."""

    def __init__(self, sock: socket.socket) -> None:
        self.sock = sock
        self.buf = bytearray()

    def recv_line(self) -> str:
        while True:
            nl = self.buf.find(b"\n")
            if nl >= 0:
                line = bytes(self.buf[:nl])
                del self.buf[:nl + 1]
                return line.decode("utf-8", errors="replace")
            chunk = self.sock.recv(4096)
            if not chunk:
                raise RuntimeError("gateway closed connection")
            self.buf.extend(chunk)


def send(sock: socket.socket, msg_type: str, fields: dict[str, str]) -> None:
    sock.sendall(build_ralf_line(msg_type, fields).encode())


with socket.create_connection(("127.0.0.1", 5580), timeout=5) as sock:
    reader = LineReader(sock)

    # Authenticate with role declaration
    send(sock, "HELLO", {"CLIENT": "mybot", "PROTO": "RALF1", "ROLE": "CLEARING", "LASTSEQ": "0"})
    welcome = parse_ralf_line(reader.recv_line())
    assert welcome.msg_type == "WELCOME", f"unexpected: {welcome}"
    replay_sec = welcome.fields.get("REPLAY", "?")
    print(f"Connected. Journal retention: {replay_sec}s")

    # Subscribe — role must match allowed channels
    send(sock, "SUB", {"CH": "CLEARING", "SYM": "*"})

    # Per-channel sequence tracking — SEQ is shared across all symbols on
    # one channel, not scoped per (channel, symbol) pair.
    last_seq: dict[str, int] = {}   # CH → last seen SEQ

    while True:
        msg = parse_ralf_line(reader.recv_line())

        if msg.msg_type in ("EXEC", "EOD", "SNAP"):
            ch  = msg.fields.get("CH", "")
            sym = msg.fields.get("SYM", "")
            seq = int(msg.fields.get("SEQ", "0"))

            # Gap check — reconnect with LASTSEQ={prev} to recover
            prev = last_seq.get(ch)
            if prev is not None and seq != prev + 1:
                print(f"GAP on {ch}: expected {prev + 1}, got {seq}")
            last_seq[ch] = seq

            if msg.msg_type == "SNAP":
                print(f"SNAP  {ch}/{sym}: baseline SEQ={seq}")

            elif msg.msg_type == "EXEC":
                print(
                    f"EXEC  {sym}: PX={msg.fields['PX']} "
                    f"QTY={msg.fields['QTY']} SIDE={msg.fields['SIDE']} "
```

25.12.4 Run the bundled examples

```
cd docs/examples/ralf

# Subscribe as CLEARING role for all symbols
python3 ralf_subscriber.py --host 127.0.0.1 --port 5580 \
    --role CLEARING --channels CLEARING --symbols '*'

# Drop-copy subscriber for specific symbols
python3 ralf_subscriber.py --role DROP_COPY --channels DROP_COPY \
    --symbols AAPL,MSFT

# Reconnect with replay from last known sequence
python3 ralf_subscriber.py --role CLEARING --channels CLEARING \
    --symbols '*' --lastseq 1200
```

For a C client (useful for latency-sensitive environments):

```
cd docs/examples/ralf && make
./ralf_subscriber 127.0.0.1 5580
```

25.13 Common errors and fixes

Error code	Typical cause	Action
`AUTH_REQUIRED`	Any message before a successful `HELLO`, or `HELLO` itself missing `CLIENT`/`PROTO`/`ROLE`	Send a well-formed `HELLO` first
`ENTITLEMENT_DENIED`	Role not permitted for that channel	Use the channel matching your role; `AUDIT` may access all
`INVALID_CHANNEL`	Unknown `CH` value	Use `CLEARING`, `DROP_COPY`, or `AUDIT`
`REPLAY_MISS`	Requested `LASTSEQ` is outside the journal window, or ahead of it after a gateway restart	Accept the recovery `SNAP` and reset local baseline
`SLOW_CLIENT`	Client cannot drain the outbound stream fast enough	Reconnect and process faster; terminal error
`BAD_MESSAGE`	Malformed line (parse failure, unsupported message type, or non-integer `LASTSEQ`)	Fix line syntax/framing

25.14 Operational notes

- Keep `pm-ralf-gwy` separate from human-facing order-entry gateways.
- Use role-based channel policy: each role should subscribe only to its matching channel. `AUDIT` clients may subscribe to any channel.
- Monitor slow-client behavior and queue pressure (`SLOW_CLIENT` errors).
- For local labs, bind to `127.0.0.1`; for multi-host deployments use a controlled network interface and perimeter security controls.
- The default replay retention (86 400 s) covers one full trading day. Tune `replay_retention_sec` for your memory budget.

25.15 Dedicated Gateway Runbook (pm-ralf-gwy)

Use this section as a focused operator runbook for the running gateway process.

25.15.1 Start commands

Installed mode:

```
pm-engine --verbose
pm-ralf-gwy
```

Developer mode:

```
poetry run pm-engine --verbose
poetry run pm-ralf-gwy
```

25.15.2 Minimal client probe

Manual probe using `nc`:

```
nc 127.0.0.1 5580
```

Then send:

```
HELLO|CLIENT=clear01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=0
SUB|CH=CLEARING|SYM=*
```

Expected sequence:

1. `WELCOME|...`
2. `SNAP|...`
3. live `EXEC|...` / `EOD|...`
4. periodic `HB|...` while idle

25.15.3 Reconnect replay behavior

Resume from checkpoint:

```
HELLO|CLIENT=clear01|PROTO=RALF1|ROLE=CLEARING|LASTSEQ=1200
```

Outcomes:

- replay hit: retained events with `SEQ > LASTSEQ`, then live continuation
- replay miss: `ERR|CODE=REPLAY_MISS`, followed by recovery `SNAP`

25.15.4 Fast error triage

Error code	Typical cause	Action
`AUTH_REQUIRED`	Any message before successful `HELLO`, or an incomplete `HELLO`	Authenticate first
`ENTITLEMENT_DENIED`	Role blocked by policy	Use an allowed role or update config
`INVALID_CHANNEL`	Unsupported `CH` value	Use `CLEARING`, `DROP_COPY`, `AUDIT`
`REPLAY_MISS`	Replay point outside retention	Accept `SNAP` and reset local baseline
`SLOW_CLIENT`	Client too slow to drain stream	Reconnect and increase consume throughput
`BAD_MESSAGE`	Malformed line syntax	Fix message format

25.15.5 Operator checklist

1. Confirm engine is publishing post-trade events
2. Confirm `pm-ralf-gwy` is running
3. Confirm TCP reachability (`nc 127.0.0.1 5580`)
4. Confirm `HELLO` receives `WELCOME`
5. Confirm `SUB` receives `SNAP` and live flow
6. Track `SEQ`; on reconnect use `LASTSEQ`

25.16 See also

In this book:

- Market Data Feed (CALF) — pre-trade streaming market data

Reference Manual:

- Processes

Protocols and Clients:

- External Protocols Overview — ALF, BALF, CALF, RALF at a glance
- Appendix — RALF Protocol — normative wire format and field tables
- RALF Protocol Spy (pm-ralf-spy) — read-only CLI for watching the raw feed, human-readable or JSON
- Drop Copy

CHAPTER 26

API Gateway (REST/WebSocket)

Learning objectives

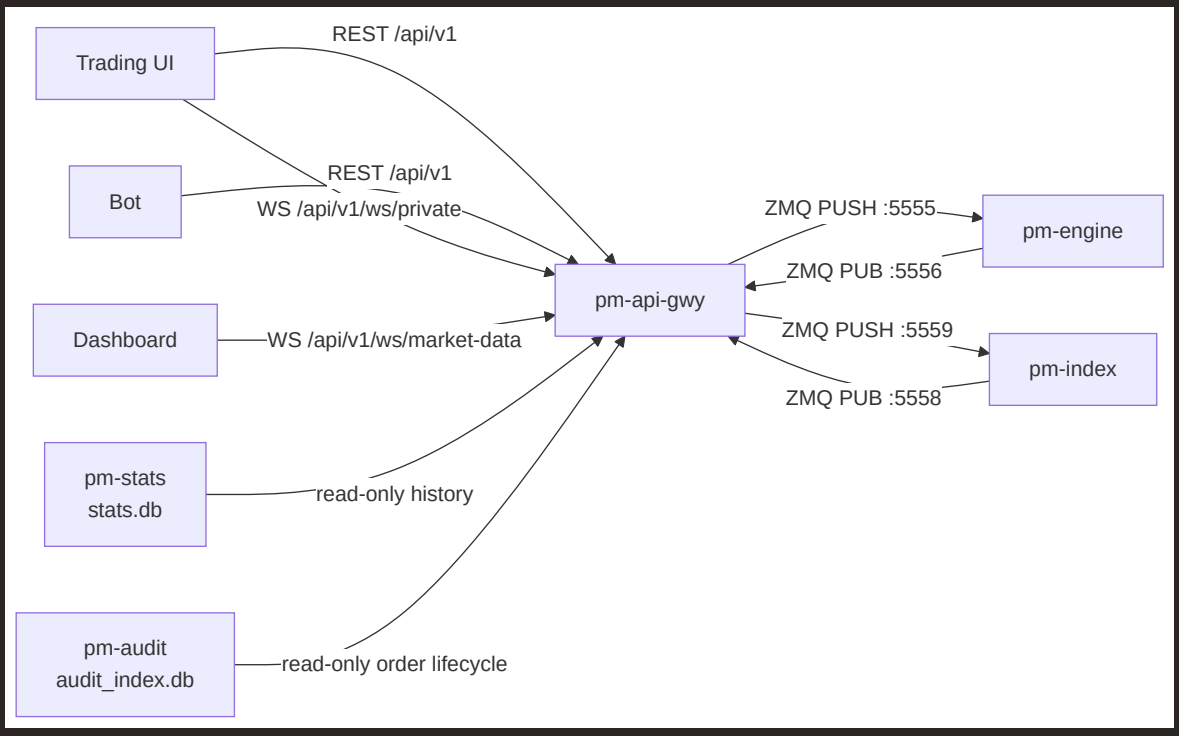
After reading this page you will understand:

- What `pm-api-gwy` does in the EduMatcher process model
- How to configure API keys in the central `engine_config.yaml`
- How to call the REST API and inspect Swagger documentation
- How private and public WebSocket streams work
- Where to find reusable Python and C REST examples

26.1 What this process is

`pm-api-gwy` exposes EduMatcher order entry, order management, reference data, history, and market data over REST/JSON and WebSocket. It is intended for third-party software: browser UIs, dashboards, simple bots, and teaching examples.

It is not a second matching engine. The process translates HTTP and WebSocket requests into the same engine ZMQ/JSON messages used by the interactive `pm-alf-console` process.



26.2 Configuration

API gateway configuration lives in the central `engine_config.yaml`, matching the existing CALF and RALF gateway pattern.

Use the top-level key `api_gateways` (underscore). The dashed form `api-gateways` is not valid.

```
api_gateways:
  desk:
    enabled: true
    host: 0.0.0.0
    port: 8080
    swagger_enabled: true
    log_level: info
    stats_db: data/stats.db
    # pm-audit's index, read-only, for GET /admin/orders/{order_id}/audit.
    # Optional: without it that one endpoint returns 503 and nothing else
    # is affected.
    audit_db: data/audit_index.db
    # pm-clearing's ledger, read-only, for the session/MTD/YTD realized P&L
    # on GET /positions. Optional: without it those three fields report null
    # and the rest of the position row is unaffected.
    clearing_db: data/clearing.db
    # Seconds a terminal order stays in the in-memory cache. 0 disables
    # eviction (unbounded growth).
    order_retention_sec: 3600
    # Seconds the market-data stream cache retains the per-symbol trades tail
    # for WS snapshot/resume. Latest book/depth/auction snapshots are kept
    # regardless of age; 0 disables the trade buffer but still serves snapshots.
    market_data_cache_sec: 60

    credentials:
      - api_key: key-trader-demo
        gateway_id: TRADER01
        description: Demo trading client
      - api_key: key-dashboard-demo
        gateway_id: null
        description: Read-only dashboard client

    rate_limit:
      writes_per_second: 10
      burst: 20

    timeouts:
      engine_auth_sec: 3.0
      engine_reply_sec: 3.0
      wait_ack_sec: 3.0
```

Field	Meaning
`api_gateways.<NAME>`	Named API gateway process configuration selected with `--instance NAME` when needed
`host` / `port`	HTTP server bind address and port
`swagger_enabled`	Enables `/docs` and `/openapi.json` when true
`credentials_file`	Optional file holding the `credentials` list, for configurations with many keys; see The Configuration File
`credentials[].api_key`	Bearer token clients use for REST and WebSocket auth
`credentials[].gateway_id`	Engine gateway identity; `null` means read-only market-data access; non-null values must be unique across `api_gateways` entries

Field	Meaning
`stats_db`	`pm-stats`' SQLite file, read-only, for `/history/*`
`audit_db`	`pm-audit`'s index, read-only, for `/admin/orders/{order_id}/audit`. Optional
`clearing_db`	`pm-clearing`'s ledger, read-only, for the session/MTD/YTD realized P&L on `/positions`. Optional: without it those three fields report `null`
`order_retention_sec`	Seconds a terminal order stays cached (default `3600`; `0` disables eviction)
`market_data_cache_sec`	Seconds the market-data cache retains the per-symbol `trades` tail for WS snapshot/resume (default `60`; latest book/depth/auction snapshots are kept regardless of age; `0` disables the trade buffer)
`rate_limit`	Per-key write limiting for POST/PATCH/DELETE endpoints
`timeouts`	Engine auth, request/reply, and synchronous ACK wait timeouts

`host` defaults to `0.0.0.0`, matching the external TCP gateways: the API gateway is intended to be reachable by browser clients, API clients, and read-only dashboards on other machines. Set it to `127.0.0.1` only for a loopback-only lab or when a reverse proxy on the same host is the only caller.

The engine's `participants` allowlist remains authoritative. If a credential maps to `TRADER01` but `TRADER01` is not allowed by the engine config, the engine rejects the API gateway handshake and every request using that credential fails with `403` and error code `ENGINE_AUTH`.

Use multiple named entries when you want logical process separation, such as one gateway for a human trading desk and another for automated clients. Each non-null `gateway_id` is owned by one API gateway process so process-local session and event state remain unambiguous. Read-only `gateway_id: null` credentials can appear in more than one entry.

26.3 Start the process

Installed mode:

```
pm-engine --verbose
pm-stats
pm-api-gwy --instance desk
```

Developer mode:

```
poetry run pm-engine --verbose
poetry run pm-stats
poetry run pm-api-gwy --instance desk
```

Useful options:

Option	Default	Description
`--host ADDR`	config value	Override HTTP bind address
`--port PORT`	config value	Override HTTP listen port
`--instance NAME`	auto-selected only when one entry exists	Select a named `api_gateways` entry
`--engine-host HOST`	config value	Override engine host for ZMQ ports `5555`/`5556`
`--stats-db PATH`	config value	SQLite database for `/history/*`

Option	Default	Description
`--clearing-db PATH`	config value	`pm-clearing` database, read-only, for the session / month-to-date / year-to-date realized P&L on `GET /positions`
`--log-level LEVEL`	`WARNING`	`CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG` (uppercase; this is the CLI process's own log level, separate from the `log_level` config value that sets uvicorn's own logger)
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	Reduce output to warnings/errors
`--log-target`	auto	Where this process's own operational log records go: `server` (default, auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file PATH`	—	Operational log file path — required when `--log-target file`
`--log-failover-timeout SECONDS`	30	Grace window before falling back to a local log file once `pm-log-srv` becomes unreachable

Uvicorn writes access and application logs to stdout/stderr. Redirect them with your shell or service manager:

```
poetry run pm-api-gwy --instance desk --log-level debug \
  > api-gateway.log 2>&1
```

26.4 Swagger interface

When `swagger_enabled: true`, open:

```
http://127.0.0.1:8080/docs
```

Swagger shows all REST endpoints, request schemas, response schemas, and enum values. Use the **Authorize** button with:

```
Bearer key-trader-demo
```

26.5 Authentication principles

REST clients send an HTTP bearer token:

```
Authorization: Bearer key-trader-demo
```

WebSocket clients send the API key as their first JSON message:

```
{ "api_key": "key-trader-demo" }
```

The API key never leaves the gateway

Engine and `pm-index` replies are addressed by a correlation id that becomes a topic suffix on the ZMQ PUB bus, where every subscriber can read it. A trading key's correlation id is its `gateway_id`; a read-only key has none, so the gateway derives one by hashing the key (`RO<16 hex>`). The raw key is used only for the in-process rate-limiter bucket. A test guard (`test_api_gateway_k13_k15.py`) scans the gateway package and fails the build if `session.api_key` is read anywhere else.

26.5.1 Auth and roles

The gateway checks auth in two steps:

1. The presented API key must exist in `api_gateways.<NAME>.credentials[].api_key`.
2. For trading/admin actions, the key must map to a non-null `gateway_id` and that gateway must be accepted by the engine allowlist and role model.

Access classes used across this guide:

- `Authenticated key` (or `Authentication key`): any configured API key.
- `Trading key`: an authenticated key with a non-null `gateway_id`.
- `Admin key`: a trading key whose mapped gateway role resolves to `ADMIN`.

What each class can do:

- Authenticated key: reference data and public history/market-data endpoints.
- Trading key: everything above plus private order/position/trading endpoints.
- Admin key: everything above plus `/api/v1/admin/*`.

Read-only credentials (`gateway_id: null`) can use public and reference surfaces but cannot submit, cancel, or inspect private orders.

26.5.2 Key types and provisioning

Keys are configured centrally in `engine_config.yaml` under `api_gateways.<NAME>.credentials`.

```
api_gateways:
  desk:
    credentials:
      - api_key: key-trader-demo
        gateway_id: TRADER01
        description: Trading client
      - api_key: key-dashboard-demo
        gateway_id: null
        description: Read-only dashboard
      - api_key: key-admin-demo
        gateway_id: ADMIN01
        description: Admin operator client
```

Provisioning rules:

- `gateway_id: null` creates a read-only authenticated key.
- `gateway_id: <GW_ID>` creates a trading-capable key bound to that gateway.
- ADMIN access is not set on the API key itself; it comes from the mapped gateway role in `participants`.
- Non-null `gateway_id` values must be unique across `api_gateways` entries.

How to get a key in practice:

- Keys are issued out-of-band by the venue/operator team.
- There is no self-service endpoint to create or rotate keys in `pm-api-gwy`.
- After key changes in config, restart/redeploy the API gateway instance.

Usage examples:

Trading REST call:

```
Authorization: Bearer key-trader-demo
POST /api/v1/orders
```

Read-only REST call:

```
Authorization: Bearer key-dashboard-demo
GET /api/v1/reference
```

Admin REST call:

```
Authorization: Bearer key-admin-demo
POST /api/v1/admin/session/transition
```

Read-only WebSocket auth:

```
{ "api_key": "key-dashboard-demo" }
```

For strict endpoint-by-endpoint access rules, see REST and WebSocket API Reference.

Error code	Status	Cause
`AUTH`	`401`	Missing/malformed `Authorization` header, or an unrecognized API key. The reply carries `WWW-Authenticate: Bearer`
`ENGINE_AUTH`	`403`	The credential's `gateway_id` isn't allowed by the engine's `participants` list
`READ_ONLY`	`403`	A `gateway_id: null` credential called a trading-only endpoint
`ROLE_DENIED`	`403`	Credential's gateway lacks the `ADMIN` role on an `/admin/*` call (or `MARKET_MAKER` on `/bootstrap/mm`)
`RATE_LIMIT`	`429`	Per-key write rate limit exceeded
`VALIDATION`	`400`	Malformed request body, query parameter or history filter (date, cursor, `event_type`); `field` names it. Includes a price that is not on the instrument's tick grid, which carries `reject_code: TICK_VIOLATION` — see Tick precision
`SYMBOLS_NOT_READY`	`503`	An order-entry request arrived before the engine's symbol metadata reached this gateway. Carries `reject_code: SYMBOL_NOT_READY`; retry shortly — see Tick precision
`STATS_DB`	`503`	`pm-stats`' SQLite file doesn't exist yet
`AUDIT_INDEX_UNAVAILABLE`	`503`	No `pm-audit` index — only affects `GET /admin/orders/{order_id}/audit`
`NOT_FOUND`	`404`	No such path
`UNKNOWN_ORDER`	`404`	Order id unknown to this gateway (`GET /orders/{id}`), or no audited events for it (`GET /admin/orders/{id}`)
`METHOD_NOT_ALLOWED`	`405`	The path exists, but not for this method
`TRANSITION_REJECTED`	`409`	The engine will not perform the requested session transition
`RELOAD_REJECTED`	`409`	`POST /admin/reference/reload` was rejected because the active config cannot be reloaded and reconstructed from the audit trail
`HALT_REJECTED` / `RESUME_REJECTED`	`409`	The engine refused a circuit-breaker trigger or resume (unknown symbol, not halted, …); `message` is the engine's reason
`KILL_SWITCH_REJECTED`	`409`	The engine refused a symbol, gateway or global kill switch; `message` is the engine's reason

Error code	Status	Cause
`REBALANCE_REJECTED`	`409`	`POST /admin/indexes/{id}/rebalance` was rejected by `pm-index`; `message` is its reason
`INTERNAL_ERROR`	`500`	A gateway bug; the message carries no internals (the gateway logs the traceback)
`ENGINE_TIMEOUT`	`503`	No engine reply within the configured timeout
`INDEX_TIMEOUT`	`503`	No `pm-index` reply within the configured timeout
`INDEX_ERROR`	`502`	`pm-index` rejected the request

Every error, including an unknown path or a gateway bug, uses one envelope: the transport-level `error.code`/`error.message` pair plus the canonical `error.reject_code` (shared with ALF; `UNKNOWN` where no engine reject applies). The full table is in the API reference. Validation failures echo `client_tag` or `request_tag` when the rejected request supplied one, so clients can correlate synchronous 4xx responses the same way they correlate asynchronous order events.

26.6 REST endpoints

The per-endpoint contract (every path, argument, reply and error code) is the normative REST API Reference, and the machine-readable one is `GET /openapi.json` (Swagger at `/docs`), also committed as `spec/rest/openapi.json`. This chapter covers what an operator needs to run the gateway and help its users: what each group of endpoints depends on, what is cached and for how long, and the behaviours that surprise people.

Group	Paths	Depends on
Order entry	`/orders`, `/oco`, `/combos`, `/quotes`, `/mass-cancel`	the engine
Account and live state	`/orders` (read), `/positions`, `/quotes` (read), `/symbols`, `/session`, `/halts`, `/status`	the engine, round trip per call
Reference data	`/reference/*`	the engine (compiled config)
Market-data snapshots	`/book`, `/book/{symbol}`	the gateway's market-data cache; `/book` also one reference round trip
History	`/history/*`	`pm-stats`' `stats.db`; `/history/index-events` asks `pm-index`
Bootstrap	`/bootstrap/trader`, `/mm`, `/admin`	the engine, several queries in parallel
Admin	`/admin/*`	the engine; rebalance asks `pm-index`; the audit view reads `audit_index.db`
Health	`/healthz`	nothing; not in Swagger

26.6.1 Order entry in practice

- **Per-type fields.** Which of `price`, `stop_price`, `visible_qty` and `trail_offset` each `order_type` needs is one rule shared by REST, ALF, BALF and the engine: a missing field is `MISSING_FIELD`, one the type does not use is `UNSUPPORTED_FIELD`, never silently dropped.
- **Waiting for the verdict.** Writes answer `202` at once; `?wait=ack` holds the reply until the engine's verdict (`200`) or `timeouts.wait_ack_sec`. A late verdict is still `202`, with the id and `ack_timeout: true`: the order was sent, so users must follow it, never resubmit it. Any `4xx`/`5xx` on a write means nothing reached the engine.

- **Safe retries.** An `Idempotency-Key` header makes a retry after a dropped connection safe: a repeat is answered with the first reply (`Idempotent-Replayed: true`) instead of a second order. Keys live in the gateway process for `order_retention_sec` and are lost on restart. `client_tag` is not an idempotency key.

```
curl -s -X POST "$GW/api/v1/orders?wait=ack" \
  -H "Authorization: Bearer $KEY" -H "Content-Type: application/json" \
  -H "Idempotency-Key: 6f1c2a9e-1f4b-4c43-9a51-3b8f0e2d7c11" \
  -d '{"symbol":"AAPL","side":"BUY","order_type":"LIMIT","quantity":100,"price":187.25}'
```

See Write outcomes and safe retries.

Tick precision

`price`, `stop_price` and `trail_offset` must be exact multiples of the instrument's tick size (`tick_decimals` in `GET /reference/symbols`). A price that is not is refused with `400`, `reject_code: TICK_VIOLATION`, not rounded: rounding would rest an order at a price the client never sent. The check tolerates ordinary floating-point error (`100.00 - 3 * 0.01` is accepted as `99.97`).

503 SYMBOLS_NOT_READY right after startup

The gateway learns each instrument's tick precision from the engine when it authenticates. An order submitted before that arrives is answered `503` with `reject_code: SYMBOL_NOT_READY` rather than converted against a default precision that may be wrong for the symbol. It is transient: retry. Only order-entry routes (and an admin reopen at a price) are affected.

26.6.2 Reference data

`/reference/*` serves the engine's **compiled, static** reference data: tick sizes, resolved risk-band collars, circuit-breaker ladders, the session schedule and index definitions, so a client never parses `engine_config.yaml`. It is distinct from `GET /symbols`, `/session` and `/halts`, which report *live* state. Every call round-trips to the engine (no gateway cache), so a reload shows at once. Any key may call it, read-only included.

`config_version`

A 16-character hex prefix of a SHA-256 hash over the compiled bundle, computed when the engine loads or reloads its config. It changes if and only if the reference data changes, so a client can cache the bundle and poll `GET /reference/config-version` to know when to refetch. Treat it as opaque.

Reloading reference data

`POST /api/v1/admin/reference/reload` (ADMIN) is currently rejected:

- A compiled config artifact has no editable source file to re-read.
- A YAML fallback's active config is not recorded in `system.startup`, so accepting a reload would make the audit trail ambiguous.

Use `pm-config-deploy` to compile and deploy the changed configuration, then restart the engine. This keeps the effective config tied to the audited run.

26.6.3 Book snapshots

`GET /book/{symbol}` and `GET /book` read the same cache a WebSocket `snapshot` is answered from, so they cost the engine nothing once a book has been seen. The engine republishes a book only when it changes, so after a gateway restart a quiet symbol has nothing cached: the first read asks the engine to publish it and waits (`503 ENGINE_TIMEOUT` if the engine does not answer). `/book` checks the symbol list against the reference data on every call, so it is complete from the first call, and it is all or nothing. A quiet book's `ts_ns` is old and the snapshot is still current: it is when the book last changed.

26.6.4 History endpoints

Every `/history/*` endpoint except `/history/index-events` reads `pm-stats`' SQLite database (`--stats-db PATH`, default `data/stats.db`). Until `pm-stats` has created it, they answer `503 STATS_DB`: a normal state for a fresh venue, not a fault. `/history/orders` and `/history/fills` need a trading key and return only the caller's own records; the rest are public market data, open to read-only keys.

`/history/index-events` is the exception: `pm-index`'s structural log (index creation, corporate actions, constituent changes) lives only in `pm-index`'s own file, so the gateway asks `pm-index` live. It answers `503 INDEX_TIMEOUT` when `pm-index` is not running, whatever the state of `stats.db`.

Lists are paginated with an opaque cursor; see the spec's pagination contract.

26.7 Bootstrap endpoints

`/bootstrap/trader`, `/bootstrap/mm` and `/bootstrap/admin` collapse the 6–13 calls a client needs at login into one, running the engine queries in parallel. A sub-query that times out is reported in `incomplete` with its field `null`, and the rest of the payload is still delivered; only the required parts (`reference`, `orders`, `positions`; `reference` for admin) fail the whole call with `503 ENGINE_TIMEOUT`, because a client cannot start without them. Positions are always the engine's: a fallback to the gateway's own fill cache would show flat positions after a gateway restart.

26.8 Admin endpoints

Every `/admin/*` route needs a key whose gateway holds the **ADMIN** role.

Role source

The API credential store does not carry role. The gateway resolves and caches the ADMIN role from the engine's gateway list reply, so the first admin call performs one extra engine round-trip.

The admin surface covers the venue (`/admin/market/halt`, `/resume`, `/kill-switch`, `/volume`, `/session/transition`, `/reference/reload`), single symbols (`/admin/symbols/{symbol}/halt`, `/resume`, `/reopen`, `/cancel-orders`, `/orders`), single participants (`/admin/gateways`, `/admin/gateways/{gateway_id}/disconnect`, `/kill-switch[?symbol=]`, `/orders`, `/positions`, `/quotes/{symbol}`), live risk (`/admin/risk/state`), order audit (`/admin/orders/{order_id}/audit`) and index rebalancing. Each is specified in the REST API Reference; the points below are what operators ask about.

- **Three kill-switch scopes.** `POST /mass-cancel` is self-service (the caller's own gateway). `/admin/gateways/{gateway_id}/kill-switch` cancels one participant's exposure (optionally one symbol) without disconnecting it, whether or not it is connected, and the audit names the ADMIN. `/admin/market/kill-switch` cancels every resting order and quote. A halt is different: it stops *trading* and leaves resting orders in place.
- **Reopening.** `/admin/symbols/{symbol}/reopen` uncrosses and clears a halt in one step, at the natural equilibrium or an operator price; run it with `dry_run` first to see the indicative print.
- **Live risk state.** `/admin/risk/state` shows where each symbol's collar and circuit breaker actually stand (reference, trigger, corridor) — unlike `/reference/risk` (the static definitions) and `/halts` (only what is halted).

26.8.1 Index administration (`pm-index` bridge)

`POST /admin/indexes/{index_id}/rebalance` is the one admin route that talks to `pm-index` instead of the engine. It applies a batch of shares-outstanding updates, validated as a whole before any is applied, and republishes the level once. A refusal is `409 REBALANCE_REJECTED` with the reason. Splits, dividends, constituent additions and delistings are not exposed over REST: use `pm-index-admin-cli`. Adding a symbol to the engine mid-session is not possible at all; the engine loads its symbol universe at startup.

26.8.2 `admin.action` monitor events

Every admin-gated engine command (halts and resumes, the kill switches, a reopen) also publishes one `admin.action` event on `/api/v1/ws/admin/monitor`: one uniform record of who ran what and what it did, so a monitor need not know each command's own ack. It never reaches a trading gateway's private stream or market data. Its fields are in the message reference. An index rebalance does not emit one: it is not an engine command.

26.9 Cross-gateway admin views

Two views answer "what is happening across all participants", from different sources:

- `GET /admin/orders` is the gateway's own **read model**: every order whose events this gateway has seen since it started, filterable, no engine round trip. It is current state bounded by order cache retention, so it misses orders that rested before the gateway started. `GET /admin/gateways/{gateway_id}/orders` asks the engine and is authoritative for one participant.

- `GET /admin/orders/{order_id}/audit` is one order's complete lifecycle, read from `pm-audit`'s index.

The audit view reads pm-audit's database

It opens **`audit_index.db` read-only** (`api_gateways[].audit_db`): the only store the gateway reads but does not own. The dependency is optional: without `pm-audit` or its index, that one route answers `503 AUDIT_INDEX_UNAVAILABLE` and every other route is unaffected. See Audit Trail for building the index.

26.9.1 Order cache retention

The in-memory order cache backs `GET /orders/{order_id}`, the private `orders.snapshot` frame and `GET /admin/orders`. Terminal orders (`FILLED`, `CANCELLED`, `EXPIRED`, `REJECTED`) are evicted after **`order_retention_sec`** (default `3600`). Resting orders are never evicted, and positions are never affected. `order_retention_sec: 0` disables eviction, and the cache then grows for the lifetime of the process. The same horizon bounds how long `Idempotency-Key` replies are kept.

26.10 WebSocket endpoints

Every socket lives under `/api/v1/ws/`, so a reverse proxy can route upgrades by that one prefix. The contract (frames, close codes, sequencing, flow control, the full event catalogue, recovery) is the normative WebSocket API Reference.

Path	Purpose	First message
`/api/v1/ws/private`	Private order, quote and risk events for one gateway	`{ "api_key": "key-trader-demo" }`
`/api/v1/ws/market-data`	Public book, trades, depth, auctions, session, halts and news	`{ "api_key": "key-dashboard-demo" }`
`/api/v1/ws/admin/monitor`	ADMIN-only feed of every engine event across all gateways, except public market data and `engine.trace`	`{ "api_key": "key-admin-demo" }`
`/api/v1/ws/admin/sim`	ADMIN-only `pm-market-sim` true values and heartbeat	`{ "api_key": "key-admin-demo" }`

What operators need to know:

- **Slow clients lose events, they do not slow the venue.** Each socket has a bounded queue; when it is full, events for that socket are dropped and the socket stays open. `GET /healthz` reports `dropped_events` per stream (`private`, `market_data`, `admin`), and the gateway logs a warning on the first drop per stream and every hundredth after. A client detects its own loss from a `seq` gap and recovers: the private stream by reconnecting (a fresh `orders.snapshot`), market data by a `snapshot` or a trades `resume`.
- **Trades are replayable for `market_data_cache_sec`** (60 s by default). Older gaps fall back to `/history/trades`.
- **The admin monitor has no replay.** On reconnect its `monitor.snapshot` is current state; history is the audit trail. Its halts and gateway roster come from the engine, not from this gateway, so participants connected over ALF, BALF or another API gateway are included; if either query times out, the snapshot says so in `incomplete` rather than refusing to open.
- **Sequence numbers restart with the gateway.** A client must treat a lower `seq` after a reconnect as a restart, not a gap.

26.11 Examples

Runnable Python and C clients live under `docs/examples/REST/`: an order's whole lifecycle with safe retries, history paging, the private stream with gap recovery, market-data subscribe and resume, and an admin halt-and-reopen. They run against a real exchange in `make test-e2e`, so they stay working. See Protocol Support Library Examples.

26.12 Implementation notes and design deviations

The original API gateway design described a separate `api_gateway_config.yaml`. EduMatcher now keeps API gateway settings in the central `engine_config.yaml` under `api_gateways:` so the API gateway follows the same configuration pattern as the other gateway processes and supports multiple named API gateway process configs.

The runtime rejects duplicate non-null `gateway_id` assignments across named API gateway entries. This is deliberate: sharing one engine gateway identity between two API gateway processes would split private session/event state across process memory. Use separate ALF gateway IDs for separately managed write paths, or use `gateway_id: null` for read-only dashboard credentials.

Swagger exposure is configurable with `swagger_enabled`. Plain bearer keys in YAML are used for teaching and local labs; production deployments should put the gateway behind TLS and manage secrets with the surrounding platform.

`?wait=ack` waits for the engine event matching both the topic and the specific `order_id`. Concurrent requests sharing one `gateway_id` each resolve independently.

`engine_auth_sec`, `engine_reply_sec`, and `wait_ack_sec` are all applied from configuration.

The implementation keeps engine payloads close to the existing EduMatcher event model. Outbound WebSocket events are wrapped in a consistent envelope, but they do not attempt broad tick-to-display price rewriting beyond the payloads already published by the engine.

Cancel-replace is implemented as cancel, wait for the cancel's outcome, then submit the replacement. The replacement body uses the same shape as `POST /orders`, including `symbol`, which must be the original's. The gateway tags its cancel, so a refused cancel (typically: the original filled before the cancel reached it) is reported as `409 REPLACE_REJECTED` rather than waited out.

`Idempotency-Key` replies are held in the gateway process (one per gateway id), for `order_retention_sec` (one hour when eviction is disabled). A duplicate that arrives while the first request is still waiting shares its outcome, and the first request completes even if its client disconnects, so its reply is there for the retry.

Startup creates the engine client and listener, but does not fail the process only because `stats.db` is absent. History endpoints depend on `pm-stats` having created and populated the configured database.

26.13 Operational checklist

1. Confirm `api_gateways.<NAME>.credentials` maps to gateways allowed under `participants`
2. Confirm each non-null `gateway_id` appears in only one API gateway entry
3. Start `pm-engine`, `pm-stats`, then `pm-api-gwy --instance NAME`
4. Open `/docs` if Swagger is enabled
5. Test `GET /api/v1/healthz` (no auth required) — returns `200 {"ok": true}` when the engine listener is running, `503 {"ok": false}` otherwise
6. Test `GET /api/v1/status` with a bearer token
7. Connect `/api/v1/ws/private` before submitting orders if you want async outcomes
8. Use `/history/*` only when `pm-stats` is running and writing `stats.db`

26.14 Minimal MARKET order CLI

The script below lives at `docs/examples/REST/python/submit_market_order.py` and reuses the same `ApiGatewayClient` library used by `demo_info.py`.

Run it from the examples directory:

```
cd docs/examples/REST/python
python3 submit_market_order.py --side BUY  --symbol AAPL --qty 100
python3 submit_market_order.py --side SELL --symbol MSFT --qty 50 --wait-ack
```

Override gateway URL and key with environment variables:

```
EDUMATCHER_API_URL=http://127.0.0.1:8080 \
EDUMATCHER_API_KEY=key-trader-demo \
python3 submit_market_order.py --side BUY --symbol AAPL --qty 100
```

Option	Required	Default	Description
`--side`	yes	—	`BUY` or `SELL`
`--symbol`	yes	—	Instrument symbol
`--qty`	yes	—	Order quantity
`--wait-ack`	no	off	Block until the matching engine ACKs the order
`--url`	no	`$EDUMATCHER_API_URL` or `http://127.0.0.1:8080`	Gateway base URL
`--key`	no	`$EDUMATCHER_API_KEY` or `key-trader-demo`	Bearer API key

Example output without `--wait-ack`:

```
order_id  : ORD-3a7f1e2c
status    : PENDING
```

Example output with `--wait-ack`:

```
order_id  : ORD-3a7f1e2c
status    : ACKED
accepted  : True
engine ack:
{
  "order_id": "ORD-3a7f1e2c",
  "accepted": true,
  "reason": null
}
```

MARKET orders must not include `price` or `stop_price`. The gateway validates this and returns `400 VALIDATION` if either field is present.

26.15 Troubleshooting

26.15.1 Check whether the port is in use

Before starting `pm-api-gwy`, or when a client cannot connect, verify that something is listening on the configured port (default `8080`).

macOS:

```
# lsof — shows the process name and PID holding the port
sudo lsof -iTCP:8080 -sTCP:LISTEN

# BSD netstat (ships with macOS)
netstat -an | grep LISTEN | grep 8080
```

Linux:

```
# ss — preferred on modern Linux
ss -tlnp 'sport = :8080'

# lsof
sudo lsof -iTCP:8080 -sTCP:LISTEN

# netstat (older distributions)
netstat -tlnp | grep 8080
```

Replace `8080` with the `port` value from your `api_gateways.<name>` config block. If no output appears, the gateway is not running.

26.15.2 Test HTTP connectivity from the command line

The health endpoint requires no authentication and is the fastest connectivity check:

```
curl -s http://127.0.0.1:8080/api/v1/healthz
# Expected: {"ok": true, "enabled": true, "active_gateways": ["TRADER01"],
#            "dropped_events": {}}
#
# A non-zero "dropped_events" entry means a WebSocket client read too slowly
# and lost events. The gateway is still healthy — shedding for a slow consumer
# is intended — but that client's data has holes. See "Detecting dropped
# events" above.
```

`ok` reflects whether the gateway is `enabled` in config **and** whether its own engine-event listener thread is alive — it does **not** confirm that `pm-engine` is actually up. The listener starts at process boot regardless of whether a peer is listening on the other end of the ZMQ socket, so `/healthz` can report `{"ok": true}` even before `pm-engine` has ever been started; it only flips to `false` if the gateway itself is disabled or its listener thread has crashed.

Test an authenticated endpoint:

```
curl -s -H "Authorization: Bearer key-trader-demo" \
     http://127.0.0.1:8080/api/v1/status
```

Test WebSocket connectivity:

```
# websocat (brew install websocat / apt install websocat)
websocat ws://127.0.0.1:8080/api/v1/ws/market-data

# curl — look for HTTP 101 Switching Protocols in the response headers
curl -v --no-buffer \
     -H "Connection: Upgrade" -H "Upgrade: websocket" \
     -H "Sec-WebSocket-Version: 13" \
     -H "Sec-WebSocket-Key: dGhlc2FtcGxla2V5" \
     http://127.0.0.1:8080/api/v1/ws/market-data
```

26.15.3 Common problems

Symptom	Likely cause	Fix
`Connection refused`	Gateway not started or wrong port	Confirm `pm-api-gwy` is running; check `port` in `api_gateways` config
`503 {"ok": false}` from `/healthz`	Gateway `enabled: false` in config, or its engine-listener thread crashed	Check `enabled` in the config block and the gateway's own logs — restarting `pm-engine` alone will not fix an already-`false` result, since `/healthz` doesn't actually probe engine liveness
`401 Unauthorized`	Missing, malformed or unknown key in the `Authorization` header (`AUTH`) — or, as `403 ENGINE_AUTH`, the engine rejected the gateway's own handshake for that `gateway_id`	Use `Authorization: Bearer <key>` with a key listed in `credentials`; if you get `403 ENGINE_AUTH` instead, check that the `gateway_id` is allowed by the engine's `participants` list
`403 Forbidden`	Credential has no `gateway_id` (`READ_ONLY`); or lacks the ADMIN role on an `/admin/*` call (`ROLE_DENIED`)	Use a credential with a non-null `gateway_id` for order-entry endpoints, or one whose engine gateway has `role: ADMIN` for admin endpoints
`429 Too Many Requests` (`RATE_LIMIT`)	Per-key write rate limit (`rate_limit.writes_per_second`/`burst`) exceeded	Slow down write requests for that API key
`404` on all endpoints	Wrong base path or wrong `--instance` flag	Check `pm-api-gwy --instance NAME` matches the config block name
Swagger UI not loading	`swagger_enabled: false`	Set `swagger_enabled: true` in the config block and restart

Symptom	Likely cause	Fix
History endpoints return empty results	`pm-stats` not running or wrong `stats_db` path	Start `pm-stats`; verify the `stats_db` path in config points to the correct file
`GET /admin/orders/{order_id}/audit` returns `503 AUDIT_INDEX_UNAVAILABLE`	`pm-audit` not deployed, or its index not built	Start `pm-audit` and run `pm-audit-cli index`; check `audit_db`. Every other endpoint is unaffected
An order vanished from `GET /admin/orders`	It reached a terminal status more than `order_retention_sec` ago	Expected — that view is current state. Use `GET /admin/orders/{order_id}/audit` for its history
`GET /admin/orders` grows without bound	`order_retention_sec: 0` disables eviction	Set a positive value
WebSocket disconnects immediately	Engine not running or client rate limit hit	Start engine; check gateway logs for disconnect reason

CHAPTER 27

Drop-Copy Gateway (`pm-dc-gwy`)

Learning objectives

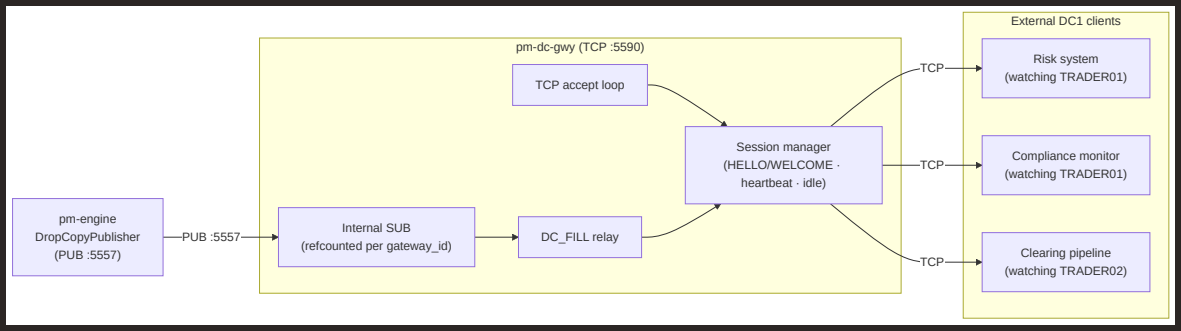
After reading this page you will understand:

- what `pm-dc-gwy` does and why it exists alongside direct port-5557 access
- how to configure and start it
- the session lifecycle: HELLO → WELCOME → unsolicited `DC_FILL` → EXIT
- the DC1 wire protocol: every message type and field
- how `pm-dc-gwy` compares to `pm-dc-spy` and the `DC` relay in `pm-alf-gwy`/`pm-alf-console`
- what it deliberately does not do (no auth, no replay-by-sequence)

27.1 What this process is

`pm-dc-gwy` is a small TCP gateway that exposes the engine's drop-copy feed (`DropCopyPublisher`, ZMQ PUB `:5557` — see Drop Copy) to plain TCP clients that cannot or should not speak ZeroMQ: a risk system, a clearing broker's ingestion pipeline, a compliance monitor, or any process on another host that only has a sockets library.

It is the TCP counterpart to `pm-dc-spy`: both connect a `zmq.SUB` socket to port 5557 and read `drop_copy.event.<gateway_id>` messages. `pm-dc-spy` prints them to a terminal for a human. `pm-dc-gwy` relays them as DC1 text lines to any number of concurrently connected TCP clients, each scoped to whichever gateway ID it asked for at connect time.



27.1.1 What this is not

`pm-dc-gwy` is deliberately minimal compared to `pm-alf-gwy` or `pm-ralf-gwy`:

Not supported	Why
Authentication / entitlement checks	Same scope decision as the raw port-5557 socket — see Drop Copy → Architecture. Any client may request any gateway ID.

Not supported	Why
Replay-by-sequence over the wire	`DropCopyPublisher.replay()` is in-process only with no external trigger — see Drop Copy → Replay. `pm-dc-gwy` does not attempt to expose it.
Order entry, quoting, or any command that mutates state	This is a read-only relay. Use `pm-alf-gwy`/`pm-alf-console` for trading.
Role/channel model (`CLEARING`/`AUDIT`/etc.)	That is RALF's `DROP_COPY` channel, a different mechanism — see RALF DROP_COPY channel vs. this feed.

A client simply says which gateway ID it wants in `HELLO` and receives that gateway's live fills for as long as it stays connected. More than one client may request the same gateway ID simultaneously — unlike `pm-alf-gwy` sessions, a `pm-dc-gwy` connection does not "occupy" a trading identity.

27.2 Prerequisites

- `pm-engine` running with its drop-copy publisher bound on `:5557` (the default; see Drop Copy → Startup and shutdown).
- Optional: add a `dc_gateway:` section to `engine_config.yaml` to customise port and limits.

27.3 Configuration

`pm-dc-gwy` reads a `dc_gateway:` section from the **compiled** engine configuration — like every other process, it never reads `engine_config.yaml` directly (see Config file location below). Author the section with `pm-config-gen --dc-gateway` (all fields are optional and default to the values shown below) or hand-edit the YAML, then install it with `pm-config-deploy` before `pm-dc-gwy` will see it:

```
dc_gateway:
  name: "dc-gwy01"
  bind_address: "0.0.0.0"
  port: 5590
  heartbeat_interval_sec: 2
  idle_timeout_sec: 20
  max_connections: 64
  max_client_queue: 10000
```

Field	Default	Description
`name`	`dc-gwy01`	Process name echoed in `WELCOME`
`bind_address`	`0.0.0.0`	Network interface to listen on (`127.0.0.1` for local-only)
`port`	`5590`	TCP listen port
`heartbeat_interval_sec`	`2`	Seconds between `HB` lines, sent whether or not fills were sent in between
`idle_timeout_sec`	`20`	Disconnect after this many seconds with no traffic in either direction. Heartbeats count, so a client that keeps reading is never idle
`max_connections`	`64`	Simultaneous TCP connections; a connection beyond this is accepted and closed at once, without a line
`max_client_queue`	`10000`	Per-client outbound line buffer capacity before the client is treated as slow and dropped

The drop-copy source address (`tcp://127.0.0.1:5557` by default) is **not** configurable via YAML — it always comes from the same `DROP_COPY_PUB_ADDR` constant the rest of the system uses. Override it per-run with `--engine-dc-pub` (see below) if you need to point at a non-default engine address.

TLS

`pm-dc-gwy` does not terminate TLS. For remote deployments, put it behind a reverse proxy (nginx, stunnel, or similar).

27.4 Start the gateway

Installed mode:

```
pm-engine --verbose
pm-dc-gwy
```

Developer mode:

```
poetry run pm-engine --verbose
poetry run pm-dc-gwy
```

CLI override options:

Option	Default	Description
`--bind ADDR`	from config / `0.0.0.0`	Override TCP bind address
`--port PORT`	from config / `5590`	Override TCP listen port
`--engine-dc-pub ADDR`	from config / `tcp://127.0.0.1:5557`	Override the engine drop-copy PUB address this gateway subscribes to
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	Reduce output to warnings/errors
`--log-target`	`server`	Where this process's own operational log records go: `server` (auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file PATH`	—	Operational log file path — required when `--log-target file`
`--log-failover-timeout SECONDS`	30	Grace window before falling back to a local log file once `pm-log-srv` becomes unreachable

Config file location

The engine configuration is read from `<EDUMATCHER_DATA_DIR>/ref_data/engine_config.json`. There is no `--config` flag: every process reads that one file, so none of them can be started against a configuration the others have not seen. Install one with `pm-config-deploy`.

27.5 Quick connect test

Use `nc` or `telnet` to validate the session lifecycle before writing any code:

```
nc 127.0.0.1 5590
```

Type the following line, pressing Enter:

```
HELLO|CLIENT=test|PROTO=DC1|ID=TRADER01
```

Expected response:

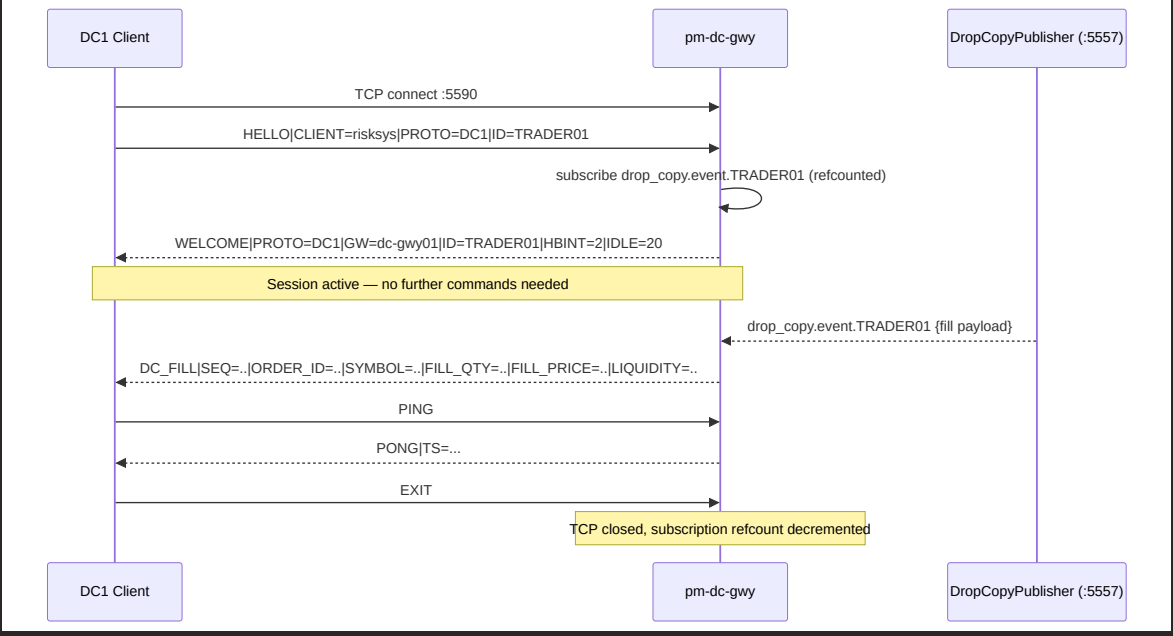
```
WELCOME|PROTO=DC1|GW=dc-gwy01|ID=TRADER01|HBINT=2|IDLE=20
```

Any fill published for `TRADER01` on port 5557 now arrives as an unsolicited `DC_FILL` line. Type `EXIT` to close cleanly.

TCP is a byte stream

Never assume one `recv()` equals one line. Always buffer and split on `\n`.

27.6 Session lifecycle



27.6.1 Step 1 — Send `HELLO`

The **first line** must be a `HELLO`:

```
HELLO|CLIENT=risksys|PROTO=DC1|ID=TRADER01
```

Field	Required	Notes
`CLIENT`	Yes	Free-text label for logging
`PROTO`	Yes	Must be exactly `DC1`
`ID`	Yes	Gateway ID to receive fills for. Not validated against any allowlist — see the auth warning below.

On any other first line, or a `HELLO` missing `CLIENT`/`PROTO`/`ID`, the gateway sends `ERR|CODE=AUTH_REQUIRED|...` and closes the connection.

A connection that sends no `HELLO` within 5 seconds receives `EXIT|REASON=auth_timeout` and is closed — the same deadline in all three line gateways (`pm-md-gwy`, `pm-ralf-gwy`, `pm-dc-gwy`). Lines a client sends after the one that ended its session (a `HELLO` pipelined behind a refused first line, for example) are ignored.

No authentication

Unlike `pm-alf-gwy`, the `ID` in `HELLO` is not checked against `participants` or any other allowlist. Any client may request fills for any gateway ID — this mirrors the lack of authentication on the raw port-5557 socket itself. See Drop Copy → Architecture.

27.6.2 Step 2 — Receive `WELCOME`

```
WELCOME|PROTO=DC1|GW=dc-gwy01|ID=TRADER01|HBINT=2|IDLE=20
```

The gateway subscribes this session to `drop_copy.event.TRADER01` on its internal SUB socket (refcounted — sharing the ZMQ subscription across every connected session watching the same gateway ID) before sending `WELCOME`.

27.6.3 Step 3 — Receive fills

No further command is needed. Every fill published on `drop_copy.event.<ID>` for the requested `ID` arrives as an unsolicited `DC_FILL` line for as long as the session stays connected.

27.6.4 Step 4 — Disconnect

Send `EXIT` for a graceful close, or simply close the TCP connection. The gateway decrements the internal subscription refcount and, if no other session still wants that gateway ID, unsubscribes from the topic.

27.7 Protocol reference

All messages follow the same line format as ALF/RALF/CALF: `MSGTYPE|FIELD=VALUE|FIELD=VALUE\n`. DC1 is intentionally the simplest of EduMatcher's TCP protocols — there is exactly one thing a client can ask for (fills for a given gateway ID), so there is no subscription grammar, no role model, and no replay-by-sequence.

27.7.1 Client → Gateway

Message	Fields	Notes
`HELLO`	`CLIENT`, `PROTO`, `ID`	Must be the first line. See Step 1.
`PING`	—	Gateway replies `PONG`
`EXIT`	—	Graceful disconnect

27.7.2 Gateway → Client

Message	Fields	Trigger
`WELCOME`	`PROTO`, `GW`, `ID`, `HBINT`, `IDLE`	Reply to a valid `HELLO`
`DC_FILL`	`SEQ`, `ORDER_ID`, `SYMBOL`, `FILL_QTY`, `FILL_PRICE`, `LIQUIDITY`	Unsolicited, one per fill for the session's gateway ID
`PONG`	`TS`	Reply to `PING`
`HB`	`TS`	Heartbeat every `HBINT` seconds
`ERR`	`CODE`, `DETAIL`	See Error codes

`DC_FILL` fields map directly from the drop-copy payload — see Drop Copy → order.fill event for the full source payload and field semantics. `SEQ` and `LIQUIDITY` (`MAKER`/`TAKER`) come from the drop-copy envelope; `DC_FILL` does not carry order state fields like `REMAINING`/`STATUS`.

```
> HELLO|CLIENT=risksys|PROTO=DC1|ID=TRADER01
WELCOME|PROTO=DC1|GW=dc-gwy01|ID=TRADER01|HBINT=2|IDLE=20
DC_FILL|SEQ=42|ORDER_ID=ORD-001|SYMBOL=AAPL|FILL_QTY=100|FILL_PRICE=150.05|LIQUIDITY=TAKER
```

27.8 Error codes

Every error arrives as `ERR|CODE=<CODE>|DETAIL=<message>`.

Code	When it occurs	Connection kept?
`AUTH_REQUIRED`	First line was not `HELLO`, or `HELLO` was missing `CLIENT`/`PROTO`/`ID`	No — closed immediately

Code	When it occurs	Connection kept?
BAD_MESSAGE	Empty line, unparseable line, or unsupported message type; or a line over 4096 bytes including the newline (DETAIL=line exceeds 4096 bytes)	Yes — except for an oversized line, which closes it
SLOW_CLIENT	Outbound queue exceeded max_client_queue	No

SLOW_CLIENT and the idle timeout end the connection even for a client that has stopped reading: the gateway tries to deliver the final line, then closes the connection after at most 5 seconds. On idle timeout the final line is EXIT|REASON=idle_timeout.

27.9 Minimal zero-dependency client (Python)

```
import socket

def dc_connect(host: str, port: int, gateway_id: str, client_name: str = "risksys"):
    sock = socket.create_connection((host, port), timeout=5)
    buf = bytearray()

    def send(line: str) -> None:
        sock.sendall((line + "\n").encode("utf-8"))

    def recv_line() -> str:
        while True:
            nl = buf.find(b"\n")
            if nl >= 0:
                line = bytes(buf[:nl]).decode("utf-8", errors="replace")
                del buf[:nl + 1]
                return line
            chunk = sock.recv(4096)
            if not chunk:
                raise RuntimeError("gateway closed connection")
            buf.extend(chunk)

    send(f"HELLO|CLIENT={client_name}|PROTO=DC1|ID={gateway_id}")
    line = recv_line()
    if not line.startswith("WELCOME"):
        raise RuntimeError(f"handshake failed: {line}")

    return sock, send, recv_line


sock, send, recv_line = dc_connect("127.0.0.1", 5590, "TRADER01")
while True:
    line = recv_line()
    if line.startswith("DC_FILL"):
        print(line)
```

TCP is a byte stream

Never assume one recv() equals one line. Always buffer and split on \n.
For full working Python and C subscribers (per-symbol volume/VWAP tally, SEQ gap and restart check, clean Ctrl-C shutdown) see docs/examples/dc/ and Protocol Support Library Examples.

27.10 When to use pm-dc-gwy vs. the alternatives

Scenario	Best choice
Ad-hoc inspection from a terminal, human-readable or JSON	`pm-dc-spy`
External process, any language, no ZeroMQ dependency allowed	`pm-dc-gwy`
Process already has ZeroMQ / pyzmq available	Connect directly to port 5557 — see Drop Copy
You already have an ALF trading session open and only want your own fills	`DC|STATE=ON` on that session — see ALF TCP Gateway → DC
Role-gated (`CLEARING`/`AUDIT`) trade feed with replay-by-sequence	`pm-ralf-gwy` — see Post-trade

27.11 Troubleshooting

27.11.1 Check whether the port is in use

macOS:

```
sudo lsof -iTCP:5590 -sTCP:LISTEN
netstat -an | grep LISTEN | grep 5590
```

Linux:

```
ss -tlnp 'sport = :5590'
sudo lsof -iTCP:5590 -sTCP:LISTEN
netstat -tlnp | grep 5590
```

27.11.2 Test the TCP connection from the command line

```
printf 'HELLO|CLIENT=test|PROTO=DC1|ID=TRADER01\nEXIT\n' | nc 127.0.0.1 5590
```

Expected output: `WELCOME|...` followed by a clean connection close.

27.11.3 Common problems

Symptom	Likely cause	Fix
`Connection refused`	Gateway not started or wrong port	Confirm `pm-dc-gwy` is running; check `dc_gateway.port` in config
`ERR|CODE=AUTH_REQUIRED` immediately	First line was not `HELLO`, or a required field was missing	Ensure the first line is `HELLO|CLIENT=..|PROTO=DC1|ID=..`
`WELCOME` arrives but no `DC_FILL` ever does	No fills have occurred for that gateway ID yet, or `pm-engine`'s drop-copy publisher failed to bind	Check `pm-engine` logs for drop-copy bind warnings; confirm trades are occurring for that gateway ID
`EXIT|REASON=idle_timeout`	Writes to the client stopped succeeding for `idle_timeout_sec` — it is not reading its socket	Read continuously; heartbeats alone keep a reading client connected
`EXIT|REASON=auth_timeout`	No `HELLO` within 5 seconds of connecting	Send `HELLO` as the first line, straight after connecting
Gateway not reachable from another host	`bind_address: 127.0.0.1`	Change `bind_address` to `0.0.0.0` (or the specific interface IP)

27.12 See also

In this book:

- ALF TCP Gateway → DC — relay your own fills through an existing ALF session instead

Reference Manual:

- Processes — process topology and ZMQ message tables

Protocols and Clients:

- Drop Copy — the engine's drop-copy feed (`:5557`) this gateway relays
- Drop-Copy Spy (pm-dc-spy) — read-only CLI for the same feed
- External Protocols Overview — protocol comparison and selection guide

CHAPTER 28

Persistence — Data Across Trading Sessions

Learning objectives

After reading this page you will understand:

- Every data file EduMatcher writes, which process creates it, when, why, and how to query it
- Which data files survive an engine restart and what each one contains
- How GTC orders preserve their price-time priority across sessions
- The exact shutdown and startup sequence that keeps the order book consistent
- How book statistics allow stop orders to trigger correctly on the first trade of a new day
- Exactly what changes — and what does not — when the business day rolls over
- How to safely inspect, edit, or delete persistence files between sessions

EduMatcher models a real exchange behaviour: **Good-Till-Cancelled (GTC)** orders survive the end of a trading session and are automatically restored when the system restarts for the next day. A resting **Day (DAY)** order also survives an engine restart, as long as the restart happens on the same business day it was submitted — a process restart is not itself a day boundary; a DAY order is only discarded, at the next startup, once its business day has actually passed. See Impact of a Business-Day Change below. Several other data files are also maintained across sessions to preserve market state and historical records.

Where is the data directory?

All persistence files live under the **data directory**, which varies by installation mode:

Mode	Default path
Developer (Poetry)	`<repo>/src/data/`
Installed (pipx)	`~/.local/share/edumatcher`
Custom	`$EDUMATCHER_DATA_DIR`

See Reference Manual → Environment variables for override details.

28.1 Data files at a glance

This is the single reference table for **every data file EduMatcher writes** — what it is, which process creates it, when, why, and how to read it. All paths are relative to the data directory shown above. Every row on this page appears in exactly one of the two tables below; there is no persistence file used by any process that is not listed here.

All data files are stored in the canonical data directory used by all processes

The full resolution order for the data directory is

Priority	Condition	Resolved path
1 — Explicit	`EDUMATCHER_DATA_DIR` env var is set	Value of `$EDUMATCHER_DATA_DIR`

Priority	Condition	Resolved path
2 — Installed	Running from a `pipx`/`pip` install	`~/.local/share/edumatcher/`
3 — Developer	Running from a source checkout	`<repo>/src/data/`

For a **clean product install** (`pipx install edumatcher`) the data directory is `~/.local/share/edumatcher/` — no `src/` prefix is involved. Run `pm-setup` once after installation to create this directory and copy a sample config file A project-root `data/` directory also exists in the repository (used for sample CSVs) but is never written to by any runtime process.

Engine state files — written by `pm-engine` and reloaded at the next startup so the market resumes where it left off.

File	Written by	When	Purpose	Read / extract with
`gtc_orders.json`	`pm-engine`	Periodic checkpoint and clean shutdown (Ctrl-C / SIGTERM)	Resting GTC orders, and resting same-day DAY orders (including MM quote legs), restored with their price-time priority at next startup — a stale DAY order (from an earlier business day) is discarded at restore instead	JSON; loaded by the engine at startup
`gtc_combos.json`	`pm-engine`	Periodic checkpoint and clean shutdown	Resting GTC combo parents and their child-leg links	JSON; loaded by the engine at startup

File	Written by	When	Purpose	Read / extract with
`book_stats.json`	`em-engine`	Periodic checkpoint and clean shutdown	Last buy/sell price and previous-close per symbol; seeds the collar and circuit-breaker reference at the next open, and the `SYMBOLS` command's `prev_close` field. Also each circuit breaker's reference window (`cb_window`: the trades its reference averages): a restart on the same business day takes it back, so the breaker goes on with the reference the market had; a later day seeds the breaker from the last price	JSON; loaded by the engine at startup
`engine_run_seq.json`	`em-engine`	Once, at the very start of every engine run — **not** check-pointed periodically	A durable, monotonically-increasing run counter, bumped and saved before anything else happens on startup (even before GTC restore). Prefixed into every trade ID (`{run_seq:06d}-{trade_seq:09d}`) so trade IDs never collide across restarts even though the in-memory per-run trade counter resets to zero each time	JSON; loaded and incremented by the engine at startup

Reading the When column for engine state files

Periodic checkpoint = the engine re-saves the current file every `_PERSIST_INTERVAL_SEC` while running, so an abrupt exit — a crash, `SIGKILL`, container eviction — loses at most one checkpoint interval of state rather than the whole session. **Clean shutdown** = the engine caught `Ctrl-C`/`SIGINT` or `SIGTERM` (the latter is how container orchestrators request a graceful stop) and finished its shutdown sequence, which also performs one final save of these same files before exit. `engine_run_seq.json`

is the one exception: it is written exactly once, at startup, and is never touched again for the life of that process — see The `engine_run_seq.json` File below.

Accumulating data stores — written by the optional subscriber processes while a session runs. They grow across sessions (never auto-truncated) and each has a dedicated query tool.

File	Written by	When	Purpose	Read / extract with
`stats.db` (SQLite)	`pm-stats`	Per trade · a price snapshot every 15 min · at EOD · every `index.update` tick	OHLCV daily stats, intraday price snapshots, per-trade log, index level snapshots and daily OHLC	**`pm-stats-cli`** or SQL — see Statistics & Reporting · schema
`clearing.db` (SQLite)	`pm-clearing`	Per trade · on gateway connect/disconnect · at EOD	Positions, VWAP cost, realized/unrealized P&L, daily summaries, trade events, sessions	**`pm-clearing-cli`** or SQL — see Positions and P&L · schema
`eod.db` (SQLite)	`pm-eod`	Once per trading date, after the close	The settlement ledger: each trader's amount per trading date and the log of `pm-eod` runs	**`pm-eod balances`** / **`status`** / **`show`** — see Closing the Day · schema
`settlement/<date>/`	`pm-eod`	Once per trading date	Trader statements, `instructions.csv`, leaderboard and the exchange summary	Plain files — see Closing the Day
`backup/<date>/pm-eod`		Once per trading date	Verified, zstd-compressed copies of `clearing.db`, `stats.db` and `eod.db`	`zstd -d`; see Restoring a backup
`audit.log`, `audit/`	`pm-audit`	Continuously (buffered flush); `audit.log` holds today and is closed as `audit/audit-DATE.N.log` at midnight or 10 MB; each past day is compressed to `audit/audit-DATE.log.zst`	Full chronological trail of every message on the bus	**`pm-audit-cli`** — see Audit Trail

`audit_index.db` (SQLite)	`pm-audit-cli`	On demand, when you run an indexed query	Fast lookup index built over the audit trail; rebuildable	**`pm-audit-cli`** — see Audit Trail · schema
`indexes/<ID>_history.jsonl`	`pm-index` (triggered by `pm-index-admin-cli` for `CORP_ACTION`/`ADD_CONSTITUENT`/`DELIST`, or the API gateway's rebalance endpoint for `REBALANCE`)	On structural events only (`INIT`, `CORP_ACTION`, `ADD_CONSTITUENT`, `DELIST`, `REBALANCE`)	Structural/corporate-action audit trail — **not** level or EOD history (that lives in `stats.db`, written by `pm-stats`)	**`pm-index-cli`** (read-only) — see Market Index
`indexes/<ID>_state.json`	`pm-index`	On startup, every trade update, EOD finalization, and every corporate action/constituent change (EOD does **not** also append to `_history.jsonl`)	Divisor, constituent list, per-symbol last prices, day OHLC and last level, so the index resumes correctly after a restart	JSON; loaded by `pm-index` at startup — rejects a mismatched constituent list (use `--reset`); see Market Index — State file
`audit_replay.db` (SQLite)	`pm-audit-replay`	On demand, when an episode index is built or refreshed	Derived episodes, causal links, anomalies and actors reconstructed from the audit trail; rebuildable	**`pm-audit-replay`** — see Audit Replay · schema
`log.db` (SQLite)	`pm-log-srv`	Continuously, one row per log event, connection and counter sample; old events pruned per `log_server.retention_days` (default 30)	Log events from every process, their connection sessions and server counters	**`pm-log-cli`** or SQL — see Log Server · schema
`logs/<name>.log` *`*.log.zst`	every `pm-*` process; compressed by `pm-eod`	Only while a process cannot reach the log server	Fallback log files; `pm-eod` compresses idle ones and removes old archives	Plain text, `zstd -d` — see Log Server and Closing the Day

`metrics.db` (SQLite)	`pm-monitor`	Continuously: raw samples and 1-minute rollups, daily totals, health events	History behind the operator dashboard	**`pm-monitor-cli`** or SQL — see Health Monitoring · schema
`log-ui-acks.db` (SQLite)	the log-gui bridge	When an operator acknowledges an issue	Acknowledgements and acknowledgement history of the log console	The log console — see Log Console · schema
`market_sim_state.json`	`pm-market-sim`	Periodically while running	Value-model state, so a restart continues where the simulation left off	JSON; loaded by `pm-market-sim` at startup — see Market Model
`market_sim_news.db` (SQLite)	`pm-market-sim`	Per published news event and value jump; never pruned	History of the simulated market's news and jumps	SQL — see Market Model · schema
`companies.db` (SQLite)	`pm-compdb-cli`	When you run `pm-compdb-cli init` / `build` / `add` / `update`	Made-up company profiles of the simulated tickers	**`pm-compdb-cli`**, the REST API — see REST and WebSocket · schema
`eod/<date>/`	`pm-engine`	At each close	Copies of `gtc_orders.json`, `gtc_combos.json` and `book_stats.json` — **not** `settlement/`	Plain JSON — see At the Close
`ai_swarm/<date>/<name>.jsonl`	the AI trader	At each session close	One summary file per trader and trading day	JSON Lines — see Running It for Weeks
`ref_data/`	`pm-setup`, `pm-config-gen`, you	When configuration is deployed or edited	`engine_config.yaml` / `.json`, the simulation file and `roster.yaml` — configuration, never cleared by `pm-opctl clear`	See The Configuration Workflow and The roster file
`emo/`	`pm-opctl`	While it starts and stops processes	Per-process stdout/stderr (`<name>.log`) and the active profile name	Plain text

`.container-config`	container entrypoint	At container start	Name of the example the data directory was set up for	Plain text — see Container Deployment Walkthrough

Reading the When column for accumulating stores

Per trade = on every `trade.executed` event. **EOD** (end of day) = when the engine broadcasts `system.eod` on a clean shutdown. **Clean shutdown** = the engine caught `Ctrl-C`/`SIGINT` or `SIGTERM` and finished its shutdown sequence — a hard kill (`SIGKILL`) skips these writes.

Not persistence

`clearing_report.csv` was v1's CSV mechanism, before clearing moved to SQLite-first persistence; the path has since been removed — use `pm-clearing-cli ... --format csv` for an ad-hoc export instead. Lock files (`.pm-eod.lock`, `audit/.archive.lock`, `stats.db.lock`, `metrics.db.lock`) only stop two runs overlapping and hold no data. `-wal` and `-shm` files next to a SQLite database are its write-ahead log and shared memory; they belong to the database and go with it.
The engine state files are described in detail in Engine State Files In Detail below; the accumulating stores each have their own chapter, linked in the table and described in Other Persistent Files.

What is deliberately not persisted: quote inactivation history

The engine keeps a small, bounded, **in-memory-only** ring buffer per gateway of recently-inactivated MM quotes (filled or cancelled), used to answer the ALF `QLEGS|SHOW=RECENT` / `SHOW=ALL` subcommands and the equivalent `system.quote_legs_request` wire message — see ALF Console → QLEGS and Messages → `system.quote_legs_request`. Unlike `gtc_orders.json`/`gtc_combos.json`, this history is **not** written to disk and does not survive an engine restart — a fresh engine process starts with empty history for every gateway. This is intentional: the persistence files on this page exist to restore **actionable, resting exposure** (orders and combos still working in the book) so the market resumes correctly after a restart. Quote inactivation history is neither resting nor actionable — it is a short operator convenience for "what just happened to my quote," and it is safe, by design, for it to reset to empty on every engine restart. The buffer's bound (30 entries per gateway by default) also does not need to be pre-sized against restart timing; it exists only to cap memory use during a single continuous run.

28.2 How It Works

28.2.1 At Shutdown (Ctrl-C / SIGTERM on the engine)

A process exit — clean or otherwise — is **not** a day boundary. `TIF = DAY` orders are no longer expired here; they are persisted the same as `TIF = GTC` orders, and the same-day-vs-stale decision is made once, at the next startup (see below). True end-of-day `DAY` expiry is driven separately by the session scheduler's transition to `CLOSED` (see Impact of a Business-Day Change and Auctions & Scheduling), not by the engine process exiting.

1. The engine collects all **resting** orders (status `NEW` or `PARTIAL`) from every order book with `TIF = GTC` or `TIF = DAY`.
2. Those orders are serialized to `<DATA_DIR>/gtc_orders.json`.
3. GTC combos (status `PENDING` or `PARTIALLY_MATCHED`) are serialized to `<DATA_DIR>/gtc_combos.json`. `DAY` combos are out of scope for this persistence: only their individual child orders survive per step 1-2 above, as plain resting orders — the combo parent record itself does not currently survive a restart.
4. Book statistics (`last_buy_price`, `last_sell_price`, and `prev_close` per symbol) are saved to `<DATA_DIR>/book_stats.json`.
5. A `system.eod` message is published with final book snapshots for all symbols (allows stats/viewers to record closing state).
6. ZMQ sockets are closed.

The engine also checkpoints steps 1-2 and 4 periodically while running (not only at shutdown), so an abrupt exit — a crash, `SIGKILL`, container eviction — loses at most one checkpoint interval of state rather than the whole session. `engine_run_seq.json` is not part of this shutdown sequence at all — it is only ever written once, at the following startup.

28.2.2 At Startup

0. Before anything else — before GTC restore, before config is loaded — the engine reads `<DATA_DIR>/engine_run_seq.json`, increments the run counter, and saves it back immediately. This guarantees every trade ID minted during this run is globally unique, even against trade IDs from earlier runs. See The `engine_run_seq.json` File.
1. The engine reads `<DATA_DIR>/gtc_orders.json` (if it exists).
2. Each `TIF = GTC` order is re-injected into its symbol's order book unconditionally, **with its original timestamp preserved**.
3. Each `TIF = DAY` order is re-injected **only if its own timestamp falls on the current business day** (machine-local calendar date). A `DAY` order dated before today — because the business day rolled over while the engine was up, or while it was down — is discarded instead, and the discard is logged at `INFO` level. This is the mechanism that ultimately gives `TIF = DAY` its intended meaning of "valid for the current trading day," independent of how many times the engine process itself has restarted in between. This rule applies to every resting order regardless of origin, including MM quote legs (step 3a below). 3a. Restored quote-origin orders (from steps 2-3 above) are grouped back into linked bid/ask pairs by `(gateway_id, quote_id)` and re-registered as active quotes in the engine's in-memory `QuoteIndex`, so a surviving quote is fully quote-managed again — not just resting as a plain order. A quote whose sibling leg did not survive (filled, cancelled, or discarded as a stale `TIF=DAY` order) restores its surviving leg as an ordinary resting order instead, logged as a "single-leg quote remnant." 3b. Restored OCO legs are linked back into their pairs by `oco_group_id`, so a leg that fills or ends after the restart still ends its sibling. A pair with only one surviving leg is not re-linked: that leg rests as an ordinary order.
4. The engine reads `<DATA_DIR>/gtc_combos.json` (if it exists) and rebuilds parent-child tracking maps for restored `GTC` combos.
5. If any orders were restored, initial book snapshots are published. Restoring never matches, so a book saved during pre-open or an auction call comes back crossed. With sessions enabled the engine starts `CLOSED` and the crossed book waits for its auction, as it would have without the restart. With sessions disabled the engine starts in continuous trading, which must not hold a crossed book, so each crossed book is uncrossed at once (`auction.result` reason `RECOVERY`).
6. The engine reads `<DATA_DIR>/book_stats.json` (if it exists) and restores `last_buy_price` / `last_sell_price` / `prev_close` per symbol. Persisted values take priority over config-seeded values. `prev_close` is also the book's last trade price again — the collar reference, a new trailing stop's starting point, the depth window's centre — unless a `RECOVERY` uncross in step 5 has printed a newer one.
7. Market-maker quotes from each symbol's `market_maker_quotes` config section are injected as linked bid/ask quote legs, unless `seed_once: true` (the default) and this `(gateway_id, symbol)` pair already has an active entry in `QuoteIndex` — i.e. a live quote was restored in step 3a. **No gateway connection is required** — seeds enter the book before any participant dials in. If a restored order already crosses a seed price, a trade executes immediately during this step. This can also happen **between two different market makers**: a restored quote leg from one gateway can cross a freshly-injected seed from a different gateway — e.g. one gateway's quote only partially survived the restart (see Operational edge cases below), so `seed_once` still allows a second gateway's seed to fire, and that seed happens to cross the first gateway's surviving leg. Same startup-crossing behaviour as above, just a new pair of counterparties.
8. Market-maker combos from the `market_maker_combos` config section are injected.
9. Book snapshots are published for any symbol where MM quotes were injected.
10. Price-time priority carries over: each book's orders are re-injected in their current priority (arrival sequence), so an order submitted yesterday still has seniority over a new order at the same price submitted today, and an order that had lost its place (an iceberg whose slice refilled, an amend that resets priority) does not get it back.

28.2.3 At the Close (every trading day)

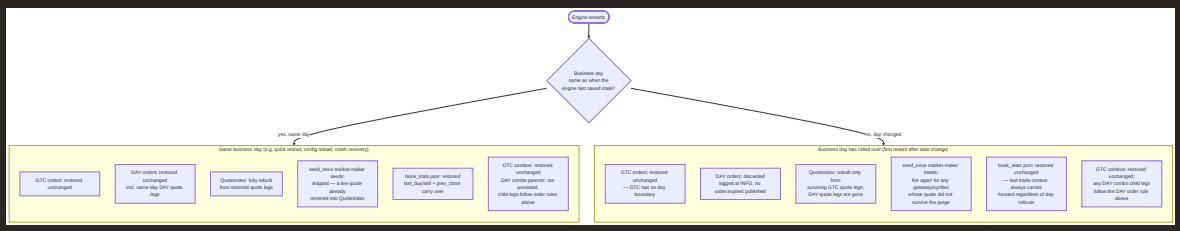
When the session moves to `CLOSED`, after the day's `DAY` orders have expired, the engine copies what it would persist — `gtc_orders.json`, `gtc_combos.json` and `book_stats.json` — into `<DATA_DIR>/eod/<YYYY-MM-DD>/` (the machine-local business date). The regular files are rewritten by every checkpoint; the archive keeps each day's closing state, so the next morning's book can be compared with it (the simulation forensics check G3 does). Nothing is ever deleted there.

eod/ is not settlement/

`<DATA_DIR>/eod/<date>/` is the **engine's** archive of its own persistence files. The end-of-day *statements* written by `pm-eod` go to a different directory, `<DATA_DIR>/settlement/<date>/`, so the two can never overwrite each other. See Closing the Day.

28.3 Impact of a Business-Day Change

The single fact that governs everything on this page: **a process restart is not a day boundary.** The engine has no clock of its own that fires at midnight — the only place "is this still the same business day?" is ever evaluated is once, at startup, by comparing each restored order's own timestamp against today's machine-local calendar date (see At Startup, steps 1-3 above). Two restarts can therefore behave very differently depending on nothing but whether a calendar date happened to change while the engine was down — the restart mechanics themselves (steps 0-10 above) are identical either way.



Reading the diagram: **only `TIF = DAY` orders — including DAY-tenored quote legs — are sensitive to the business-day check.** Everything else (`gtc_orders.json`'s GTC rows, `gtc_combos.json`, `book_stats.json`, `engine_run_seq.json`) is restored identically whichever branch is taken, because none of those depend on the calendar at all.

The practical consequences of the "rolled over" branch:

- A DAY order a trader left resting overnight is gone the next trading day — by design, this is what `TIF = DAY` means. It is not cancelled with an `order.cancelled` event; it is silently dropped during restore (a discard is logged at `INFO` level and counted in a debug counter, but no order-lifecycle message is published for it — from a gateway's point of view the order simply never comes back).
- A market maker's `TIF = DAY` quote leg is purged the same way. Because `seed_once` only skips re-seeding when a *live* quote is already in `QuoteIndex`, a purged DAY quote's `(gateway_id, symbol)` pair has no `QuoteIndex` entry after restore, so the configured `market_maker_quotes` seed for that pair fires again on this startup — the market reopens with a fresh seed rather than a stale, days-old order.
- `book_stats.json` is **not** day-sensitive: `last_buy_price` / `last_sell_price` / `prev_close` restore unchanged across a day rollover. This is intentional — `prev_close` exists specifically to carry the prior session's closing price into the new session.
- `gtc_combos.json` combo parents are GTC-only by construction (see The `gtc_combos.json` File) and are therefore never subject to the day check either; only their child orders, if any leg were ever a DAY order, would be.
- There is currently **no live, mid-session sweep** for a day rollover that happens while the engine keeps running — the check above only runs at startup. See the note below.

No day-rollover sweep while running

The engine does **not** watch for midnight and does not halt trading to purge stale DAY orders while it is running — the business-day check above only runs at startup. In a classroom setting with `sessions_enabled: false` (no `pm-scheduler`), a stale TIF=DAY order from a previous business day is only discarded the next time the engine restarts. If you deliberately want to clear all resting DAY orders between sessions in a classroom setup that does not run `pm-scheduler`, restart the engine at least once after the date has rolled over — the stale-order check at startup discards them then. See the design rationale in `docs-design/EduMatcher-Revised-Quote-Persistence.md` §13.1 (repository checkout only).
When `pm-scheduler` **is** running, true end-of-day `DAY` expiry is additionally driven live by the session's transition to `CLOSED`, independent of any engine restart — see Auctions & Scheduling.

28.4 Operational edge cases

- If `<DATA_DIR>/gtc_orders.json`, `<DATA_DIR>/gtc_combos.json`, or `<DATA_DIR>/book_stats.json` is malformed JSON, startup does **not** fail. The loader returns empty state for that file and the engine continues. `<DATA_DIR>/engine_run_seq.json` is the one exception: a corrupt or unreadable run-sequence file **is** fatal at startup, because silently restarting the counter risks reissuing a trade ID that a downstream store (e.g. `stats.db`'s `trade_log.trade_id` primary key) already treats as durable. See The `engine_run_seq.json` File.
- Restored orders for symbols that no longer exist in the current config are skipped during restore rather than aborting startup.
- Because config quote seeds run **after** persisted order/quote restore, a seeded `market_maker_quotes` entry with `seed_once:`

`false` can duplicate already-restored quote inventory on restart — `seed_once: true` (the default) avoids this by skipping the seed whenever this `(gateway_id, symbol)` pair already has an active entry in `QuoteIndex`, i.e. a live quote was actually restored. This is a live-presence check, not a trading-history check: a quote that later gets fully hit through (or expires as a stale `TIF=DAY` order) correctly has no `QuoteIndex` entry, so the next startup's seed fires again rather than leaving the book with no MM presence.

- Quote legs (`origin=QUOTE`) persist by the same `TIF=GTC`/`TIF=DAY` rule as any other resting order — they are no longer a special case excluded from `gtc_orders.json`. A restored quote is re-registered as a live, quote-managed entry in `QuoteIndex` (not just a plain resting order); see At Startup above, step 3a. See `docs-design/EduMatcher-Revised-Quote-Persistence.md` (repository checkout only) for the full design rationale, and Market-Maker page for the market-maker-specific walk-through of this behaviour.
- A `TIF = DAY` order now survives an engine restart the same as a `TIF = GTC` order does, as long as the restart happens on the same business day it rested on. This means restarting the engine mid-session to pick up a config change, or recovering from a crash, no longer cancels traders' DAY orders — only an actual business-day rollover does, checked once at the next startup. See Impact of a Business-Day Change above for the full breakdown.

28.5 Engine State Files In Detail

The four files below are written and read exclusively by `pm-engine` itself (the engine state files row group in the at-a-glance table). This section is organized file-by-file; for command syntax to submit or cancel a GTC order, see Working with GTC Orders further down the page.

28.5.1 The `gtc_orders.json` File

Format: a JSON array of serialized `Order` objects.

```
[
  {
    "id": "8f3a1b4c9d2e04f7a6b1c8d9e0f1a2b3",
    "symbol": "AAPL",
    "side": "BUY",
    "order_type": "LIMIT",
    "tif": "GTC",
    "quantity": 100,
    "remaining_qty": 100,
    "gateway_id": "GW01",
    "tick_decimals": 2,
    "ts_ns": 1714393921345678000,
    "status": "NEW",
    "price_ticks": 14800,
    "stop_price_ticks": null,
    "trail_offset_ticks": null,
    "oco_group_id": null,
    ...
  }
]
```

The real serialized keys are `price_ticks`, `stop_price_ticks`, `trail_offset_ticks` and `ts_ns` (`Order.to_dict()`) — not `price`, `stop_price`, `trail_offset` or `timestamp`. `Order.from_dict()` requires `ts_ns` to be present and raises `KeyError` otherwise, so a hand-edited file using the wrong key names fails to load (each bad entry is skipped with a CRITICAL log line, not a hard crash). The full record also carries `tick_decimals`, `oco_group_id`, `visible_qty`, `displayed_qty`, `smp_action`, `combo_parent_id`, `leg_index`, `origin`, `is_seed`, `quote_id`, `client_tag` and `arrival_seq`.

Internal representations in the JSON

Prices (`price_ticks`, `stop_price_ticks`, `trail_offset_ticks`) are stored as **integer tick values** — e.g. `14800` represents `148.00` for a symbol with `tick_decimals: 2`. Timestamps (`ts_ns`) are **nanoseconds** since the Unix epoch, not seconds. You can inspect or edit this file between trading sessions. Since this file now also holds resting TIF=DAY orders (see How It Works above), deleting it before restarting clears **both** GTC and same-day DAY orders — not just GTC ones as in earlier versions.

28.5.2 The `engine_run_seq.json` File

Format: a single JSON object holding the last-issued run number.

```
{
  "run_seq": 42
}
```

This file exists for exactly one reason: to keep trade IDs globally unique across engine restarts. Within a single run, trades are numbered `1, 2, 3, ...` in memory; that counter always restarts at `1` on the next process start. Without something to distinguish runs from each other, the second run's trade `1` would collide with the first run's trade `1` in any downstream store that treats trade IDs as a primary key (notably `stats.db`'s `trade_log` table — see the `trade_log` table further down this page).

`engine_run_seq.json` solves this by persisting a run counter that only ever goes up. At the very start of every `run()` — before GTC restore, before config is loaded, before anything else — the engine reads the current `run_seq`, increments it, writes the new value back immediately, and holds the incremented value in memory for the rest of that process's life. Every trade ID minted during that run is then formatted as `{run_seq:06d}-{trade_seq:09d}`, e.g. `000042-000000001` for the first trade of run 42 — so trade IDs from different runs can never collide, even though each run's own counter starts over at 1.

Unlike the other three engine state files, this file is **not** periodically checkpointed and is untouched by the shutdown sequence — it is written once, at startup, and not written again until the next startup. A missing file is treated as run `0` (so the very first run becomes run `1`); a **corrupt or unreadable** file is fatal — the engine refuses to start rather than risk reissuing trade IDs a downstream store already considers durable. See Operational edge cases above.

28.5.3 The `book_stats.json` File

Preserves the **last trade price context** per symbol across sessions. This serves two purposes: it allows the engine to correctly trigger stop orders on the first trade of a new day (stops compare against `last_trade_price`, which would otherwise be unknown), and it carries the prior session's closing price forward as `prev_close`, which the engine reports in its `SYMBOLS` command response so gateways can show a previous-close reference.

Format: a JSON object keyed by symbol. Unlike `gtc_orders.json` and `gtc_combos.json`, prices here are stored as **display floats** (not integer ticks) — the save path deliberately converts ticks to display prices before writing, and the load path converts back on restore, so values round-trip exactly regardless of a symbol's `tick_decimals`:

```
{
  "AAPL": {"last_buy_price": 150.25, "last_sell_price": 149.80, "prev_close": 150.10},
  "MSFT": {"last_buy_price": null, "last_sell_price": 415.50, "prev_close": 415.50}
}
```

- `last_buy_price`: display price of the most recent trade where the buyer was the aggressor
- `last_sell_price`: display price of the most recent trade where the seller was the aggressor
- `prev_close`: display price of the most recent trade overall (`last_trade_price`), carried forward as the next session's previous-close reference
- `null` means no trade of that type occurred during the session

On startup, persisted values **override** any `last_buy_price` / `last_sell_price` seeded in `engine_config.yaml`. Config seeds are only used when no persisted file exists (first run).

This file is never day-sensitive — see Impact of a Business-Day Change above.

Config seeds are the IPO price; persisted stats are the carried-over close

The `last_buy_price` / `last_sell_price` in `engine_config.yaml` are the symbol's opening (IPO) reference, used **only** on the very first startup. On every later restart the persisted `book_stats.json` value wins, so a symbol re-opens from where it last traded rather than snapping back to a now-stale config price. Both the collar and circuit-breaker references use this same resolved value — see Risk Controls - Day one (IPO) behaviour.

28.5.4 The `gtc_combos.json` File

Format: a JSON array of serialized `ComboOrder` objects (only combos with TIF=GTC and status `PENDING` or `PARTIALLY_MATCHED`):

```
[
  {
    "id": "internal-uuid",
    "combo_id": "MY-PAIR-01",
    "gateway_id": "GW01",
    "combo_type": "AON",
    "tif": "GTC",
    "timestamp": 1714393921345678000,
    "legs": [ ... ],
    "status": "PARTIALLY_MATCHED",
    "child_order_ids": ["uuid-1", "uuid-2"],
    "leg_fill_qty": {"0": 50, "1": 0},
    "leg_statuses": {"0": "PARTIAL", "1": "NEW"}
  }
]
```

On restore, the engine rebuilds the `_combos` and `_order_to_combo` tracking maps so that fill events on restored child orders correctly propagate to their parent combo. Only `TIF=GTC` combo parents are ever written here, so this file is never day-sensitive either — see Impact of a Business-Day Change above.

28.6 Working with GTC Orders

28.6.1 Submitting a GTC Order

Add `TIF=GTC` to any LIMIT, STOP, STOP_LIMIT, or ICEBERG order:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=148.00|TIF=GTC
NEW|SYM=AAPL|SIDE=BUY|TYPE=ICEBERG|QTY=1000|PRICE=149.00|VISIBLE=100|TIF=GTC
```

MARKET, FOK, and IOC orders are always DAY orders — they cannot be GTC because they do not rest.

28.6.2 Cancelling a GTC Order

Cancel it like any other order while the engine is running:

```
CANCEL|ID=<full-order-id>|RTAG=gtc-cxl-001
```

Cancelled orders are **not** included in the GTC save at shutdown — they are already marked `CANCELLED`.

Tip

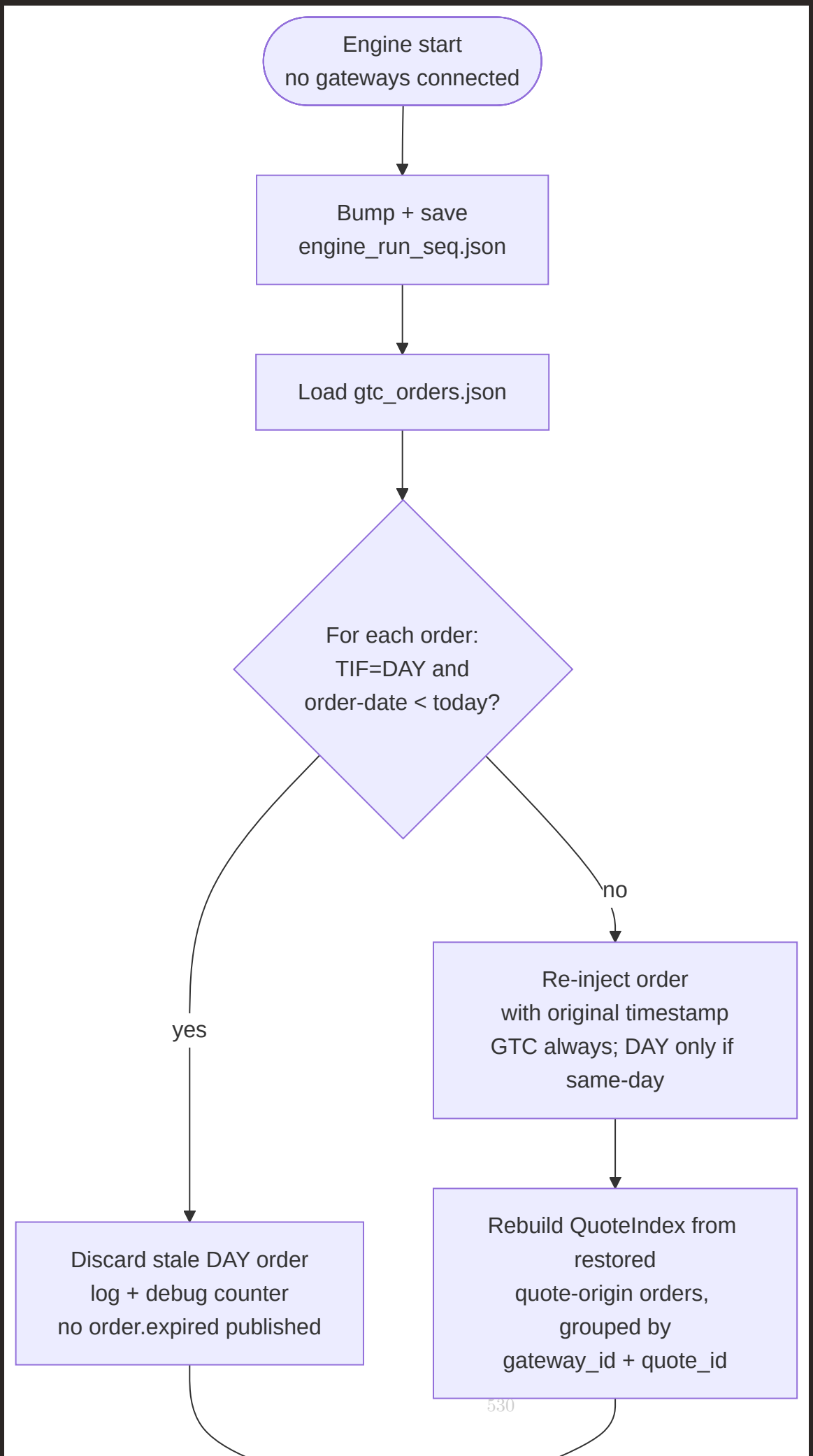
To find the full order ID, type `ORDERS` in your gateway terminal or check the audit log.

28.6.3 Order ID Stability

GTC order IDs are 32-character random hex strings (`os.urandom(16).hex()`), generated at submission time — **not** UUID4 strings; EduMatcher deliberately avoids the `uuid` module's formatting overhead while keeping the same 128 bits of entropy (see `src/edumatcher/models/ids.py`). They **do not change** across restarts. Gateways and the order monitor will see the same order ID in all events throughout the order's life.

28.7 General Startup / Shutdown Sequence

The diagram below shows the full mechanical sequence for every restart — the same steps run whichever branch of Impact of a Business-Day Change applies; only the `DAYCHK` decision differs between them.


Engine start
no gateways connected
Bump + save
engine_run_seq.json
Load gtc_orders.json
For each order:
TIF=DAY and
order-date < today?
yes
no
Discard stale DAY order
log + debug counter
no order.expired published
Re-inject order
with original timestamp
GTC always; DAY only if
same-day
Rebuild QuoteIndex from
restored
quote-origin orders,
grouped by
gateway_id + quote_id

28.8 Other Persistent Files

These files are maintained by subscriber processes (not the engine) and accumulate data continuously across sessions. They are **never truncated automatically** — to reset them, delete them manually between sessions.

28.8.1 `src/data/audit.log`

Written by: `pm-audit`
Written: continuously — one line per ZeroMQ message received
Read by: `pm-audit-cli`, manual inspection, `grep`, log-analysis tools
Reset: delete or rotate manually; the process creates a fresh file on startup

`pm-audit` subscribes to **all topics** on the engine PUB socket (`:5556`) and appends every message as a single line:

```
[TIMESTAMP] [TOPIC] {JSON_PAYLOAD}
```

Example lines:

```
[2026-04-29T14:30:00.123+00:00] [system.gateway_auth.GW01] [seq=1] {"accepted": true, "gateway_id": "GW01"}
[2026-04-29T14:30:01.456+00:00] [order.ack.GW01] [seq=2] {"id": "8f3a1b4c9d2e04f7a6b1c8d9e0f1a2b3", "symbol": "AAPL", "accepted": true
[2026-04-29T14:30:02.789+00:00] [trade.executed] [seq=3] {"id": "abc123", "symbol": "AAPL", "price": 150.05, "quantity": 200, "buy_gat
[2026-04-29T14:30:02.791+00:00] [order.fill.GW01] [seq=4] {"id": "8f3a1b4c9d2e04f7a6b1c8d9e0f1a2b3", "symbol": "AAPL", "side": "BUY",
[2026-04-29T16:05:00.000+00:00] [session.state] [seq=5] {"state": "CLOSED"}
```

Format details:

Component	Description
`TIMESTAMP`	ISO 8601 UTC with millisecond precision (`2026-04-29T14:30:01.456+00:00`)
`TOPIC`	The ZeroMQ topic string, e.g. `order.fill.GW01`, `trade.executed`, `session.state`
`[META]`	Present on almost every line: a bracketed `seq=/msg=/cause=/chain=` section built from the message's sequence and causal envelope; absent only for the rare message with no envelope at all — see Audit
`{JSON_PAYLOAD}`	The full message payload as compact JSON — no pretty-printing

The topic is **not** a JSON field inside the payload; it appears as a separate bracket-delimited token on the same line, and the `[META]` section, when present, is a further bracketed token between the topic and the payload.

Day closing: `audit.log` holds one trading day (session timezone). At the date change, or at `--max-file-mb` (10 MB), it is closed as `audit/audit-YYYY-MM-DD.N.log`; nothing is deleted. Each closed day is compressed to one `audit/audit-YYYY-MM-DD.log.zst` (zstd, 13:1), and `--keep-days N` deletes archives older than N days. See Audit Trail.

Useful grep patterns:

```
# All trades
grep '\[trade\.executed\]' src/data/audit.log

# All fills for gateway GW01
grep '\[order\.fill\.GW01\]' src/data/audit.log

# All session-state changes
grep '\[session\.state\]' src/data/audit.log

# Events in a specific time window
grep '^\[2026-04-29T14:3' src/data/audit.log
```

28.8.2 `src/data/clearing.db`

Written by: `pm-clearing`
Written: continuously — on every `trade.executed`, gateway connect/disconnect, and `system.eod` event

Read by: `pm-clearing-cli`, direct SQL queries, post-trade analysis scripts
Reset: delete manually; `pm-clearing` recreates the schema (`CREATE TABLE IF NOT EXISTS`) on startup, so restarting against an existing file is safe

A SQLite database that accumulates across sessions. It holds an append-only `trade_events` fact table plus running-state tables for positions and daily summaries, and clearing-lifecycle tables for sessions and connections. The full schema (five tables and two views) and the VWAP/realized/unrealized P&L formulas are documented in Positions and P&L.

If `pm-clearing` is not running when a trade executes, that trade is not recorded here (it is still in `stats.db` and `audit.log`).

28.8.3 `src/data/stats.db`

Written by: `pm-stats`
Written: on every `trade.executed` event, a book snapshot every 15 minutes, and at end-of-day finalization (`daily_stats` only)
Read by: `pm-ticker`, `pm-board`, direct SQL queries, `pm-stats-cli`
Reset: delete manually; `pm-stats` creates a fresh database with the schema on startup

This SQLite database holds nine tables in total — seven data tables plus an instrument reference table and a metadata table; this page covers the three most relevant to session-to-session persistence. See Statistics & Reporting — The Statistics Database Schema for the complete schema, including `feed_gaps`, `order_events`, `index_daily_stats`, and `index_level_snapshots`. The schema is created automatically by `pm-stats` using `CREATE TABLE IF NOT EXISTS` on startup; it is safe to restart `pm-stats` against an existing database.

Table: `daily_stats`

One row per `(date, symbol)` pair. Upserted on every trade and at end-of-day.

```
CREATE TABLE IF NOT EXISTS daily_stats (
    date                TEXT NOT NULL,    -- ISO date string, e.g. '2026-04-29'
    symbol              TEXT NOT NULL,    -- e.g. 'AAPL'
    open_price          INTEGER,          -- first trade price of the day, in ticks
    high_price          INTEGER,          -- highest trade price of the day, in ticks
    low_price           INTEGER,          -- lowest trade price of the day, in ticks
    close_price         INTEGER,          -- most recent trade price, in ticks (updated on every trade)
    open_bid            INTEGER,          -- best bid at session open, in ticks
    open_ask            INTEGER,          -- best ask at session open, in ticks
    close_bid           INTEGER,          -- most recent best bid, in ticks
    close_ask           INTEGER,          -- most recent best ask, in ticks
    volume              INTEGER NOT NULL DEFAULT 0,   -- total shares traded today
    trade_count         INTEGER NOT NULL DEFAULT 0,   -- number of individual executions
    turnover            INTEGER NOT NULL DEFAULT 0,    -- sum(price_ticks x quantity); exact VWAP numerator
    vwap                REAL,             -- turnover / volume, in ticks (derived, hence REAL)
    largest_trade_qty   INTEGER,          -- single largest execution quantity
    largest_trade_price INTEGER,          -- price of that largest execution, in ticks
    tick_decimals       INTEGER NOT NULL DEFAULT 2,     -- decimal scale for this row's tick columns
    PRIMARY KEY (date, symbol)
);
```

Prices are ticks, not display decimals

Every price column here is an **integer tick count**, converted with the row's own `tick_decimals` (`display price = price / 10^tick_decimals`) — the same convention `gtc_orders.json`'s `price_ticks` uses (see Internal representations in the JSON above). `vwap` is the one exception: it's `turnover / volume`, a derived ratio stored as `REAL`. See Statistics & Reporting — The conversion rule for the full rationale and every table's exceptions.
Example row:

```
SELECT * FROM daily_stats WHERE symbol = 'AAPL' ORDER BY date DESC LIMIT 1;
-- date='2026-04-29', symbol='AAPL', open_price=14950, high_price=15200,
-- low_price=14875, close_price=15125, open_bid=14945, open_ask=14955,
-- close_bid=15120, close_ask=15130, volume=42300, trade_count=184,
-- turnover=636065100, vwap=15037.0 (= 150.37 at tick_decimals=2),
-- largest_trade_qty=500, largest_trade_price=15000, tick_decimals=2
```

Table: `price_snapshots`

One row per `(timestamp, symbol)` pair, written approximately every 15 minutes from book-state events. Used by `pm-ticker` and `pm-board` for intraday charts.

```
CREATE TABLE IF NOT EXISTS price_snapshots (
    ts            TEXT NOT NULL,   -- ISO datetime string, e.g. '2026-04-29T14:30:00'
    symbol        TEXT NOT NULL,   -- e.g. 'AAPL'
    mid_price     REAL,            -- (best_bid + best_ask) / 2 in ticks; NULL if no quote
    best_bid      INTEGER,         -- top-of-book bid price, in ticks; NULL if empty
    best_ask      INTEGER,         -- top-of-book ask price, in ticks; NULL if empty
    pct_change    REAL,            -- % change from open_price; NULL if no open yet
    tick_decimals INTEGER NOT NULL DEFAULT 2,  -- decimal scale for this row's tick columns
    PRIMARY KEY (ts, symbol)
);
```

`mid_price` is REAL even though `best_bid`/`best_ask` are ticks, because a midpoint can land on a half tick that no integer can hold — the one exception on this table; see the conversion-rule link above. The mid-price fallback chain when the book has no two-sided quote: 1. `(best_bid + best_ask) / 2` if both sides present 2. `best_bid` if only bids present 3. `best_ask` if only asks present 4. `NULL` if the book is completely empty

`INSERT OR IGNORE` is used — duplicate `(ts, symbol)` entries are silently discarded, so re-sending a snapshot for the same timestamp is safe.

Table: `trade_log`

One row per individual trade execution. Written on every `trade.executed` event.

```
CREATE TABLE IF NOT EXISTS trade_log (
    ts              TEXT NOT NULL,                -- ISO datetime string
    trade_id        TEXT NOT NULL,                -- durable engine trade id
    symbol          TEXT NOT NULL,                -- e.g. 'AAPL'
    price           INTEGER NOT NULL,              -- execution price, in ticks
    quantity        INTEGER NOT NULL,              -- executed quantity
    tick_decimals   INTEGER NOT NULL DEFAULT 2,    -- decimal scale for price
    buy_gateway_id  TEXT,                          -- gateway that was the buyer
    sell_gateway_id TEXT,                          -- gateway that was the seller
    aggressor_side  TEXT,                          -- BUY, SELL, or AUCTION for an uncrossing print
    PRIMARY KEY (trade_id)
);
```

`INSERT OR IGNORE` on `trade_id` makes duplicate deliveries safe. Engine trade IDs are durable strings such as `000042-000000001`, where the prefix is the engine run sequence persisted in `engine_run_seq.json` (see The `engine_run_seq.json` File above) and the suffix is the trade counter within that run — this is what keeps `trade_id` collision-free as a primary key across restarts.

Example queries (converting ticks to display prices with each row's own `tick_decimals`):

```
-- Today's OHLCV for all symbols
SELECT symbol,
       open_price  / power(10, tick_decimals)  AS open_price,
       high_price  / power(10, tick_decimals)  AS high_price,
       low_price   / power(10, tick_decimals)  AS low_price,
       close_price / power(10, tick_decimals)  AS close_price,
       volume
FROM daily_stats
WHERE date = date('now')
ORDER BY symbol;

-- Trade history for AAPL in the last hour
SELECT ts, price / power(10, tick_decimals) AS price, quantity,
       buy_gateway_id, sell_gateway_id, aggressor_side
FROM trade_log
WHERE symbol = 'AAPL'
  AND ts >= datetime('now', '-1 hour')
ORDER BY ts;

-- Intraday mid-price series for charting
SELECT ts, mid_price / power(10, tick_decimals) AS mid_price, pct_change
FROM price_snapshots
WHERE symbol = 'AAPL'
  AND ts >= date('now')
ORDER BY ts;
```

For the full per-table column reference, the complete conversion rule and its exceptions (`vwap`, `turnover`, `order_events.price/fill_pri` and the other six tables (`stats_meta`, `instruments`, `feed_gaps`, `order_events`, `index_daily_stats`, `index_level_snapshots`), see Statistics & Reporting.

28.9 Summary of All Data Files

For the complete file-by-file map — every data file, the process that writes it, its cadence, its purpose, and the tool used to read it — see Data files at a glance near the top of this page. The engine state files (`gtc_orders.json`, `gtc_combos.json`, `book_stats.json`, `engine_run_seq.json`) are described in detail in Engine State Files In Detail above.

Data directory path depends on how EduMatcher is run

The paths shown above use `src/data/` because that is the default for a **developer source checkout** (detected by `config.py` checking whether its own parent directory is named `src`).

28.10 See also

In this book:

- Auctions & Scheduling — how ATO/ATC orders expire at phase transitions
- The Configuration Workflow — `last_buy_price`/`last_sell_price` config seeds vs persisted values

Participant Guide:

- Order Types — TIF — GTC, ATO, and ATC lifetime rules

Reference Manual:

- Processes — pm-stats writes `stats.db`; pm-audit writes `audit.log`

CHAPTER 29

Audit Trail

Learning objectives

After reading this page you will understand:

- Why a full event log matters for an exchange and what `pm-audit` records
- The JSONL log format, how `pm-audit` closes each day and how days are archived
- All `pm-audit` startup options including write buffering
- How to query audit logs from the command line using every `pm-audit-cli` verb
- When to use the optional SQLite index for faster queries
- A practical cookbook for common audit investigation workflows

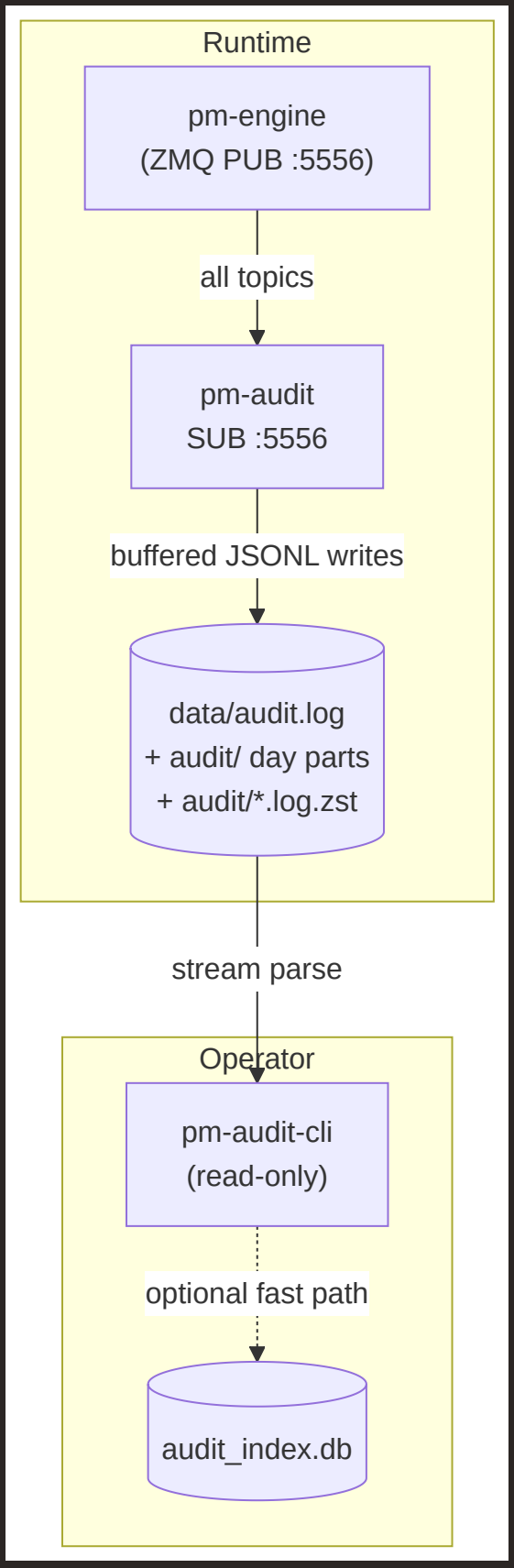
29.1 Overview

EduMatcher maintains a complete, append-only, tamper-evident record of every message that passes through the ZeroMQ bus. This is the **audit trail** — a chronological log of every order, fill, trade, cancellation, session transition, and system event that occurred during a trading session.

The audit system has two independent components:

Component	Role	Type
`pm-audit`	Subscriber — records every event to a JSONL log file, closed day by day	Long-running process
`pm-audit-cli`	Query tool — reads audit logs and prints structured output	One-shot CLI

The recorder and the query tool are completely independent. `pm-audit-cli` reads log files directly from disk and works even when `pm-audit` is not running.



29.1.1 Why a full event log matters

A financial exchange must be able to answer questions after the fact:

- Was a specific order ever acknowledged by the engine?
- What sequence of events led to a particular fill?

- Did a circuit breaker fire before or after an order was submitted?
- Which gateways were active at a given time?

The audit trail answers all of these by preserving every message exactly as it was broadcast, with a millisecond-precision UTC timestamp.

29.2 Data Directory

Audit logs are stored in the same data directory used by all other EduMatcher persistent files:

Running mode	Default location
Source checkout (`poetry run pm-audit`)	`<repo>/src/data/audit.log`
Installed (`pm-audit` on PATH)	`~/.local/share/edumatcher/audit.log`

Override with `EDUMATCHER_DATA_DIR` or `--audit-log-file`:

```
export EDUMATCHER_DATA_DIR="$HOME/sessions/morning"
pm-audit  # writes to $HOME/sessions/morning/audit.log
```

29.3 pm-audit — Event Recorder

`pm-audit` subscribes to all topics on the engine's PUB socket and appends every message as a single JSON line to the configured log file. It also binds a small health socket (port `5562`) that answers `pm-audit-cli health` pings.

ZeroMQ PUB/SUB subscriptions join asynchronously. A publication made before `pm-audit`'s subscription has reached the engine is not queued or replayed; the recorder cannot write an event it never received. Start `pm-audit` before starting `pm-engine`, wait for `pm-audit-cli health` to report the recorder healthy, and verify that the engine's `system.startup` event appears in the audit log before trading. The health check confirms the recorder's receive loop is running; it does not itself acknowledge the upstream subscription. If attaching to an already-running engine, allow the subscription to join before relying on the trail. The write buffer affects when received events appear on disk, not whether the subscription receives them.

```
pm-audit [options]
```

29.3.1 Startup options

Flag	Default	Description
`--audit-log-file PATH`	`<DATA_DIR>/audit.log`	Audit-trail output log file path
`--terminal` / `-t`	off	Also print each entry to stdout (useful during demos)
`--buffer-size N`	`100`	Number of messages to buffer in memory before flushing to disk
`--flush-interval SECONDS`	`10.0`	Maximum seconds to wait before flushing buffer regardless of size
`--max-file-mb MB`	`10`	Close the audit log as a day part when it reaches this size — see Day parts and archives
`--keep-days N`	`0`	Delete daily archives older than N days; `0` keeps them all
`--fsync-interval SECONDS`	`1.0`	Sync the active file to disk at most this often; `0` syncs every flush — see What can be lost

Flag	Default	Description
`--log-level`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	No-op — the default level is already `WARNING`, so `--quiet` currently sets the same level and has no observable effect
`--log-target`	`server`	Where this process's own operational log records go (not the audit trail): `server` (auto-detected `pm-log-srv`), `stdout`, or `file`
`--log-file PATH`	—	Operational log file path — required when `--log-target file`. Distinct from `--audit-log-file`, which is the audit-trail output
`--log-failover-timeout SECONDS`	`30`	Grace window before falling back to a local log file once `pm-log-srv` becomes unreachable

`--log-level` takes priority over `-v`/`-q` when both are given.

29.3.2 Write buffering

Writing every message to disk individually would generate enormous I/O on active sessions. `pm-audit` batches messages in memory and flushes as a group when either limit is reached:

- **Buffer full**: buffer contains `--buffer-size` messages (default 100)
- **Timer expired**: `--flush-interval` seconds have elapsed since the last flush (default 10 s)
- **Shutdown**: Ctrl-C, SIGINT, or SIGTERM always flushes remaining messages before exit

No messages are lost on clean shutdown. A hard kill (`kill -9`) may lose up to `--buffer-size` messages that had not yet been flushed. See What can be lost for the whole picture.

Tuning examples:

```
# Default: good balance of throughput and latency
pm-audit

# High-throughput session: larger buffer, less frequent disk I/O
pm-audit --buffer-size 500 --flush-interval 30

# Low-latency / compliance: flush aggressively
pm-audit --buffer-size 1 --flush-interval 1

# Demo mode: see every event on the terminal as well
pm-audit --terminal
```

29.3.3 What can be lost

`pm-audit` is a subscriber, so the trail holds what it *received*. Four things sit between an engine publication and the disk, and each has a bound:

Where	What is lost	Bound
`pm-audit` is not running	Everything the engine publishes meanwhile	Unbounded. The next line shows a jump in `eseq` (replay: `ESEQ_GAP`), and a `# recorder-start` marker before it (replay: `RECORDER_RESTART`) says why
The receive queue overflows (a burst faster than the writer)	The messages past the queue's 200 000	Counted by `audit.missed_messages` when the next message shows the jump
`pm-audit` is killed (`kill -9`)	Lines buffered but not yet written	At most `--buffer-size` messages or `--flush-interval` seconds, whichever comes first
The host loses power or the kernel crashes	Lines written but not yet synced	About `--fsync-interval` seconds (default 1) plus the line above. A write syncs when one is due, and while no line arrives the receive loop does (it wakes at least every 0.3 s), so a quiet tail is not left in the page cache

A stopped-and-started `pm-audit` writes one comment line, `# edumatcher-audit recorder-start pm-audit=<version> started=<time>`, before its first line, so the restart is visible in the file itself and not only as a hole in the numbering. `pm-audit-replay` reports it as `RECORDER_RESTART` (info) and does not count it as an unreadable line.

The health beacon carries `audit.missed_messages`: the number of engine messages `pm-audit` can prove it missed from forward sequence gaps. It stays 0 on a healthy exchange. An engine restart is not a miss (the sequence begins again at `system.startup`). The counter sees only a jump it observes, so messages lost after the last one received, because `pm-audit` was stopped, appear in the *next* run's trail as the gap, not in this counter.

The separate `audit.sequence_discontinuities` counter records a repeated or decreasing `eseq` without an observed `system.startup`. Such a discontinuity can indicate a missed restart marker or duplicate/out-of-order publication; it does not reveal an exact number of missing messages. Check the trail and recorder health before treating the run as complete.

29.3.4 Log format

Every line is an independent record in this format:

```
[2026-07-08T09:30:00.123+00:00] [trade.executed] [seq=42 msg=01J… cause=01J… chain=01J…] {"id": "TRD-001", ...}
```

Part	Example	Description
`[timestamp]`	`2026-07-08T09:30:00.123+00:00`	UTC ISO-8601 with millisecond precision. This is `pm-audit`'s **receipt** clock, not the publisher's
`[topic]`	`trade.executed`	ZeroMQ topic exactly as broadcast by the engine
`[metadata]`	`seq=42 msg=01J… cause=01J… chain=01J…`	Envelope frames, recorded as `key=value` pairs. Optional — see below
`{...}`	`{"symbol": "AAPL", ...}`	Full JSON payload of the message, byte-identical to what was published

The metadata section

Key	Meaning
`seq`	The publisher's per-topic sequence number. Dense within a run, so a gap means messages were **dropped** — PUB/SUB discards silently once a subscriber falls behind, and this is how you find out
`eseq`	The publisher's count over *all* its messages: one total order of everything the engine published since it started. A gap is a lost message even where `seq` cannot show one — the last message of a topic
`msg`	ULID of this message
`cause`	ULID of the message that caused this one. **Absent means nothing caused it** — a scheduler tick, a circuit-breaker trip — not that the cause is unknown
`chain`	ULID shared by every message descending from one original request

The section is omitted entirely for a message whose publisher stamps no envelope, and lines archived before the section existed are still read correctly, so a mixed archive needs no flag.

The header line and the engine's start

Every file — `audit.log` and each day part — begins with one header line that names the format and the recorder:

```
# edumatcher-audit format=1 pm-audit=1.0.0 started=2026-07-08T00:00:00.002+00:00
```

Readers skip it (it does not start with `[`). The first record the engine publishes on every start is `system.startup`: its version and source commit, the `run_seq` it took, the hashes of the persistence files it is about to restore from, and the compiled configuration it runs on, verbatim. An audit trail therefore describes its own run — a replay can rebuild the engine from that one message, after the data directory's `ref_data/` has been redeployed. Every per-topic `seq` starts again at 1 after it.

The engine also records every book whole, as `system.book_checkpoint.<SYM>`: after startup (what the restore and the seeds made), after each session transition, and at shutdown — each resting order in priority order with its hidden iceberg quantity, each armed stop with its current trigger, and the last prices. A replay can start from any of them, so a day's audit file, or an audit that runs across an engine restart, can be checked on its own.

This is what makes a post-mortem tractable. To follow one order submission through everything it caused:

```
# find the submission
poetry run pm-audit-cli --format json events --topic order.new --limit 1

# then everything in its causal chain, in order
grep 'chain=01ARZ3NDEKTSV4RRFFQ69G5FAV' data/audit.log
```

Lines are appended in arrival order. Within a single session they are chronologically monotonic by receipt time — but note that receipt order across several publishers is not causal order. `cause` and `chain` are what establish causality; the timestamp only establishes when `pm-audit` saw it.

29.3.5 Day parts and archives

`audit.log` always holds **one trading day**: the day is midnight to midnight in the exchange's `locale.session_timezone` (as in `pm-stats`), judged by each line's receive time. At the date change, or when it reaches `--max-file-mb`, `pm-audit` closes it as a numbered part of its day. Parts are never renamed again and **nothing is ever deleted** by the recorder:

```
data/audit.log                          ← active: today's lines, being written
data/audit/audit-2026-10-25.1.log       ← closed parts of a day, in write order
data/audit/audit-2026-10-25.2.log
data/audit/audit-2026-10-24.log.zst     ← one archive per past day
```

A line received at 23:59:59.990 and flushed at 00:00:05 still lands in the old day's part, so every line in `audit-D.*` was received on day D. A day that ends while the exchange is idle, or while `pm-audit` is down, is closed when `pm-audit` next starts.

Archives. Right after closing a day, and once at start-up, `pm-audit` starts a low-priority child process, `pm-audit-cli archive --pending`. It concatenates the day's parts into one zstd stream, `audit-D.log.zst` (13:1 on this data, ~140 MB/s at the default level 9), checks it by decompressing and comparing a SHA-256 and the line count, renames it into place and only then deletes the parts. A crash at any point leaves the parts, or the archive and the parts, never neither; the next run finishes the job. `zstdcat audit-2026-10-24.log.zst | grep …` works, since the archive is the day's log byte for byte.

A 100-trader AI simulation writes about 1.2 GB an hour, so a day is 10–14 GB as parts and about 1 GB as an archive. Disk use is governed by `--keep-days`:

```
pm-audit --max-file-mb 256 --keep-days 7    # what the ai-sim emo profile runs
```

Only archives are ever deleted, and only those older than `--keep-days` days.

Recovering a failed archive by hand. A failed archive leaves the day's parts in place, where they are still read as part of the live trail, and logs an ERROR (`pm-log-srv`) and counts `audit.archive_errors`, which pm-monitor shows as an `audit_archive_errors` WARN. Free the cause (usually disk space: the archive needs about a tenth of the parts' size, but the run refuses below 25 %) and run it yourself:

```
pm-audit-cli archive --date 2026-10-24      # one day, must be before today
pm-audit-cli archive --pending              # every closed day that has parts
pm-audit-cli archive --pending --level 12 --keep-days 30
```

Two runs never overlap; a second one exits at once with a note. A leftover `audit-D.log.zst.part` file is deleted and redone. A day that `audit.log` still holds (`pm-audit` was down at midnight) is not archived until `pm-audit` has closed it at its next start.

An archive is never deleted by a run. Parts found beside it are either what a crash left while deleting them (they are the archive's tail, and are deleted) or lines of that day written after it was archived (they are appended to it). If the archive itself cannot be decompressed, the run refuses that day and leaves both: check it with `zstd -t audit-D.log.zst`, and only if it is damaged and the parts are known to be the whole day, move it aside and re-run.

29.3.6 Proving a day is unchanged: the evidence manifest

Each archive appends one line to `data/audit/manifest.jsonl`: the day, its line and byte counts, and the SHA-256 of its uncompressed content. Every line also carries the SHA-256 of the line before it, so changing or removing any entry breaks every entry after it. Retention writes a `retired` line before it deletes an archive, so the chain survives `--keep-days`. A day reopened after it was archived appends a second entry, and the newest is current.

```
pm-audit-cli verify                    # the whole chain, and every archive against it
pm-audit-cli verify --date 2026-10-24  # the chain, and that day's archive
```

`verify` exits 1 and names the day when an archive's content differs from its entry, an entry is missing, an archive is missing without being retired, or the chain is broken. It ends with the **head hash**, the hash of the newest line.

A crash while a line was being appended leaves it torn (no newline, invalid JSON). `verify` reports it on every run from then on, as it should: the record of that moment is damaged. The next entry starts on a line of its own and chains to the torn line, so everything after it still verifies; the archive whose entry was torn is recorded again when the archiver finishes it.

This makes an edit *evident*; it does not prevent one. Someone who can rewrite the files can rewrite the whole chain, or cut off its newest entries. To rule that out, copy the head hash somewhere the recorder host cannot write (a ticket, an email, a separate system) after each close, and compare it with the one `verify` prints when a day is disputed.

29.3.7 Subscribed topics

`pm-audit` subscribes with an **empty topic filter**, which causes ZeroMQ to deliver every published message without exception:

Topic	Source
`order.new`	Gateway order submissions
`order.ack.*`	Engine order acknowledgements
`order.fill.*`	Fill notifications
`order.cancelled.*`	Cancel confirmations
`order.expired.*`	TIF expiry notifications

Topic	Source
`trade.executed`	Every matched trade pair
`book.*`	Book snapshots after every change (each carries `ts_ns`, so a snapshot can be ordered against the trades around it)
`session.state`	Session phase transitions (`command_id` present when an operator, not the scheduler, requested it)
`system.*`	Gateway auth, symbols, EOD, GTC-restore summary and per-entity restore outcomes, absorbed internal failures
(all others)	Admin events, combos, quotes, circuit breaker

29.3.8 Audit completeness — every state-changing decision, not just its outcome

`pm-audit` can only ever see what the engine publishes on `:5556` — it is a bare subscriber with no other visibility into the process. Reconstructing "what happened and why" from the log is therefore only as complete as the publish call sites are, and a few decision points used to publish an outcome without the reason behind it, or nothing at all. The fields and events below close those gaps:

Gap closed	Where it shows up
Every engine-initiated cancel now states why	`order.cancelled.{GW_ID}`'s `cancel_reason` covers a kill switch (self or admin), a circuit-breaker halt, a gateway disconnect, an admin symbol-level mass cancel, a quote being replaced, and a quote leg cancelled because its sibling filled — in addition to the original `SELF_MATCH_PREVENTED` / `INSUFFICIENT_LIQUIDITY`. Still `null` for a client-requested cancel, which is not the exchange's decision to explain.
Cancels and quote removals join back to the command that caused them	`order.cancelled.{GW_ID}` and `quote.status.{GW_ID}` both carry `command_id` when an admin/kill-switch command caused the event — the same `command_id` echoed on that command's ack and on `admin.action`. `null` when there is no such command (self-match prevention, a participant's own quote cancel, and so on).
A resting order can be told apart from a config-seeded one	`order.ack.{GW_ID}` and `order.fill.{GW_ID}` carry `is_seed` — `true` for an order or quote leg created by `market_maker_quotes`/`market_maker_combos` at startup rather than a live gateway submission (an order's acknowledgement carries it too, and `order.new` itself is in the trail, since the engine echoes every command it is asked to perform before acting on it). The on-demand `order.orders`/`order.price_level_orders` snapshot rows carry it too. Purely observational: nothing in the engine branches on it.
Every combo leg gets an acknowledgement	`order.ack.{GW_ID}` now fires once per child leg when a combo is accepted — live or config-seeded. Previously no combo leg published an ack at all; a subscriber that started after acceptance had no way to learn a leg's symbol, side, price, or qty until it was cancelled, and `order.cancelled`'s minimal shape cannot supply them either.

Gap closed	Where it shows up
Startup recovery is on the record	`system.startup_recovery`, published once right after `_restore_gtc()`, reports how many orders were restored, discarded as stale `TIF=DAY`, or failed to restore; how many single-leg quote remnants and rebuilt `QuoteIndex` entries resulted; and how many GTC combos came back. Previously these counts reached only `log.info/log.error`. `system.recovery_item` now accompanies it with one line per entity — `entity_id`, `kind` (`ORDER`/`COMBO`) and `outcome` (`RESTORED`/`DISCARDED_STALE_DAY`/`FAILED`/`QUOTE_REMNANT`), plus `detail` carrying the exception text for a `FAILED` order — so a single bad GTC record is a wire fact with a reason attached, not just a decremented count.
Kill-switch and admin sweeps say which ids they hit, not just how many	`risk.kill_switch_ack`, `kill_switch_gateway_ack`, `kill_switch_global_ack`, `symbol_halt_ack`, `cancel_symbol_ack`, `circuit_breaker_halt_all_ack`, `circuit_breaker_resume_all_ack`, and `admin.action`'s `scope` all carry an id list beside every count they already reported — `cancelled_order_ids`, `cancelled_quote_order_ids`, `halted_symbol_ids`, `resumed_symbol_ids`, `affected_gateway_ids`. Previously undoing or auditing the effect of a mass cancel meant re-deriving the affected set from `order.cancelled` lines by `command_id`; the ack now says it directly, and a length mismatch against the count is itself a detectable defect.
Cancellations, expiries and amendments can be filtered by symbol	`order.cancelled`, `order.expired` and `order.amended` all carry `symbol` now. `pm-audit-cli events --topic order.cancelled --symbol AAPL` previously returned nothing for any symbol — the engine had `symbol` at every publish site but never put it on the wire.
Book and depth snapshots carry a clock	`book.{symbol}` and `depth.{symbol}` both gain `ts_ns`, stamped from the same strictly-monotonic clock `trade.executed` uses. Previously a snapshot carried no time field at all, so there was no way to tell from the log alone whether a given snapshot reflects a particular trade.
An operator-driven session transition is distinguishable from a scheduled one	`session.state` gains an optional `command_id`, echoing the `command_id` on the `session.transition` that caused it — present only when an operator (not `pm-scheduler`) requested the transition. Previously the broadcast looked identical either way; telling them apart meant inferring it from what else was on the bus at the time.
Absorbed internal failures are on the record	`system.diagnostic` fires for a maintenance-flush exception, a dispatch-handler crash, an undecodable inbound message, or a message on a topic with no handler — cases the engine deliberately survives so trading is not interrupted, but which previously reached only the process log. Carries `component`, `detail` (the flush name or topic, when there is one), `error` (the exception text, when there is one), and `count` (that component's running occurrence count). The client-facing reject for a crashed order handler still stays generic — see `order.ack`'s `INTERNAL_ERROR` — so `system.diagnostic` is the only wire-visible place the real exception text appears.

Gap closed	Where it shows up
Amendments show the prior price/qty	`order.amended.{GW_ID}` carries `old_price/old_qty` alongside the new values, so a post-mortem does not need the preceding `order.new/order.amended` to know what changed.
Index corporate actions show the prior divisor	`index.corp_action_ack`, `index.constituent_change_ack`, and `index.rebalance_ack` all carry `old_divisor` alongside `divisor`, mirroring `HistoryRecord.old_divisor` in pm-index's own local archive — a post-mortem no longer needs that archive to see what changed.

29.3.9 Refusals made at a gateway

The ALF, BALF and REST gateways refuse some requests themselves, before the engine sees them: a command before login, a failed authentication, a request the key's role does not permit, and a client over its rate limit. Each is sent to the engine as `system.gateway_reject` and echoed into the trail, so "I sent it and you ignored it" can be answered either way. A record carries the gateway (`source`), the participant it believed it was talking to (`gateway_id`, claimed and empty before login), the remote address, the reject code, the reason and the gateway's clock. It never carries an API key.

A gateway sends at most one record per peer host and code per second and puts the number it folded in as `count`, so a flood of refused commands is one line a second, not a flood. The host, not the host and port: a client that opens a new connection per request comes from a new port each time. The record names the full `peer` of the refusal that sent it. The count of the last burst is lost if the peer then goes quiet, and the first record of the burst is not.

Not recorded, on purpose: malformed frames and field-validation refusals (missing field, bad value), which the gateways count, and a read-only REST key attempting a write. `pm-audit-replay story` narrates a record as *"A request from GW01 (10.0.0.5:4411) was refused at the ALF gateway — rate limiting"*.

29.3.10 Which clock a line is stamped with

Every audit line starts with the time `pm-audit` *received* it, which is the only clock present on every line. Three messages also carry the **engine's** clock, in Unix epoch nanoseconds: `trade.executed.ts_ns` (the match), `order.fill.ts_ns` (the same value as the trade it reports, so a fill and its print share one reading) and `order.ack.ts_ns` (when the engine accepted or refused the order, from the same strictly increasing source). For "when did the exchange accept this order", use the ack's `ts_ns`; the receipt time can lag it under load. Ordering between messages is by the engine's `eseq`, not by either clock. Lines recorded before this field existed carry the receipt time only.

29.3.11 Commands the trail may not hold

The engine echoes every inbound command into the trail before it acts on it, which is what lets a command that had no effect still be found. A command that reaches the engine **without an envelope frame** (a client that built its own PUSH socket instead of using the project's client library) cannot be echoed, and a failed echo is the other way one goes missing. Both are counted in `engine.diagnostics{component="UNRECORDED_COMMAND"}` in the engine's health beacon, and the first of each thousand is logged at WARNING. On a healthy exchange the counter stays at 0; any other value means the trail is not a complete record of what the exchange was asked to do.

An audited run must also be started from a deployed config artifact (`pm-config-deploy`). With none, `system.startup` carries no config and `pm-audit-replay anomalies` reports `CONFIG_NOT_RECORDED`.

29.3.12 Signal handling and graceful shutdown

```
Ctrl-C / SIGINT / SIGTERM → flush buffer → close socket → exit
```

The buffer flush happens synchronously before the process exits so the last batch of messages is always written to disk.

29.4 pm-audit-cli — Query Tool

`pm-audit-cli` is a read-only command-line tool for querying audit logs without shell pipelines or SQL knowledge. It reads JSONL files directly from disk and produces structured output. The one command that talks to the running recorder is

`health`, which pings `pm-audit` to confirm it is alive.

```
pm-audit-cli [global-options] COMMAND [command-options]
```

29.4.1 Why a separate query tool

`pm-audit` is a long-running writer. Querying is a separate one-shot operation. Keeping them separate means:

- You can query logs while `pm-audit` is still writing
- The tool works on archived logs from previous sessions
- Queries never affect the running recorder

This follows the same pattern as `pm-stats-cli` and `pm-clearing-cli`.

29.4.2 Global options

These flags are parsed once, before the subcommand:

Flag	Default	Description
`--log-file PATH`	`data/audit.log`	Primary audit log file to read
`--log-dir PATH`	(parent of `--log-file`)	Directory holding the `audit/` day parts and archives
`--format table|json|csv`	`table`	Output format
`--no-header`	off	Suppress CSV header row
`--use-index PATH`	(auto)	Path to SQLite index file; auto-detected as `<the directory containing --log-file>/audit_index.db` when present

Not every flag applies to every command

`--use-index` only affects the `events` command — `orders`, `trades`, `topics`, `gateways`, and `timeline` always stream the JSONL log files directly and never consult the SQLite index, even when one exists and `--use-index` is given. `--format`/`--no-header` are also ignored by `stats` (its output is a fixed text summary, not a table) and by `index` (which prints build progress, not rows). `health` honours `--format json` (the report as a JSON object); `csv` prints the same text as `table`.
The auto-detected index path is always `<the directory containing --log-file>/audit_index.db` — it does **not** use `--log-dir`. If `--log-dir` points somewhere other than `--log-file`'s own parent directory, the index lookup still only checks the log file's parent.
Output formats:

Format	When to use
`table`	Interactive terminal — aligned columns, human-readable
`json`	Automation, scripts, downstream processing
`csv`	Export to spreadsheets or pipelines

No-result behaviour:

- `table`: prints `No matching events found.`
- `json`: prints `[]`
- `csv`: prints header only (or nothing with `--no-header`)

Exit code is always `0` when a query succeeds with no results. Non-zero only on argument errors (exit `2`) or I/O failures (exit `1`).

29.4.3 Which files are read

By default `pm-audit-cli` reads the **live trail**: every closed day part that is not yet archived, oldest first, then `audit.log`. That is today, plus any day that has not been archived yet. Archives are not read unless you ask:

You give	It reads
nothing	the live trail
`--date 2026-10-24`	that trading day (session timezone): its archive, or its parts if not archived yet, plus `audit.log` if it still holds that day
`--log-file data/audit/audit-2026-10-24.log.zst`	exactly that archive (a day part path works too)

`--date` selects a day's *files*, not a time window, and cannot be combined with an archive `--log-file` or with `--follow`. Results are chronological across all files.

29.4.4 `events` — General event search

Search log entries by topic prefix, gateway, symbol, and time range. This is the most flexible command and the right starting point for any investigation.

```
pm-audit-cli events [options]
```

Options:

Flag	Default	Description
`--topic PREFIX`	(all)	Filter by topic prefix, e.g. `order.fill` or `trade.`
`--gateway GW_ID`	(all)	Filter by gateway ID appearing in the payload
`--symbol SYMBOL`	(all)	Filter by instrument symbol
`--date YYYY-MM-DD`	(live trail)	Read this trading day's archive or parts — see Which files are read
`--from ISO_TS`	(none)	Start of time range (inclusive)
`--to ISO_TS`	(none)	End of time range (inclusive)
`--limit N`	`100`	Maximum rows returned
`--reverse`	off	Show newest events first
`-f`, `--follow`	off	Keep watching the log and print new events as they arrive; see Follow mode below
`--interval SEC`	`1.0`	Polling interval for `--follow` (ignored otherwise)

Output columns:

Column	Description
`timestamp`	UTC timestamp of the event
`topic`	ZeroMQ topic
`gateway`	Gateway ID extracted from payload (if present)
`symbol`	Instrument symbol extracted from payload (if present)
`order_id`	Order ID extracted from payload (if present)
`summary`	Brief human-readable description of the event

Column	Description

summary format differs when the SQLite index is used

When `events` is served from the JSONL log files, `summary` is a structured, topic-aware description (e.g. `FILL 60@150.00`). When a usable index exists at the resolved `--use-index` path, `events` is served from SQLite instead, and `summary` is just the payload dict's `str()` representation truncated to 80 characters — a different, less readable format. Rebuild the index after querying if you need the latest events; the index is never queried automatically for freshness. `--follow` always reads the JSONL files directly and never uses the index, even when one is available — see below.

Examples:

```
# All events in the log (last 100)
pm-audit-cli events

# All fills for gateway GW01 on a specific date
pm-audit-cli events --topic order.fill --gateway GW01 --date 2026-07-08

# All trade executions for AAPL in a two-hour window
pm-audit-cli events --topic trade.executed --symbol AAPL \
  --from 2026-07-08T09:30:00+00:00 --to 2026-07-08T11:30:00+00:00

# Most recent 50 events across all topics
pm-audit-cli events --limit 50 --reverse

# All session phase transitions
pm-audit-cli events --topic session.state

# All order rejections
pm-audit-cli events --topic order.ack --symbol AAPL

# Export all trade events as JSON for analysis
pm-audit-cli --format json events --topic trade.executed --limit 5000 > trades.json

# Order submissions from a specific gateway in CSV format
pm-audit-cli --format csv events --topic order.new --gateway GW01 > gw01_orders.csv

# Watch fills arrive live during a training session
pm-audit-cli events --topic order.fill --follow

# Watch one symbol, polling every 2 seconds instead of the 1s default
pm-audit-cli events --symbol AAPL --follow --interval 2
```

Follow mode

`events` and `timeline` both accept `-f`/`--follow` to keep the query running: after printing the normal (filtered, `--limit`-capped) result, the command keeps polling the log files every `--interval` seconds and prints each newly-appended, matching entry as it arrives, until interrupted with `Ctrl-C`. This is the audit-log equivalent of `pm-log-cli tail`.

```
# Watch the whole log live, table format
pm-audit-cli events --follow

# Watch order lifecycle events for one gateway, as JSON
pm-audit-cli --format json events --topic order. --gateway GW01 --follow
```

A few things behave differently in follow mode:

- **`--limit` only bounds the initial backfill.** The first batch printed respects `--limit` (`100` by default for `events`, `500` for `timeline`) exactly as it would without `--follow`. Once the poll loop starts, every subsequent batch prints however many new matching entries actually arrived since the last poll — there is no cap. This mirrors `pm-log-cli tail --before N`.
- **`events --reverse --follow` together is rejected** (exit code `2`): `timeline` has no `--reverse` flag, and for `events`, newest-first ordering is meaningless once the log is still growing.

- **`events --follow` always reads the JSONL log files**, even when a usable SQLite index exists at the resolved `--use-index` path. The index is a point-in-time snapshot and is never refreshed by a running query, so follow mode bypasses it rather than showing stale data.
- **Closing a day is handled automatically.** Each poll re-discovers the live trail, so the midnight close — `audit.log` becoming a day part and a fresh `audit.log` starting — is picked up on the next poll without restarting the command, and no line is printed twice.
- **No `seq` requirement.** Resuming after each poll is based on the entries' timestamps, not the optional `seq=` field in the log line's metadata section — so `--follow` works identically whether or not `pm-audit` happens to be stamping envelope metadata.

`--format csv`/`--format json` both work under `--follow`. Each poll batch is printed as its own block, one after another, rather than as a single combined document: for `csv`, only the very first batch gets a header row (later batches never repeat it, regardless of `--no-header`); for `json`, each batch is its own JSON array printed back to back with no separator between them. A consumer streaming the output should parse it incrementally (read one CSV block or JSON array at a time) rather than treating the whole stream as one JSON value or one CSV table.

29.4.5 `orders` — Order lifecycle investigation

Find all audit events related to specific order IDs or filter by gateway and symbol to reconstruct an order's full journey through the engine.

```
pm-audit-cli orders [options]
```

Options:

Flag	Default	Description
`--id ORDER_ID`	(none)	Order ID to find (repeatable; searches `order_id` and `id` fields in payload)
`--gateway GW_ID`	(all)	Filter by gateway ID
`--symbol SYMBOL`	(all)	Filter by symbol
`--date YYYY-MM-DD`	(live trail)	Read this trading day — see Which files are read
`--from ISO_TS`	(none)	Start of time range
`--to ISO_TS`	(none)	End of time range
`--limit N`	`100`	Maximum rows

Output columns:

Column	Description
`timestamp`	Event timestamp
`order_id`	Order identifier
`event`	Last dot-separated segment of the topic. For two-part topics like `order.new` this is the verb (`new`). For per-gateway topics like `order.ack.GW01`, `order.fill.GW01`, or `order.cancelled.GW01` it is the **gateway ID**, not the verb — the topic's own verb segment (`ack`, `fill`, `cancelled`) is dropped.
`gateway`	Gateway ID
`symbol`	Instrument symbol
`side`	`BUY` or `SELL`

Column	Description
`qty`	`quantity or fill_qty or remaining_qty` — the first **truthy** value in that order (not merely the first present key; a legitimate `0` or `null` falls through to the next)
`price`	Price (limit price or fill price depending on event type)
`status`	Order status after the event
`reason`	`reject_code` (from an `order.ack` rejection), `cancel_reason` (from `order.cancelled`), or a generic `reason` string — whichever the event carries

Examples:

```
# Full lifecycle of one specific order
pm-audit-cli orders --id ORD-GW01-00142

# Two orders side by side
pm-audit-cli orders --id ORD-GW01-00142 --id ORD-GW01-00143

# All order events for gateway GW01 today
pm-audit-cli orders --gateway GW01 --date 2026-07-08 --limit 50

# Order events for AAPL in a session
pm-audit-cli orders --symbol AAPL --from 2026-07-08T09:30:00+00:00

# Export one order's lifecycle as JSON for a support ticket
pm-audit-cli --format json orders --id ORD-GW01-00142

# Order activity for market-maker in CSV
pm-audit-cli --format csv orders --gateway MM_AAPL_01 > mm_orders.csv
```

29.4.6 `trades` — Trade execution search

Find `trade.executed` events with precise filtering on both sides of the trade, price ranges, and minimum quantities.

```
pm-audit-cli trades [options]
```

Options:

Flag	Default	Description
`--symbol SYMBOL`	(all)	Filter by instrument symbol
`--gateway GW_ID`	(all)	Match trades where this gateway appears on either side
`--buy-gateway GW_ID`	(all)	Filter by buyer gateway specifically
`--sell-gateway GW_ID`	(all)	Filter by seller gateway specifically
`--min-price PRICE`	(none)	Minimum trade price (inclusive)
`--max-price PRICE`	(none)	Maximum trade price (inclusive)
`--min-qty QTY`	(none)	Minimum trade quantity (inclusive)
`--date YYYY-MM-DD`	(live trail)	Read this trading day — see Which files are read
`--from ISO_TS`	(none)	Start of time range
`--to ISO_TS`	(none)	End of time range
`--limit N`	`100`	Maximum rows
`--reverse`	off	Show newest first

Output columns:

Column	Description
`timestamp`	Trade execution timestamp
`trade_id`	Durable trade identifier, formatted as `run_seq-counter` (for example, `000042-000000001`). The prefix identifies the engine run; the suffix is monotonic within that run.
`symbol`	Instrument symbol
`price`	Execution price
`quantity`	Matched quantity
`buy_gateway`	Buyer gateway ID
`sell_gateway`	Seller gateway ID
`aggressor`	Which side crossed the spread: `BUY`, `SELL`, or `AUCTION` for trades produced by an auction match

Examples:

```
# All trades for AAPL today
pm-audit-cli trades --symbol AAPL --date 2026-07-08

# Block trades (>= 1,000 shares) across all symbols
pm-audit-cli trades --min-qty 1000

# Trades between two specific gateways
pm-audit-cli trades --buy-gateway GW01 --sell-gateway MM_AAPL_01

# MSFT trades in a specific price band
pm-audit-cli trades --symbol MSFT --min-price 410.0 --max-price 420.0

# Most recent 20 trades
pm-audit-cli trades --limit 20 --reverse

# Export full day as CSV for post-trade analysis
pm-audit-cli --format csv trades --date 2026-07-08 --limit 10000 > day_trades.csv

# All trades involving market maker
pm-audit-cli --format json trades --gateway MM_AAPL_01
```

29.4.7 `topics` — Topic discovery and statistics

List all topics present in the audit logs with event counts and first/last occurrence timestamps. Useful for understanding what happened in a session and for checking whether expected event types were produced.

```
pm-audit-cli topics [options]
```

Options:

Flag	Default	Description
`--date YYYY-MM-DD`	(live trail)	Read this trading day — see Which files are read
`--from ISO_TS`	(none)	Start of time range
`--to ISO_TS`	(none)	End of time range
`--prefix PREFIX`	(all)	Filter topics by prefix, e.g. `order.`

Flag	Default	Description
`--sort count|alpha`	`count`	Sort by event count descending (default) or alphabetically

Output columns:

Column	Description
`topic`	ZeroMQ topic name
`count`	Number of events with this topic
`first_seen`	Timestamp of the first occurrence
`last_seen`	Timestamp of the last occurrence

Examples:

```
# All topics in the log, most frequent first
pm-audit-cli topics

# Topics on a specific trading date
pm-audit-cli topics --date 2026-07-08

# Order-related topics only, sorted alphabetically
pm-audit-cli topics --prefix order. --sort alpha

# What happened during the opening auction?
pm-audit-cli topics --from 2026-07-08T09:00:00+00:00 --to 2026-07-08T09:30:00+00:00

# Export all topic stats as JSON
pm-audit-cli --format json topics

# Check that trades were produced today
pm-audit-cli topics --prefix trade. --date 2026-07-08
```

29.4.8 `gateways` — Gateway activity summary

List all gateways seen in the audit logs and show a summary of their activity: order submissions, fills, and trades participated in.

```
pm-audit-cli gateways [options]
```

Options:

Flag	Default	Description
`--date YYYY-MM-DD`	(live trail)	Read this trading day — see Which files are read
`--from ISO_TS`	(none)	Start of time range
`--to ISO_TS`	(none)	End of time range
`--min-events N`	(none)	Only show gateways with at least N events

Output columns:

Column	Description
`gateway_id`	Gateway identifier
`events`	Total event count (all topics)
`orders`	`order.new` events submitted
`fills`	`order.fill.*` events received
`trades`	`trade.executed` events participated in
`first_seen`	First event timestamp
`last_seen`	Last event timestamp

Results are sorted by total event count descending — the most active gateway appears first.

Examples:

```
# All gateways in the logs
pm-audit-cli gateways

# Active gateways on a specific date
pm-audit-cli gateways --date 2026-07-08

# Only gateways with meaningful activity
pm-audit-cli gateways --min-events 10

# Gateway activity in the first hour of trading
pm-audit-cli gateways --from 2026-07-08T09:30:00+00:00 --to 2026-07-08T10:30:00+00:00

# Export as JSON for automation
pm-audit-cli --format json gateways

# Check which gateways connected during the closing auction
pm-audit-cli gateways \
  --from 2026-07-08T15:30:00+00:00 --to 2026-07-08T16:00:00+00:00
```

29.4.9 `timeline` — Chronological event stream

Show a raw chronological stream of events for session replay and forensic investigation. Unlike `events`, the `timeline` command exposes the full JSON payload as a single column, making it suitable for detailed reconstruction of what happened over a time window.

```
pm-audit-cli timeline [options]
```

Options:

Flag	Default	Description
`--from ISO_TS`	(none)	Start time
`--to ISO_TS`	(none)	End time
`--topic PREFIX`	(all)	Filter by topic prefix
`--gateway GW_ID`	(all)	Filter by gateway
`--symbol SYMBOL`	(all)	Filter by symbol
`--limit N`	`500`	Maximum events
`-f`, `--follow`	off	Keep watching the log and print new events as they arrive; see Follow mode above
`--interval SEC`	`1.0`	Polling interval for `--follow` (ignored otherwise)

Output columns:

Column	Description
`timestamp`	Event timestamp
`topic`	ZeroMQ topic
`gateway`	Gateway ID (if present in payload)
`symbol`	Symbol (if present in payload)
`payload`	Full JSON payload as a string

Examples:

```
# Opening 5 minutes of trading
pm-audit-cli timeline \
  --from 2026-07-08T09:30:00+00:00 --to 2026-07-08T09:35:00+00:00

# Gateway activity in a 10-minute window
pm-audit-cli timeline --gateway GW01 \
  --from 2026-07-08T14:00:00+00:00 --to 2026-07-08T14:10:00+00:00

# Symbol-focused replay with higher limit
pm-audit-cli timeline --symbol AAPL \
  --from 2026-07-08T09:30:00+00:00 --to 2026-07-08T16:00:00+00:00 \
  --limit 2000

# All session state changes across the day (timeline has no --date flag;
# use --from/--to to bound a trading day instead)
pm-audit-cli timeline --topic session.state \
  --from 2026-07-08T00:00:00+00:00 --to 2026-07-08T23:59:59+00:00

# Only trades and fills in the closing auction
pm-audit-cli timeline --topic trade. \
  --from 2026-07-08T15:30:00+00:00 --to 2026-07-08T16:00:00+00:00

# Export for external replay tooling
pm-audit-cli --format json timeline \
  --from 2026-07-08T09:30:00+00:00 --to 2026-07-08T16:00:00+00:00 \
  --limit 100000 > session_replay.json

# Watch the full raw event stream live (see Follow mode above)
pm-audit-cli timeline --follow
```

29.4.10 `stats` — Log file statistics

Show a health-check summary of the audit log files: total events, file sizes, date range, and counts of unique topics and gateways. Run this to confirm that `pm-audit` has been recording as expected.

```
pm-audit-cli stats [--verbose]
```

Options:

Flag	Default	Description
`--verbose`	off	Show a per-file breakdown in addition to the summary

Example output (default):

```
Audit Log Statistics
──────────────────────────────────────────────
  Total events:       45,678
  Total size:         12.4 MB
  Log files:          3
  Oldest event:       2026-07-01T09:30:00.123+00:00
  Newest event:       2026-07-08T15:59:59.987+00:00
  Topics seen:        24
  Gateways seen:      5
```

Example output (`--verbose`):

```
Audit Log Statistics
──────────────────────────────────────────────
  Total events:       45,678
  Total size:         12.4 MB
  Log files:          3
  Oldest event:       2026-07-01T09:30:00.123+00:00
  Newest event:       2026-07-08T15:59:59.987+00:00
  Topics seen:        24
  Gateways seen:      5

  Per-file breakdown:
    data/audit/audit-2026-07-07.1.log  12,000 events  4.1 MB
    data/audit/audit-2026-07-07.2.log  18,234 events  6.2 MB
    data/audit.log     15,444 events  2.1 MB
```

Examples:

```
# Quick health check
pm-audit-cli stats

# Full breakdown including per-file stats
pm-audit-cli stats --verbose

# An archived day, by path
pm-audit-cli --log-file data/audit/audit-2026-07-01.log.zst stats
```

29.4.11 `index` — Build or update the SQLite index

For large audit histories or frequent complex queries, `pm-audit-cli` can build an optional SQLite index from the JSONL log files. Once built, the `events` command uses it automatically for significantly faster response.

```
pm-audit-cli index [options]
```

Options:

Flag	Default	Description
`--output PATH`	`<the directory containing --log-file>/audit_index.db`	Destination SQLite database
`--days N`	(none)	Index only the last N days of logs
`--from ISO_TS`	(none)	Start of index range
`--to ISO_TS`	(none)	End of index range
`--rebuild`	off	Delete all existing index data and rebuild from scratch
`--incremental`	off	Add only entries newer than the last indexed timestamp

`--rebuild` and `--incremental` are mutually exclusive.

Re-running index without –rebuild/–incremental duplicates rows

Without either flag, `index` re-inserts every entry in the requested range on each run with no deduplication — running the same `--days 7` command twice doubles the row count for that window. Use `--incremental` for routine re-indexing, or `--rebuild` to start clean.

Index schema highlights:

The index stores one row per log line in a `audit_events` table with dedicated columns for `timestamp`, `topic`, `gateway_id`, `symbol`, `order_id`, and `trade_id`, each with a covering index. The full JSON payload is kept in a `payload` column so no information is lost.

Performance comparison (1 M events):

Query type	JSONL streaming	SQLite index
Recent 100 events	< 1 s	< 0.1 s
Single topic filter	4–6 s	0.2 s
Multi-filter (topic + gateway + date)	10–15 s	0.5 s
Order ID lookup	8 s	0.05 s

Index auto-detection:

When `audit_index.db` exists in the log directory, the `events` command uses it automatically. Override with `--use-index PATH` or point at a different index location.

Examples:

```
# Build index for the last 7 days
pm-audit-cli index --days 7

# Build complete index from all log files
pm-audit-cli index --rebuild

# Incremental update — only adds entries not yet in the index
pm-audit-cli index --incremental

# Index a specific date range
pm-audit-cli index \
  --from 2026-07-01T00:00:00+00:00 --to 2026-07-08T23:59:59+00:00

# Build index to a custom location
pm-audit-cli index --output /tmp/session_idx.db --rebuild

# Use a non-default index when querying
pm-audit-cli --use-index /tmp/session_idx.db \
  events --topic trade.executed --limit 100
```

When to build an index

Build an index when:

- The audit log contains more than ~500,000 events
- You run more than a few queries per session
- You need to correlate events across multiple fields (topic + gateway + symbol + date)

For ad-hoc queries on small logs, streaming JSONL is fast enough without the extra disk space.

29.4.12 `health` — Recorder and log file health

Check that `pm-audit` is alive and actually recording. By default two fast checks run:

- **Process** — a `pm-audit` process is running (found with `pgrep`).
- **ZMQ ping** — `pm-audit` answers a ping on its health socket (port `5562`) with its current UTC time. The reply comes from the recorder's own receive loop, so it proves the loop is running, not just that the process exists.

The log file check is opt-in with `--extended`. It checks that `--log-file` exists and is a regular file, that its last entry is a

well-formed audit line (an empty log is fine — a freshly started `pm-audit` has written nothing yet), and that the file can be opened for append. Nothing is ever written to the log. Only the tail of the file is read, so the check stays fast however large the log grows; it is still bounded at 15 seconds so a stalled or network-mounted volume cannot hang the command.

Unlike the query commands, `health` runs even when the log file does not exist; a missing file is only reported as a failure under `--extended`.

```
pm-audit-cli health
pm-audit-cli health --extended
pm-audit-cli health --extended --verbose
pm-audit-cli --format json health
pm-audit-cli health --quiet
```

```
pm-audit process: OK - pm-audit is running
audit log file:   OK - skipped (pass --extended to check read/write access)
audit ZMQ ping:   OK - replied at 2026-10-17T14:23:37.797827+00:00
health: OK
```

Options:

Option	Description
`--extended`	Also check the audit log file is readable, writable and ends in a well-formed entry (up to 15 s)
`-v`, `--verbose`	List exactly which checks ran; with `--extended`, also print the log size, day parts, archives (with the compression ratio) and last entry time
`-q`, `--quiet`	Print nothing; report only through the exit code

With `--extended --verbose` the report ends with the checks performed and the log file figures. Day parts and archives are found under `<dir>/audit/`, so `--log-dir` applies:

```
Checks performed:
  pm-audit process (pgrep):     yes
  ZMQ health ping:               yes
  Log file read/write/entry:     yes

Log file stats:
  path: data/audit.log
  size: 2.1MB
  day parts: 2 (10.3MB)
  archives: 3 (2.1MB, 13.4:1)
  total size: 14.5MB
  last entry: 2026-07-08T16:29:59.874+00:00
```

The exit code is `0` when healthy and `1` otherwise. With `--quiet` nothing is printed and the exit code is `0` when healthy or `-1` (255 in a shell) otherwise.

The health socket binds to `127.0.0.1:5562` by default; set `EDUMATCHER_AUDIT_BIND_HOST` / `EDUMATCHER_AUDIT_HEALTH_PORT` to change where `pm-audit` binds it, and `EDUMATCHER_ENGINE_HOST` / `EDUMATCHER_AUDIT_HEALTH_PORT` to change where `pm-audit-cli` connects.

29.5 Cookbook — Common Audit Workflows

29.5.1 Investigate a specific order

```
# Find all events for order ORD-GW01-00142
pm-audit-cli orders --id ORD-GW01-00142

# Same as JSON (attach to support ticket)
pm-audit-cli --format json orders --id ORD-GW01-00142

# Find by gateway and time window (when order ID is not known)
pm-audit-cli orders --gateway GW01 \
  --from 2026-07-08T09:30:00+00:00 --to 2026-07-08T09:35:00+00:00
```

29.5.2 Reconstruct a trade

```
# Find trade details by symbol and time
pm-audit-cli trades --symbol AAPL \
  --from 2026-07-08T10:15:00+00:00 --to 2026-07-08T10:15:01+00:00

# Find all large trades above a price threshold
pm-audit-cli trades --min-qty 500 --min-price 150.0 --date 2026-07-08
```

29.5.3 Verify session phases fired correctly

```
# See all session transitions in order
pm-audit-cli events --topic session.state --date 2026-07-08

# Timeline around market open
pm-audit-cli timeline --topic session. \
  --from 2026-07-08T09:25:00+00:00 --to 2026-07-08T09:35:00+00:00
```

29.5.4 Identify the most active gateway

```
pm-audit-cli gateways --date 2026-07-08
```

29.5.5 Check log health before end of day

```
pm-audit-cli health --extended
pm-audit-cli stats --verbose
pm-audit-cli topics --date 2026-07-08 --sort count | head -10
```

29.5.6 Export a full session for offline analysis

```
# Build index first for speed
pm-audit-cli index --rebuild

# Export all trades
pm-audit-cli --format csv trades --date 2026-07-08 --limit 50000 > trades.csv

# Export all order events for gateway GW01
pm-audit-cli --format json events --topic order. --gateway GW01 \
  --date 2026-07-08 --limit 50000 > gw01_orders.json
```

29.5.7 Look at an archived day

```
# The live trail (unarchived parts + audit.log) is the default
pm-audit-cli trades --symbol AAPL --limit 10000

# One past day, read from its archive (or its parts, if not archived yet)
pm-audit-cli trades --symbol AAPL --date 2026-07-07 --limit 10000

# A single archive by path, or straight from the shell
pm-audit-cli --log-file data/audit/audit-2026-07-07.log.zst trades --symbol AAPL
zstdcat data/audit/audit-2026-07-07.log.zst | grep '\[trade\.executed\]'
```

29.5.8 Watch a live training session

```
# Everything, as it happens
pm-audit-cli events --follow

# Just order lifecycle events for one gateway, while proctoring a session
pm-audit-cli events --gateway GW01 --topic order. --follow

# Full raw payloads for the whole floor, piped to a file for later replay
# (each poll batch is its own JSON array back to back -- see Follow mode
# above -- not JSON Lines, so replay tooling should parse incrementally)
pm-audit-cli --format json timeline --follow > live_session.json
```

29.6 Error Handling

Exit code	Meaning
`0`	Success (including queries that returned zero rows)
`1`	I/O error (log file not found, unreadable database)
`2`	Argument error (invalid date format, bad `--limit`, conflicting flags)

Invalid log lines (malformed JSON, truncated writes) are always silently skipped — no warning is printed, and there is no global `--verbose` flag on `pm-audit-cli` to enable one. (`stats --verbose` is a different, per-command flag that only adds a per-file size/count breakdown to the `stats` output.)

29.7 See Also

In this book:

- Persistence — which files each process writes and where they live
- Statistics and Reporting — `pm-stats` and `pm-stats-cli` for structured market data queries

Participant Guide:

- P&L & Clearing — `pm-clearing` and `pm-clearing-cli` for trade settlement

Reference Manual:

- Processes — pm-audit — startup reference table in the process overview

CHAPTER 30

Audit Replay (pm-audit-replay)

Learning objectives

After reading this page you will understand:

- Why a second tool reads the same audit trail `pm-audit-cli` already reads
- The four-layer model — Event, Fact, Episode, Narrative — and what each adds
- Why this is a tool for debugging **the exchange**, not for trading on it
- Every subcommand and option, and which combinations are refused
- The anomaly codes and what each one indicates
- The five output formats and how to interpret text and NDJSON
- A cookbook of investigations, from "what happened to this order" to "is this build's trail clean?"

30.1 Background

The audit trail records every message the exchange published, exactly as it was broadcast. That is the right thing to record and the wrong thing to read.

A single limit order that partially fills produces an `order.new`, an `order.ack`, an `order.fill` for each side, a `trade.executed`, a `drop_copy.fill`, and a `book` snapshot — six to eight lines, interleaved with everything else the exchange was doing in the same millisecond. Answering *"why did this order not fill?"* from that file means reconstructing, by hand and by eye, facts the log contains but does not state:

- Which `order.fill` belongs to which `trade.executed`
- Whether the ack really preceded the fill, or only appears to because of the order the recorder happened to write them in
- Whether `price_ticks: 7569` is 75.69 or 7.569 for this symbol
- Which of the six cancellations a kill switch produced were the ones it claimed

`pm-audit-cli` answers *"which lines match this filter?"*. `pm-audit-replay` answers *"what happened, in order, and why?"* — it reconstructs the causal structure the trail records and then narrates it in English.

Both read the same files, read-only, and neither needs the other. The recorder does not have to be running.

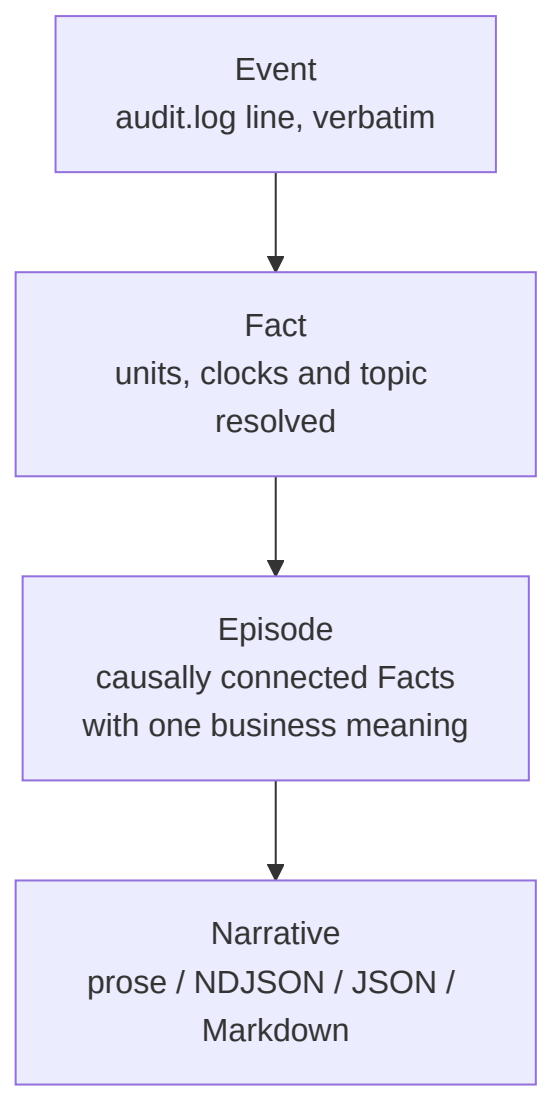
This is a tool for debugging the exchange, not for trading on it

Every other chapter in this guide describes something a participant, an operator, or a student uses to run or observe a market. This one does not. `pm-audit-replay` exists to answer questions about **the exchange's own correctness** — did the matching engine publish what it should have, in the order it should have, with the quantities it should have.
Its audience is whoever is changing the engine, the gateways, or the message specification, and its findings are stated in terms of internal invariants rather than in terms of anything a trader can act on. A trader who wants to know what happened to an order should use the Trading GUI, the drop copy feed, or `pm-audit-cli`.

30.2 How it works

30.2.1 Four layers

Each layer adds exactly one thing, which is what keeps the narrator from becoming a pile of per-topic special cases.



Layer	Adds	Example
Event	Nothing — the line as `pm-audit` wrote it	`[..] [order.fill.TRADER01] {...}`
Fact	Resolved units, clocks and wildcard topics. Removes ambiguity; adds no information	`price_ticks: 7569` → `75.69`, `order.fill.TRADER01` → kind `order.fill`, actor `TRADER01`
Episode	Grouping. One order, one trade, one quote, one command — with its outcome and its derived facts	"order `4f2c9a…`, PARTIAL, 150 of 200 filled, VWAP 74.80"
Narrative	Wording. A pure function of the episode graph	"TRADER01 submitted buy limit 200 AAPL @ 75.69"

30.2.2 Where the causality comes from

Every message the exchange publishes carries a **causal envelope**: a `msg_id` (a ULID), the `causation_id` of the message that caused it, and a `correlation_id` naming the whole chain. So in the common case the tool does not infer causality — it *reads* it.

Where an envelope is absent (an archived log from before the envelope landed) the resolver falls back through four tiers, and every link it produces carries a confidence:

Confidence	Meaning
`RECORDED`	The publisher stated it, via `causation_id`
`CERTAIN`	A structural identity in the payload, e.g. `trade_ids` naming the trade
`STRONG`	An unambiguous match on ids and time
`HEURISTIC`	A plausible match. Always hedged in the prose — never stated as fact

`pm-audit-replay stats` reports the mix. On a log captured since the envelope landed, `RECORDED` and `CERTAIN` should account for essentially all of it.

30.2.3 Ordering

The trail is written in *receipt* order, which is not always *publication* order. Replay uses the publisher's socket-wide `eseq` to recover publication order: an engine can preserve an inbound gateway message's `msg_id`, so that ULID's timestamp does not always say when the engine published it. `eseq` increases across topics and resets when the engine publishes `system.startup`; replay assigns a local `publisher_run` number at each such boundary and orders by that run and `eseq`. This `publisher_run` is a replay ordering aid, **not** the engine's payload `run_seq`.

For older records without `eseq`, replay falls back to the ULID timestamp, then audit receipt time and read order. In logs mixing sequenced and unsequenced records within one run, their relative order cannot be established from the available metadata. The `receipt_ts` shown in output remains the audit receipt timestamp; it is not replaced by `eseq`.

A **reorder window** bounds that sort so a log larger than memory can still be read: `--reorder-window` defaults to 2000 facts or 5 seconds, whichever comes first. A fact arriving after its window has closed is emitted where it landed and tagged `LATE_ARRIVAL` rather than silently misplaced.

Why eseq is the ordering authority

The engine stamps `eseq` immediately before publishing each message on its socket. The `msg_id` may belong to a gateway and be preserved unchanged during republishing, so ordering by its timestamp can place the message before events the engine actually published first. `msg_id` remains the identity used for causality; `eseq` tells replay when the engine published the recorded message.

30.2.4 The episode index

Reconstruction is a single streaming pass with bounded memory, so `--no-index` works on any log. The optional SQLite index at `data/audit_replay.db` materialises the episode model so that `story` can find one order without re-reading the log.

Property	Behaviour
Built	Automatically, when a command needs it and it is absent or stale
Covers	The whole log, always — never the window of the query that built it
Stale when	The source logs changed, the reconstruction rules changed, or the detection settings changed
Incremental	**No.** A rebuild is the whole story — see AR-3.4 in the design
Size	Roughly 4.4× the log (it keeps every payload, which is what makes it a complete substitute)
Build rate	~5 500 events/s; 87 s for a 205 MiB log on the dev VM

Property	Behaviour
Discarded	When its schema version is not current — no migration, by design

`story` requires the index. Everything else works with or without it, and is required to produce identical output either way.

Why the index ignores the window

A cache may not depend on the question that populated it. Narrowing the build by `--from`/`--to` produced an index whose currency check — a source-file fingerprint — could not see the window it was built to, so a later, wider query was answered from a truncated model and said nothing about it. Putting the window into that check would be worse: `--last` is relative, so it moves, and almost every call would rebuild.
The cost is that a narrow question against a large log pays for a full build once. `--no-index` is the way to avoid paying it at all on a one-off trail.

30.3 Subcommands

Subcommand	What it does	Needs the index
`stream`	Narrates a window chronologically. The default view	No
`story`	Narrates one entity and everything causally connected to it	**Yes**
`digest`	One paragraph per episode, most significant first	No
`episodes`	The index as a table, one row per episode	No
`anomalies`	Every finding, worst first, each with the `story` command to run next	No
`stats`	Link confidence, envelope coverage and anomaly counts over the window	No
`rerun`	Feeds the trail's commands to a fresh engine and compares what it publishes	No
`queue`	Follows one resting order, or one price level, through the queue (observes the re-run)	No
`index`	Builds or refreshes the episode index; `--stats` reports what it holds	—

30.3.1 `story` selectors

Exactly one is required.

Selector	Follows
`--order ORDER_ID`	An order (an unambiguous prefix will do)
`--trade TRADE_ID`	A trade and both its legs
`--quote QUOTE_ID`	A quote and its derived leg orders
`--oco OCO_ID`	An OCO pair
`--combo COMBO_ID`	A combo and its legs

Selector	Follows
`--command COMMAND_ID`	A risk or admin command and its effects
`--client-tag TAG`	Whatever carries the client's own tag
`--chain ULID`	A whole causal chain — the complete descent, no `--depth` needed
`--msg ULID`	One message and what it caused

`--depth N` (default 2) bounds link-following for every selector except `--chain`. `--strict-causality` follows only `RECORDED` links, so the result contains nothing the tool inferred.

30.3.2 Per-subcommand options

Subcommand	Option	Meaning
`index`	`--stats`	Report what the index holds once it is current
`digest`	`--top N`	Show only the N most significant episodes
`digest`	`--significance {anomalies,notional,qty,duration}`	What makes an episode significant (default: anomalies, then notional)
`episodes`	`--outcome OUTCOME`	Restrict to outcomes, e.g. `REJECTED` (repeatable)
`anomalies`	`--severity {error,warn,info}`	Report findings at this severity or worse (default: `info`)

30.4 Global options

Every option below is accepted by every subcommand, as are `-h`/`--help`. `--version` is accepted by the top-level command.

30.4.1 Source

Option	Default	Meaning
`--log-file PATH`	`<DATA_DIR>/audit.log`	The trail to read: the live trail — every not-yet-archived day part, then this file. An archive or day part named here is read alone
`--db PATH`	`<DATA_DIR>/audit_replay.db`	The episode index
`--no-index`	off	Stream without building or reading an index
`--rebuild`	off	Rebuild the index before rendering

Day parts are part of the trail

`--log-file audit.log` reads the unarchived day parts in `audit/` beside it as well. That is right for a real trail and surprising for a scratch one: a previous run's parts left in the same directory are replayed alongside the current one. If two runs replay the same seeded dataset, every order appears twice and the tool correctly reports each as `ACK_DUPLICATE`.

30.4.2 Window

Option	Meaning
`--from ISO_TS`	Start of window (ISO-8601, or `YYYY-MM-DD`)
`--to ISO_TS`	End of window
`--date YYYY-MM-DD`	Replay this trading day (session timezone) from its archive, or from its parts before it is archived. Not a time window: the day is its files
`--last DURATION`	Relative window: `15m`, `2h`, `1d`

`--date` cannot be combined with `--from`/`--to`, with `--last`, or with an archive given as `--log-file`; `--last` cannot be combined with `--from`/`--to`. `--from`, `--to` and `--last` are UTC, which is what the trail records — `--tz` changes what is displayed, never what is selected. Yesterday replays from its archive with `--date`; today replays by default, across the midnight close, with no event repeated or missing. The episode index follows the file set, so it rebuilds by itself once a day has been archived.

The window narrows the **query**, never the index: an index always covers the whole log, so the same one answers a wide question and a narrow one. Passing a window to the `index` subcommand is refused rather than ignored.

30.4.3 Filters

Option	Meaning
`--symbol SYMBOL`	Restrict to one or more symbols (repeatable)
`--gateway GW_ID`	Restrict to one or more gateways (repeatable)
`--kind KIND`	Restrict to episode kinds (repeatable)

Episode kinds are `order`, `trade`, `quote`, `oco`, `combo`, `command`, `market_phase`, `session`, `gateway`, `index`, `recovery` and `orphan`.

Outcomes are `FILLED`, `PARTIAL`, `CANCELLED`, `REJECTED`, `EXPIRED`, `ACCEPTED`, `DENIED`, `OPEN` and `UNKNOWN`.

30.4.4 Output

Option	Default	Meaning
`-q`	—	Detail level 0: episode outcomes only
`-v`, `-vv`, `-vvv`	level 1	Raise the detail level
`--format {text,ndjson,json,markdown,csv}`	`text`	Output shape
`--show-source`	off	Append `audit.log:LINE` to every narrated line
`--show-units`	off	Append unit provenance to every price
`--explain`	off	Show link evidence and confidence inline
`--id-len N|full`	`6`	Order-id abbreviation. Lengthened automatically if two ids would collide
`--actor-style {id,descriptive}`	`id`	`TRADER01`, or `TRADER01 (Nordic Equities desk)`
`--reorder-window SPEC`	`2000/5s`	Reorder buffer, as a fact count or a duration

Option	Default	Meaning
`--tz TZ`	`UTC`	Render timestamps in this zone. An unknown zone is an argument error
`--no-color`	off	Never emit ANSI colour, even to a terminal
`--force-color`	off	Always emit it, even when stdout is not a terminal

30.4.5 Colour

Colour is on when stdout is a terminal and off otherwise, so a redirect or a pipe gets plain text without anyone asking:

```
pm-audit-replay anomalies --severity warn            # coloured
pm-audit-replay anomalies --severity warn > run.txt  # plain, automatically
pm-audit-replay stream | wc -l                       # plain, automatically
```

Three ways to end up plain, and two of them need no flag:

`--no-color`	The caller asked
`NO_COLOR` set in the environment	The convention the rest of the tree follows
stdout is not a terminal	A redirect, a pipe, a CI job, a captured test

Paging

That last rule is a guess at what is on the other end of the pipe, and there is one common case where it guesses wrong: a pager. `less` ends at a terminal the tool cannot see, so the colour is switched off exactly when it would have been most useful — a long `stream` is the output most worth paging.

`--force-color` overrides the guess. `less` needs `-R` to render the escapes rather than print them:

```
pm-audit-replay stream -vv --force-color | less -R
pm-audit-replay anomalies --severity warn --force-color | less -R
```

Set `LESS=-R` in your shell profile and the `-R` becomes unnecessary.

`--force-color` beats `NO_COLOR` in the environment — a flag you typed is a more specific instruction than a default you may not remember setting. It cannot be combined with `--no-color`; asking for both is an argument error rather than a silent winner.

What gets a hue is what a reader scans for, and nothing else — a report where every column is coloured is one where no column stands out:

Where	What
Every narrated line	The clock is dimmed: findable when scanned for, out of the way when not
`-v` and above	The annotation lines beneath a sentence are dimmed as a group — they are supporting evidence, and reading a page of them at full weight is what makes `-vvv` unreadable
`anomalies`	`ERROR` red, `WARN` yellow, `INFO` cyan; the code in bold; the `->` story suggestion dimmed
`episodes`	The outcome column — `REJECTED`/`DENIED` red, `CANCELLED`/`EXPIRED` yellow, `FILLED` green, `OPEN`/`UNKNOWN` dim
`digest`	The anchor in bold, the derived numbers dimmed, a finding line red or yellow by its worst severity
`stats`	Section headings in bold

`ndjson`, `json`, `markdown` and `csv` are never coloured, whatever stdout is. They are formats something else reads.

Colour is applied after the layout, never before

`episodes` pads each cell to a column width and paints afterwards. The other order pads to the wrong width, because an escape code is characters the terminal never shows — the table would look ragged by exactly the length of the codes.

30.4.6 Detection

Option	Default	Meaning
`--strict`	off	Also report findings that are expected at a window edge (`ARRIVAL_SEQ_GAP`)
`--clock-skew-warn SECONDS`	`0.1`	Engine/receipt clock difference worth reporting

Detection settings are part of what makes an index stale: changing them forces a rebuild, so a stored index can never disagree with the flags that produced it.

30.4.7 Exit codes

Code	Meaning
`0`	Success
`1`	Nothing to narrate, or no audit log at the given path
`2`	Argument error, including a format a subcommand cannot produce
`3`	`rerun` or `queue`: the trail cannot be re-run (see Re-executing the trail)

30.5 What it can catch

Reconstruction gives consistency checks almost for free, and these are the real prize: a bug that is hard to find in the raw log is usually a bug that breaks one of these invariants.

Each finding names the episode it belongs to and prints the `story` command that shows it, because the next thing anyone does with a finding is go and look at it.

Three severities: **error** is a broken invariant, **warn** is something a reader should look at, **info** is something expected at a window edge.

30.5.1 Lifecycle

Code	Severity	Condition
`ACK_MISSING`	warn	An `order.new` with no `order.ack` inside the window
`ACK_DUPLICATE`	error	One order accepted twice
`FILL_BEFORE_ACK`	warn	A fill whose canonical key precedes its ack — a genuine inversion, not a reorder artefact
`FILL_AFTER_TERMINAL`	error	A fill for an order the trail has already ended

Code	Severity	Condition
`ILLEGAL_STATUS_TRANSITION`	error	A status change outside `NEW → PARTIAL → FILLED | CANCELLED | REJECTED | EXPIRED`
`TERMINAL_MISSING`	info	An order still open when the window ends
`CANCEL_UNSOLICITED`	warn	`order.cancelled` with neither a request in its episode nor a `cancel_reason` naming a cause
`CANCEL_UNMATCHED`	warn	`order.cancel` with nothing coming back
`FILL_STATUS_DISAGREE`	error	An `order.fill` whose `status` and `remaining_qty` contradict each other — `FILLED` with quantity left, or `PARTIAL` with none

30.5.2 Quantity and price conservation

Code	Severity	Condition
`QTY_MISMATCH`	error	`Σ fill_qty ≠ quantity − remaining_qty` — the tool's own tally against the engine's
`REMAINING_NOT_MONOTONIC`	error	`remaining_qty` rose across fills of one order
`TRADE_LEG_MISSING`	error	A `trade.executed` with fewer than two `order.fill` legs naming it
`FILL_WITHOUT_TRADE`	error	An `order.fill` whose `trade_ids` name no `trade.executed` in the window
`LEG_QTY_DISAGREE`	error	The two legs of one trade report different `fill_qty`
`LEG_PRICE_DISAGREE`	error	The two legs report different `fill_price`
`PRINT_QTY_DISAGREE`	error	A non-coalesced leg whose `fill_qty` is not the trade's printed `quantity`. Two equally wrong legs agree with each other; neither can argue with the tape
`PRINT_PRICE_DISAGREE`	error	The same for `fill_price` against the trade's `price`
`TRADE_LEG_UNKNOWN`	error	A leg whose `order_id` is neither the trade's `buy_order_id` nor its `sell_order_id`
`LIQUIDITY_FLAG_DISAGREE`	error	A `liquidity_flag` that contradicts the trade's `aggressor_side`. A billing invariant — maker and taker fees invert on it
`DROP_COPY_DISAGREE`	error	A drop copy whose `fill_qty`, `fill_price` or `symbol` differs from the `order.fill` it copies

Code	Severity	Condition
DROP_COPY_MISSING	warn	A trade that did not produce two drop copies, one per counterparty. Silent when the window holds no drop-copy feed at all
PRICE_THROUGH_LIMIT	error	A fill outside its order's limit — a buy above it or a sell below it
PRICE_OUTSIDE_CORRIDOR	warn	A print outside the circuit-breaker corridor in force for its symbol

Why both tallies are kept

QTY_MISMATCH compares what the fills add up to against what the engine said was left. Keeping both numbers is the point: it is what caught the engine publishing `fill_qty` as a running total rather than as the quantity matched in that event, which every consumer summing the field was over-counting.

30.5.3 Ordering, clocks and the envelope

Code	Severity	Condition
SEQ_GAP	error	A gap in a topic's dense per-topic `seq` — a **proof** that messages are missing from the trail
ESEQ_GAP	error	A gap in the engine publisher's socket-wide `eseq` — messages are missing across the stream, including a topic's last message
ESEQ_DISCONTINUITY	error	A repeated or decreasing `eseq` without a `system.startup` boundary
TRADE_COUNTER_GAP	error	A gap in the per-run trade counter. Redundant with SEQ_GAP, kept because it localises the loss to the trade stream
MSG_ID_DUPLICATE	error	Two messages with the same `msg_id`. Should be impossible; would mean a ULID generator shared unsafely across threads
CHAIN_BROKEN	error	An effect whose `correlation_id` differs from its cause's, which breaks `story --chain`
CAUSE_NOT_FOUND	warn	A `causation_id` naming a `msg_id` nowhere in the window. Usually the window starts too late
CLOCK_SKEW	warn	The engine's own clock and `pm-audit`'s receipt clock differ by more than `--clock-skew-warn`
LATE_ARRIVAL	info	A fact arrived after its reorder window had closed
CLIENT_CLOCK_ABSURD	info	An `order.new` client clock more than an hour from receipt. Harmless — priority is `arrival_seq` — but worth knowing

Code	Severity	Condition
`DROP_COPY_SEQ_GAP`	error	A gap **or a repeat** in the drop-copy feed's own payload `seq`, tracked independently per gateway. Not covered by `SEQ_GAP`, which reads the audit metadata's per-topic sequence
`ARRIVAL_SEQ_REUSED`	error	A repeated or decreasing `arrival_seq` within one run: two orders claiming one queue position. Always reported, unlike the gap below
`ARRIVAL_SEQ_GAP`	info	A gap in `arrival_seq` within one run. Expected whenever the window omits another gateway's orders, so `--strict` only
`RUN_SEQ_CHANGE`	info	An engine restart observed mid-window. Not a fault, but nothing may be compared across it
`RECORDER_RESTART`	info	`pm-audit` was (re)started just before this line, so nothing was recorded while it was down. See What can be lost
`ENVELOPE_MISSING`	info	An engine-published message with no envelope. Expected on archived lines; on a current log, a publisher is bypassing `CausalPublisher`

30.5.4 Command reconciliation

Code	Severity	Condition
`EFFECT_COUNT_MISMATCH`	warn	An ack's list of affected entities disagrees with what was observed. The finding names which are missing
`COMMAND_UNACKED`	warn	A risk or admin command with no ack carrying its `command_id`
`ACK_WITHOUT_COMMAND`	warn	An ack whose `command_id` matches no request in the window: the window starts after the command, or its echo was lost
`RESUME_WITHOUT_HALT`	warn	A resume for a symbol with no halt on record
`HALT_UNRESUMED`	info	A halt with no resume by the time its episode retired

30.5.5 Coverage — the tool admitting what it does not know

Code	Severity	Condition
`PARSE_FAILURE`	error	A line the audit format does not match, detected by the hole it leaves

Code	Severity	Condition
`UNKNOWN_TOPIC`	warn	A topic with no entry in the generated registry: the message spec has grown and the tool has not
`TICK_SCALE_UNKNOWN`	warn	A tick-scaled price with no `tick_decimals` for its symbol anywhere in the window. The price is reported **in ticks and labelled as ticks** rather than guessed
`UNKNOWN_ENUM`	warn	An enum value with no lexicon entry. The value is printed verbatim in backticks rather than guessed at or dropped. It is a renderer annotation, not a finding in the `anomalies` report or an NDJSON `anomaly` object
`ORPHAN_EVENT`	info	A fact the resolver could not attach to anything
`CONFIG_NOT_RECORDED`	warn	A `system.startup` with no compiled config: the engine ran from a YAML file, so the trail does not hold the limits, collars and tick sizes it ran on. Deploy with `pm-config-deploy` for any run that must stand as evidence

30.5.6 What a clean run does not prove

The catalogue above is checked against every message the trail contains. Two families of check are in it and cannot fire on a trail `pm-audit` recorded, because the trail does not carry what they read:

Check	Why it is dormant
`DROP_COPY_SEQ_GAP`, `DROP_COPY_DISAGREE`, `DROP_COPY_MISSING`	The drop-copy feed is published on its own socket (`DROP_COPY_PUB_ADDR`, default `:5557`); `pm-audit` subscribes only to the engine event socket (`ENGINE_PUB_ADDR`, default `:5556`). No drop copy reaches a trail recorded by `pm-audit`, so these checks cannot fire on that trail
`ARRIVAL_SEQ_REUSED`, `ARRIVAL_SEQ_GAP`	`order.new` is the command as the engine received it and `arrival_seq` is stamped when the book accepts the order, so the recorded field is `0` — "unassigned" — on every order

Both are properties of what is recorded rather than of the checks. They fire the moment a trail carries the fields, which is why they are implemented rather than deferred.

A clean run is a claim, not a hope

`tools/verify_audit_trail.sh` records a full `verify_matching.sh` run with `pm-audit` and then runs `anomalies --severity warn` over the trail it left. Any finding at warn or above fails the script. It is the end-to-end check that the engine's published account of itself is internally consistent, and it is expected to come out green.

30.6 Examples

30.6.1 What happened to this order?

```
pm-audit-replay story --order 4f2c9a
```

```
09:31:02.120  TRADER01 submitted buy limit 200 AAPL @ 75.69 (good for the day) — order 4f2c9a…
09:31:02.121  Engine accepted 4f2c9a…
09:31:02.122  AAPL traded 150 @ 74.80 — TRADER01's buy 4f2c9a… took from TRADER02's sell 9ab1c4… (trade 000042…)
09:31:02.122  4f2c9a… took 150 @ 74.80 from TRADER02's sell 9ab1c4…
09:31:02.122  9ab1c4… was lifted for 150 @ 74.80 by TRADER01's buy 4f2c9a…
```

An unambiguous prefix is enough. Both sides of every trade are narrated, not just the side that was asked about.

30.6.2 …and show me the evidence

```
pm-audit-replay story --order 4f2c9a -v --explain
```

```
09:31:02.120  TRADER01 submitted buy limit 200 AAPL @ 75.69 (good for the day) — order 4f2c9a… — a direct submission
              ∎ the publisher recorded that nothing caused this
09:31:02.121  Engine accepted 4f2c9a…
              ∎ caused 01M205PK330000000000000001 [recorded: causation_id]
09:31:02.122  AAPL traded 150 @ 74.80 — … — notional 11,220.00
              ∎ caused 01M205PK330000000000000001 [recorded: causation_id]
              ∎ matched with 01M205PK330000000000000001 [certain: buy_order_id]
```

Every link states its confidence and the evidence behind it. *"the publisher recorded that nothing caused this"* is a declared origin — different from *"no cause was found"*, and never rendered as one.

30.6.3 What did the exchange do in the last fifteen minutes?

```
pm-audit-replay stream --last 15m
pm-audit-replay stream --last 15m -q          # outcomes only
pm-audit-replay stream --date 2026-09-08 --symbol AAPL --symbol MSFT
```

30.6.4 Is anything wrong?

```
pm-audit-replay anomalies --severity warn
pm-audit-replay anomalies --severity error --date 2026-09-08
```

```
WARN   09:44:10.019  EFFECT_COUNT_MISMATCH  command 8812
       risk.kill_switch_ack for command 8812 names 3 cancelled order(s); 1 were not observed: aaaa0001…
       -> pm-audit-replay story --command 8812 -v --explain

1 finding(s) (1 warn) across 7 episode(s).
```

30.6.5 Did a kill switch do what it said?

```
pm-audit-replay story --command 8812 -v --explain
```

30.6.6 Follow one whole causal chain

```
pm-audit-replay story --chain 01ARZ3NDEKTSV4RRFFQ69G5FAV
```

`--chain` is the complete descent and needs no `--depth` tuning. Every message in the chain carries the same `correlation_id`, so this is one indexed read rather than a graph walk.

30.6.7 What is worth looking at first?

```
pm-audit-replay digest --top 20
pm-audit-replay digest --significance notional --top 10
```

30.6.8 Every rejected order as a table

```
pm-audit-replay episodes --outcome REJECTED
pm-audit-replay episodes --outcome REJECTED --format csv > rejects.csv
```

30.6.9 Is the trail itself trustworthy?

```
pm-audit-replay stats --date 2026-09-08
```

```
Envelope coverage
  with envelope         10  100.0%  ########################
  declared origin        6
  orphan                 0

Link confidence
  RECORDED         4   57.1%  ##############..........
  CERTAIN          3   42.9%  ##########..............
  STRONG           0    0.0%  ........................
  HEURISTIC        0    0.0%  ........................
```

Low envelope coverage on a current log means a publisher is bypassing `CausalPublisher`. A high `HEURISTIC` share means the narrative is more inference than record, and should be read as such.

30.6.10 Re-executing the trail

Everything above reads what the trail *says*. `rerun` checks it against the engine:

```
pm-audit-replay --log-file data/audit.log rerun
```

The engine echoes every command it is asked to perform into the trail before it acts, so the trail is the engine's complete input. `rerun` builds a fresh engine from the configuration in `system.startup`, feeds it each command in order, and compares what it publishes with what the live engine published for that command. It also fires the engine's own timers where the trail names them: a circuit breaker reopening, and a gateway disconnected for missed heartbeats.

```
re-ran 8 commands (order.new 2, system.gateway_connect 3, ...)
hidden inputs: 0
outputs: 0 differing episodes; run made with 0.72.1, re-run with 0.72.1; ...
final book: 0 differences
```

Line	Means
`hidden inputs`	State changes the live engine published with no command or named timer behind them. The trail does not hold what caused them, so the re-run cannot reproduce them
`outputs`	Commands after which the fresh engine published something different. A difference is non-determinism, a hidden input, or a change in the engine since the run, so re-run with the engine version the run used; the line names both versions
`final book`	The re-run's resting book against `gtc_orders.json` beside the trail. Said plainly when the file is not there

`rerun` reads every file in the data directory, archived days included, from the first line on. An engine restart in the trail is re-run as a restart: the fresh engine persists its book as the live one did on its way down, and the next run's engine starts from those files, on that run's own configuration and run sequence. Each start's book is compared with the live engine's `STARTUP` checkpoint, so a restart that came back with a different book (after `kill -9` the engine restores its last periodic checkpoint, not the book it had) is counted under `outputs`, and the note says how many restarts were re-run.

Exit `0` is clean, `1` a difference, `3` the trail cannot be re-run. That is when:

- the first line is not an engine start, or that start restored orders from files the trail does not hold (retention removed the day it began empty);
- a run carries no config (the engine ran from a YAML file; see `CONFIG_NOT_RECORDED`);
- the trail provably misses engine output: the engine-wide sequence (`eseq`) jumps, or `pm-audit` restarted while the engine ran. A recorder restart between an engine's `system.eod` and its next start loses nothing, and neither does one after which `eseq` simply continues.

It reads the whole trail in one pass, so allow minutes for a full simulated day.

30.6.11 Where was my order in the queue?

`story` shows an order's own events. `queue` shows its *level*: where it stood among the orders at its price, what traded or was cancelled ahead of it, and whether the market ever reached it. It observes the re-run, so the positions are the engine's own, with its rules for time priority, iceberg slices and amends.

```
pm-audit-replay --log-file data/audit.log queue --order 2b153da6
pm-audit-replay --log-file data/audit.log queue --symbol AAPL --price 264.15
```

```
order 2b153da6  BUY 50 AAPL @ 264.15  (TRADER01)
07:00:00.118  rested at 264.15: 4 of 4 in its level, 1,300 shares ahead (3 orders); ahead of it: MM01 1,000, TRADER02 100, TRADER02 20
07:00:09.402  TRADER02's order ahead (100 shares) was cancelled by its owner -> now 3 of 3 in its level, 1,200 shares ahead (2 orders)
07:00:31.870  1,150 shares traded ahead of it (MM01, TRADER02) (trade 000001-000000001, 000001-000000002) -> now 2 of 2 in its level,
07:00:44.015  50 shares traded ahead of it (TRADER02) (trade 000001-000000003); filled 10 of 50 (trade 000001-000000004), 40 left -> n
07:01:12.560  cancelled by its owner
verdict: cancelled by its owner with 40 of 50 shares unfilled; 4 trades at its price (1,210 shares) while it rested; at the last look
```

The order view ends in a **verdict**: filled in full, cancelled or expired with quantity left (and how much stood ahead), never queued, or still resting at the end of the trail. A stop waits outside the level until a trade fires it: the timeline says when it was armed, which trade triggered it and what it became, and then follows it into its level. One never triggered ends in "still armed" or "never triggered" with its stop price. For a resting order the verdict also says whether the market ever traded at its price; if not, it gives the best opposite price against the order's limit, which is usually the whole answer to "why did it not fill". The level view lists each change at one price (`--side` narrows it) with what the level held after it.

Quantity *ahead* is what is displayed. An iceberg's reserve joins the back of the level when its slice refills, so it stands ahead of nothing. The reserve is shown only when the output is not anonymised.

`--anonymise` names every other gateway `gateway A`, `gateway B`, … in order of first appearance (the inspected order's own gateway keeps its name) and leaves iceberg reserve out, so the output can be quoted to a participant without naming their counterparties.

Two warnings can appear in the timeline. One says the re-run and the live engine differ on the symbol from that point, so the positions after it are the re-run's. The other says the live engine's own `engine.trace` queued the order at a different priority than the re-run did; a clean trail never shows either. Like `rerun`, `queue` needs a trail it can re-run (see above), and reads the whole trail per query. Exit codes: `0` answered, `1` the order is not in the trail, `2` bad arguments, `3` the trail cannot be re-run.

30.6.12 Feed it to something else

```
pm-audit-replay stream --last 1h --format ndjson | jq -c 'select(.type=="beat")'
pm-audit-replay anomalies --severity error --format json > findings.json
pm-audit-replay story --order 4f2c9a --format markdown >> incident.md
```

30.6.13 Read a log with no index at all

```
pm-audit-replay stream --no-index --log-file /tmp/captured.log
```

Useful for a one-off trail that would only leave a database behind. `story` is the one subcommand this cannot serve.

30.7 Outputs

30.7.1 Detail levels

Detail is one axis, and raising it only ever reveals — nothing shown at a lower level disappears at a higher one.

Level	Flag	Adds
0	`-q`	Episode outcomes only — one line per episode
1	*(default)*	Business lifecycle: submissions, acks, fills, trades, cancels, halts, session changes, commands

Level	Flag	Adds
2	`-v`	Counterparty detail, derived facts (VWAP, price improvement, timings), rejection reasoning, link confidence
3	`-vv`	Market-data context (`book`, `depth`, `index.update`), drop copy, clock skew, `arrival_seq`
4	`-vvv`	Every unclassified event, and the raw payload beneath each narrated line

Level 0 is a different shape rather than a quieter level 1:

```
09:31:02.120  order 4f2c9a1e6d… is still working at the end of the window — 150 of 200 filled
09:31:02.122  trade 000042-000… — AAPL 150 @ 74.80
09:31:02.122  order 9ab1c47f2e… filled — 150 of 150 filled
```

30.7.2 `--format text`

The default. One sentence per narrated fact, with any annotations the switches added indented beneath it.

30.7.3 `--format ndjson`

One JSON object per line — the reconstructed model, not the raw log. Four object types:

Type	One per	Carries
`episode`	narrated episode	kind, anchor, actor, symbol, correlation id, outcome, summary, derived facts
`beat`	narrated fact	the envelope, the `text` of the prose line, `source` file and line, the whole payload in `fields`, resolved display prices in `prices`
`link`	edge between two episodes	`from`, `to`, relation, confidence, evidence
`anomaly`	finding	code, severity, episode, detail, `source`

One beat, pretty-printed (it is one line on the wire, and `fields` is shown abridged):

```
{
  "type": "beat",
  "episode": 4,
  "seq": 2,
  "sort_key": "000000000000\u001f0\u001f0000000000000000019\u001f2026-09-08T09:31:02.122000+00:00\u001f000000000006",
  "msg_id": "01M205PK3A0000000000000019",
  "causation_id": "01M205PK330000000000000001",
  "correlation_id": "01M205PK330000000000000001",
  "topic_seq": 1,
  "eseq": 19,
  "publisher_run": 0,
  "receipt_ts": "2026-09-08T09:31:02.122+00:00",
  "topic": "order.fill.TRADER01",
  "kind": "order.fill",
  "role": "progress",
  "late": false,
  "text": "4f2c9a1e6d… took 150 @ 74.80 from TRADER02's sell 9ab1c47f2e…",
  "source": {"file": "audit.log", "line": 7},
  "fields": {"order_id": "4f2c9a1e6d8b47c3a5f09e21b7d4c6a8", "fill_qty": 150,
             "fill_price": 74.8, "remaining_qty": 50, "liquidity_flag": "TAKER",
             "trade_ids": ["000042-000001873"]},
  "prices": {"fill_price": 74.8, "price": 75.69}
}
```


`fields` is the recorded payload **whole**, so a consumer needing something the prose does not mention never has to go back to the log for it. `prices` is what the unit resolution worked out, in display money, handed over rather than left to be redone.

At detail levels 1–4, each narrated fact is represented by a `beat`, and its `text` is the corresponding prose sentence. At level 0 (`-q`), output is episode summaries instead: NDJSON emits `episode` objects and no beats. Findings are separate `anomaly` objects, and link edges are separate `link` objects. At levels 1–4, text and beat ordering agree, while episode announcements and trailing links or anomalies are additional NDJSON records.

The `sort_key` is a packed internal ordering key, not a timestamp. Its fields encode `publisher_run`, sequence mode (`0` for `eseq`, `1` for the legacy fallback), sequence or fallback timestamp, `receipt_ts`, and stream ordinal. Prefer the explicit `eseq`, `publisher_run`, and `receipt_ts` fields when interpreting or displaying a beat.

Episodes are announced before their first beat, so the stream can be processed without buffering. Links come last, deduplicated and sorted: an edge is a fact about two episodes, not about the beat that happened to state it.

30.7.4 `--format json`

The same objects in one document, with a header — the window, the source files, the rules version and per-type counts — for a consumer that prefers one read to a stream.

30.7.5 `--format markdown`

The prose under a heading per episode, for pasting into a bug report or an incident write-up. Grouped rather than chronological, because a write-up is read episode by episode.

30.7.6 `--format csv`

`episodes` only, for export.

30.7.7 Which formats each subcommand produces

Subcommand	text	ndjson	json	markdown	csv
`stream`					—
`story`					—
`anomalies`				—	—

Subcommand	text	ndjson	json	markdown	csv
`digest`		—	—	—	—
`episodes`		—	—	—	
`stats`		—	—	—	—

A combination outside that table is **refused** with exit 2 rather than quietly ignored. A tool that accepts `--format json` and prints a table has told the caller something untrue, and a script finds out only by failing to parse it.

30.8 Tips & tricks

Most misses are a window that starts too late

`no order matching '4f2c9a' in this window` almost always means the submission is behind `--from`, not that the order is absent. The tool says so and suggests widening the window. The same cause explains most `CAUSE_NOT_FOUND` findings: the effect is in the window and its cause is not.

Start with `anomalies`, not with `stream`. Every finding prints the `story` command that shows it, so the fastest route from "something is wrong" to the evidence is `anomalies --severity warn`, then paste the suggestion.

`--show-source` turns prose back into lines. Every narrated sentence gets `audit.log:118423` appended, so anything the narrator says can be checked against the bytes it came from. Pair it with `--id-len full` when you intend to grep for the ids.

`--explain` is how you tell reading from guessing. It prints each link's confidence and evidence. If a narrative surprises you, check whether the tool recorded the link or inferred it before believing either.

`--strict-causality` when you need certainty. `story --strict-causality` follows only `RECORDED` links, so the result contains nothing the tool worked out for itself.

`-vvv` is the round-trip check. At level 4 nothing is withheld: every event in the window is narrated. If you suspect the tool is hiding something, that is the level that proves otherwise.

Prices are never guessed. A tick-scaled price whose `tick_decimals` the window does not contain is printed *in ticks and labelled as ticks*, with a `TICK_SCALE_UNKNOWN` finding — never silently rescaled. `--show-units` shows where each scale came from.

`--tz` is for reading, never for filtering. `--from`/`--to`/`--last` are UTC, because that is what the trail records. `--tz` moves only what is *shown*, and only in the human-readable formats: `ndjson` and `json` keep the recorded timestamps, because a display switch that moved them would make the two formats disagree about when something happened.

Give a scratch trail its own directory

`--log-file` pulls in the day parts beside it, and the index lands next to the log by default. For a one-off capture, point `EDUMATCHER_DATA_DIR` at a scratch directory, or pass `--log-file` and `--db` explicitly, so the findings are about the run you meant.

Page a long narrative with `--force-color | less -R`. Piping switches colour off, because a pipe is not a terminal — and a pager is the one pipe that is. Without the `-R`, `less` prints the escape codes instead of acting on them.

An index is a cache, and is allowed to be deleted. It is derived entirely from the log. If anything looks stale, `--rebuild` or `rm` it; there is no state in it that the trail does not have.

`pm-help pm-audit-replay` lists the command alongside every other EduMatcher tool.

30.9 See Also

In this book:

- Audit Trail — `pm-audit` and `pm-audit-cli`, the trail this tool reads
- Persistence — which files each process writes, and where

Reference Manual:

- Processes — pm-audit — where `pm-audit` sits among the runtime processes
- Known Limitations & Bugs — what is known not to work yet

Protocols and Clients:

- Message Reference — the wire contract every finding is stated against

CHAPTER 31

Answering a Dispute

Learning objectives

After reading this page you will be able to:

- Say which command answers which dispute: “you never got my order”, “why was I rejected”, “why did I not trade”, “why that price”, “was the record changed”
- Show that the trail you are quoting is complete, unaltered, and agrees with what the engine does
- Anchor the trail’s evidence outside the host that wrote it
- State plainly what the trail cannot show

The audit trail is written to be read by someone who was not there. This chapter is the order to read it in when a participant, a clearing counterparty or a reviewer asks the exchange to prove what happened. The tools are described in Audit Trail and Audit Replay; this page says which one to reach for.

31.1 Which command answers which question

The question	Reach for	What it shows
“I sent an order and you ignored it”	`pm-audit-replay story --order ID`, then `stream --gateway GW` around the time	The order as received (the engine echoes every command before acting on it), its acknowledgement with the engine’s clock (`ts_ns`), or the reject and its reason. If the gateway itself refused the request (login, role, rate limit) it is a `system.gateway_reject` record with the peer address
“Why was I rejected?”	`story --order ID -v --explain`	The `reject_code`, and the market state behind it: the halt, session phase, collar or limit, each tied to the message that set it
“Why did my order not trade?”	`pm-audit-replay queue --order ID`	Where it stood in its level after every change, what traded or was cancelled ahead of it, and whether the market ever reached its price (best opposite price against its limit)
“What happened at that price?”	`queue --symbol SYM --price P`	Who rested, traded and left at one level, with what it held after each change

The question	Reach for	What it shows
"Why did I get that fill and that price?"	`story --order ID`, `story --trade ID`	Both sides of each trade, aggressor and resting side, maker or taker, the VWAP of a sweep and the trade ids behind it
"Is the trail complete?"	`pm-audit-replay anomalies --severity warn`, `pm-audit-replay stats`	`SEQ_GAP` and `ESEQ_GAP` prove a missing message; `RECORDER_RESTART` says the recorder was down; low envelope coverage says a publisher bypassed the causal wrapper
"Has the record been altered?"	`pm-audit-cli verify`	Every archived day against the hash chain in `manifest.jsonl`; it names the day that differs
"Does the engine agree with the record?"	`pm-audit-replay rerun`	A fresh engine fed the trail's commands publishes what the live engine did, message for message
"What was the exchange running?"	`story` on the `system.startup` episode	Engine version, source commit, the full compiled configuration and its SHA-256, the state files it started from

31.2 The procedure

Do these in order and keep the output; each step makes the next one worth quoting.

1. **Prove the record is unaltered.** `pm-audit-cli verify` must exit `0`. Compare the *head hash* it prints with the one you anchored off-host (see below). A match means no archived day, and no manifest entry before the anchor, has changed since it was anchored.
2. **Prove it is complete.** `pm-audit-replay anomalies --severity warn` over the day. A clean result means every per-topic and engine-wide sequence runs without a hole, every fill has its trade, and the quantities conserve. Any `SEQ_GAP`, `ESEQ_GAP` or `RECORDER_RESTART` in the window you are about to quote must be explained first.
3. **Show the engine agrees.** `pm-audit-replay rerun` over the trail. Zero hidden inputs and zero differing episodes mean the trail is not only self-consistent but is what the matching engine does with those commands. Run it with the engine version the day ran on (the output names both versions).
4. **Answer the question** with `story`, `queue` or `stream` from the table above.
5. **Quote with `--anonymise`** when the answer goes to a participant, so other gateways appear as `gateway A`, `gateway B` and iceberg reserve is left out. The inspected order's own gateway keeps its name.

Every sentence the narrator prints traces to a field in a recorded line, and `--show-source` prints the file and line number so a reader can go and look.

31.3 Anchoring the head hash

`pm-audit-cli verify` ends with the hash of the newest manifest entry. Each entry contains the hash of the one before it, so that single value commits to every archived day to date. Anyone who can rewrite the files on the recorder host can rewrite the whole chain, so the evidence is only as good as the place that value is kept.

After each day closes (the archive is made shortly after midnight in the session timezone), copy the head hash somewhere the recorder host cannot write:

```
pm-audit-cli verify | tail -1      # OK: 12 day(s) in the manifest, 0 problem(s); head 3f9a…
```

Send it by mail to a mailbox the operator does not control, commit it to a separate repository, or write it to an append-only system. Keep the date with it. When a day is disputed, run `verify` and compare. A chain that verifies but whose head differs from the anchored one has been cut back or rewritten after the anchor.

Retention (`--keep-days`) removes old archives but leaves a `retired` line in the manifest with the day's last hash, so the chain still verifies and the day's digest is still on record; the content is not.

31.4 What the trail cannot show

Say these before someone else does.

- **Messages the recorder did not receive.** While `pm-audit` is stopped nothing is recorded. The gap is proven by `ESEQ_GAP` and the restart by `RECORDER_RESTART`, but the content is not recoverable. Past a 200 000-message receive backlog the subscriber drops; `audit.missed_messages` counts what it can prove it missed. See What can be lost.
- **A hard stop of the recorder or the host.** Up to one write buffer (`--buffer-size` messages or `--flush-interval` seconds) is lost to `kill -9`; up to `--fsync-interval` seconds more to a power loss.
- **Refusals the gateways do not report.** Login, role and rate-limit refusals are recorded. Malformed frames, field-validation refusals and a read-only REST key attempting a write are counted by the gateway and are not in the trail. A burst of repeated refusals is one record per second per peer host with a `count`; the count of the final burst is lost if the peer then goes quiet.
- **Commands that bypassed the envelope.** A client that does not use the project's client library cannot be echoed. `engine.diagnostics{component="UNRECORDED_COMMAND"}` counts them; a non-zero value means the trail is not a complete record of what the exchange was asked to do.
- **Inputs other than commands and two timers.** The re-run reproduces the circuit breaker's reopen and the disconnect of a gateway that stopped sending heartbeats. Anything else the engine did with no command behind it appears as a *hidden input* and cannot be reproduced.
- **A trail that cannot be re-run.** `rerun` and `queue` start from the trail's first line, which must be an engine start that began with an empty book, and every run must carry its compiled configuration. Restarts in between are re-run. Once retention has removed the day the data directory began empty, the trail can still be read with `story` and `stream`, but not re-executed. An engine that ran from a YAML file rather than a deployed artifact has no configuration in its trail (`CONFIG_NOT_RECORDED`). A trail with a proven hole (`ESEQ_GAP`, or a recorder restart while the engine ran) is not re-run either: its verdict would blame the engine for what the recorder missed.
- **Time.** Every line carries the recorder's receipt time. Engine clock readings (`ts_ns`) are on trades, fills, acknowledgements and book snapshots, and ordering between messages is by the engine's own sequence, not by either clock. A client's own timestamp on an order is the client's, and the engine does not use it.
- **Tamper-proof storage.** The manifest makes an edit evident; it does not prevent one. Without an anchored head hash the chain proves only internal consistency.

31.5 See Also

- Audit Trail: the file format, day parts and archives, the evidence manifest, what can be lost
- Audit Replay: every subcommand, the anomaly codes, re-executing the trail, the queue view

CHAPTER 32

Centralized Log Server

Learning objectives

After reading this page you will understand:

- Why a centralized log server matters and what `pm-log-srv` records
- How to start `pm-log-srv` and where `log.db` lives
- The LALF wire protocol at a glance and how auto-detection works
- How LALF-PS distributes logs over ZeroMQ to live viewers, and why a blind `PUB` socket forces the lease-based liveness design it uses
- All `pm-log-cli` subcommands, options, and output formats
- A practical cookbook of query workflows grouped by purpose
- How to read a `diagnose` report and act on its recommendations

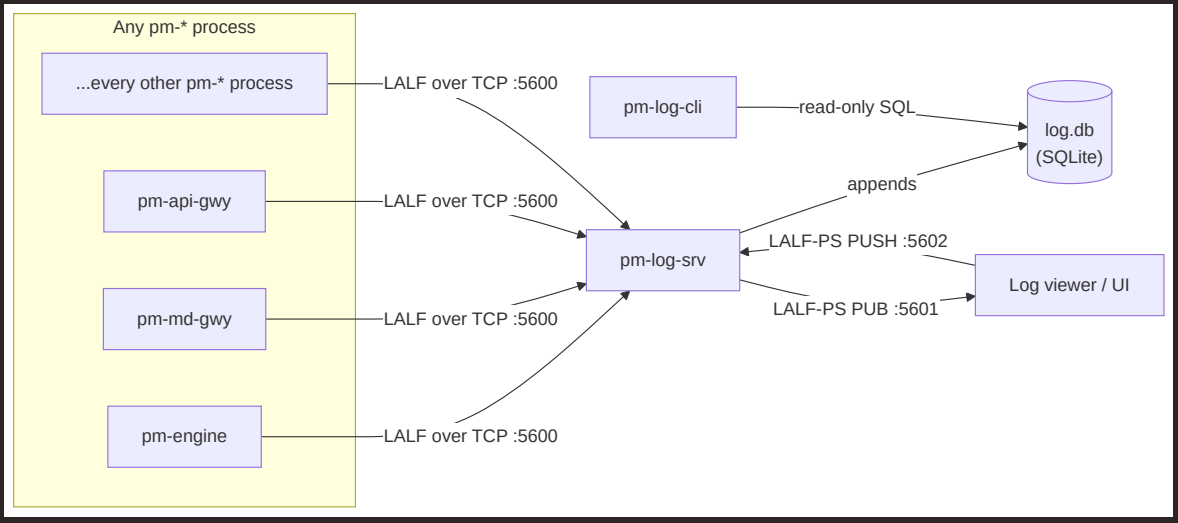
32.1 Overview

Every EduMatcher process — the engine, gateways, `pm-stats`, `pm-audit`, bots, viewers, admin tools — already logs through Python's standard `logging` module. Until now, each process's log output went only to its own terminal or its own redirected file: there was no single place to ask "what did every process log in the last five minutes," and correlating an issue spanning two processes meant manually lining up timestamps across separate terminal scrollbacks.

`pm-log-srv` solves this the same way `pm-audit` already solved it for trading events and `pm-stats` solved it for market statistics: a dedicated collector process, a purpose-built SQLite schema, and a read-only query CLI.

Component	Role	Type
`pm-log-srv`	Collector — accepts logging from every `pm-*` process over TCP and appends it to a queryable SQLite database	Long-running process
`pm-log-cli`	Query/troubleshooting tool — reads `log.db` and prints structured output, including a rule-based `diagnose` report	One-shot CLI
LALF-PS	Distribution interface — the ZeroMQ `PUB`/`PULL` pair `pm-log-srv` binds so live viewers can be pushed rows as they land instead of polling `log.db`	Interface, not a process

The collector and the query tool are completely independent. `pm-log-cli` reads `log.db` directly from disk and works even when `pm-log-srv` is not currently running — the same posture `pm-audit-cli`/`pm-stats-cli` already take relative to their own recorder processes.



There are therefore two distinct ways to get logging *out* of `pm-log-srv`, and they suit different jobs:

	`pm-log-cli`	LALF-PS
Transport	Direct read-only SQL against `log.db`	ZeroMQ `PUB/PULL`
Works when the server is stopped	Yes	No
Latency	Whatever your polling interval is	Pushed on commit
Best for	Ad-hoc investigation, scripting, export	Live viewers, dashboards, anything long-lived and interactive

`pm-log-srv` is not a replacement for `pm-audit` or `pm-stats`. Trading events (orders, fills) and market statistics have their own purpose-built collectors; `pm-log-srv` is for operational `logging`-module output only — think "what would otherwise have gone to a terminal or a log file."

Phased rollout

As of this revision, `pm-log-srv` and `pm-log-cli` are fully implemented and usable standalone — you can start the server and point a test client at it today. Automatically wiring every existing `pm-*` process's own logging into a `TcpLogHandler` (so it auto-detects and uses a running `pm-log-srv` with zero configuration) is a follow-up phase of this feature and has not yet been rolled out to every entrypoint. Until then, use `pm-log-srv`/`pm-log-cli` to inspect logging emitted by any client that already speaks LALF (see the wire protocol reference below if you are writing one).

32.2 Data Directory

`log.db` is stored in the same data directory used by all other EduMatcher persistent files:

Running mode	Default location
Source checkout (`poetry run pm-log-srv`)	`<repo>/src/data/log.db`
Installed (`pm-log-srv` on PATH)	`~/.local/share/edumatcher/log.db`

Override with `EDUMATCHER_DATA_DIR` or `--db`:

```
export EDUMATCHER_DATA_DIR="$HOME/sessions/morning"
pm-log-srv  # writes to $HOME/sessions/morning/log.db
```

32.3 pm-log-srv — Log Collector

`pm-log-srv` binds a TCP listen socket, accepts any number of concurrent LALF connections, and appends every accepted `LOG` record to `log.db`. It never talks to the engine's ZeroMQ bus, to the engine, or to any gateway — its own ZeroMQ sockets

(LALF-PS, below) carry log distribution only and are entirely separate from the trading bus on `:5555/:5556`.

```
pm-log-srv [options]
```

32.3.1 Startup options

Flag	Default	Description
`--host ADDR`	from config / `0.0.0.0`	TCP bind address
`--port PORT`	from config / `5600`	TCP listen port for LALF clients
`--db PATH`	from config / `data/log.db`	SQLite database path
`--retention-days N`	from config / `30`	Prune `log_events` rows older than N days, once per hour. `0` disables pruning (unbounded retention)
`--max-message-bytes N`	from config / `65536`	Maximum `LOG` payload size before truncation — oversized messages are truncated and stored, never dropped
`--pub-port PORT`	from config / `5601`	LALF-PS ZeroMQ `PUB` bind port (log distribution out)
`--pull-port PORT`	from config / `5602`	LALF-PS ZeroMQ `PULL` bind port (subscriber control in)
`--lease-sec N`	from config / `30`	Subscription lease TTL; a subscriber that stops renewing is reaped after this long
`--no-pubsub`	off	Disable LALF-PS entirely — bind no ZeroMQ sockets and run as a pure TCP collector
`--config PATH` / `-c`	`engine_config.yaml`	Engine config YAML path, read for the optional `log_server:` block
`--log-level LEVEL`	`WARNING`	Explicit level: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`
`-v` / `--verbose`	off	Increase verbosity (`-v` → `INFO`, `-vv` → `DEBUG`)
`-q` / `--quiet`	off	Reduce log output to warnings/errors

CLI flags override the corresponding field in `engine_config.yaml`'s `log_server:` block for that invocation only. See Configuration — Configuring pm-log-srv for the full config field reference.

```
# Start with defaults — binds 0.0.0.0:5600, writes to data/log.db
pm-log-srv

# Loopback-only, custom port, verbose startup logging
pm-log-srv --host 127.0.0.1 --port 5600 -v

# Unbounded retention (never prune)
pm-log-srv --retention-days 0

# Point at a specific database for a scratch/test session
pm-log-srv --db /tmp/session.db

# Collector only — no ZeroMQ sockets bound at all
pm-log-srv --no-pubsub

# Move LALF-PS off the default ports (e.g. two servers on one host)
pm-log-srv --port 5700 --pub-port 5701 --pull-port 5702
```

Expected startup output:

```
2026-07-29 09:30:00,180 INFO edumatcher.log_srv.main - starting pm-log-srv with log level INFO
2026-07-29 09:30:00,206 INFO edumatcher.log_srv.server - pm-log-srv 'log-srv01' listening on 0.0.0.0:5600 db=data/log.db retention_da
2026-07-29 09:30:00,211 INFO edumatcher.log_srv.pubsub - LALF-PS interface up: PUB=tcp://0.0.0.0:5601 PULL=tcp://0.0.0.0:5602 lease=3
```

`pm-log-srv` occupies a contiguous three-port block: `5600` for LALF/TCP collection, `5601` for the LALF-PS `PUB`, `5602` for the LALF-PS `PULL`. All three must differ; the server refuses to start otherwise.

`pm-log-srv` is the one `pm-*` process that always logs to stdout/file only — it never sends its own operational logging to another `pm-log-srv` instance (there is nothing listening for its own bootstrap messages before it has finished starting).

32.3.2 Retention

`log_events` defaults to a bounded 30-day retention rather than growing unbounded like `pm-audit`'s/`pm-stats`' own deliberately-forever stores — operational logging is disposable in a way trading history and market statistics are not. Pruning runs automatically once per hour in the background; `pm-log-cli prune` (below) is also available for pruning on demand.

32.4 LALF — The Wire Protocol at a Glance

Every `pm-*` process (or, in principle, any TCP client in any language) speaks LALF ("Logging ALF") to `pm-log-srv`: a long-lived TCP connection, one `HELLO`/`WELCOME` handshake, and then a stream of `LOG` messages until the connection ends.

```
C: HELLO|CLIENT=pm-md-gwy|PID=51002|HOST=trader-laptop|PROTO=LALF1
S: WELCOME|PROTO=LALF1|SRV=log-srv01|HBINT=5|SESSION=7f3a9c21
C: LOG|SEQ=1|TS=2026-07-28T09:30:00.010Z|LEVEL=INFO|LOGGER=edumatcher.md_gateway.gateway|LEN=29
   md-gwy01 listening on :5570
C: HB|TS=2026-07-28T09:30:05.500Z
C: EXIT
```

For the full normative wire specification — every message type, field table, error code, and conformance rule — see the LALF Protocol Reference.

32.5 LALF-PS — The ZeroMQ Log Distribution Interface

LALF brings logging *in*. **LALF-PS** ("LALF Pub/Sub") sends it back *out*, over ZeroMQ, to anything that wants to watch the system live: a log viewer, a filter/search UI, a dashboard, an alerting shim.

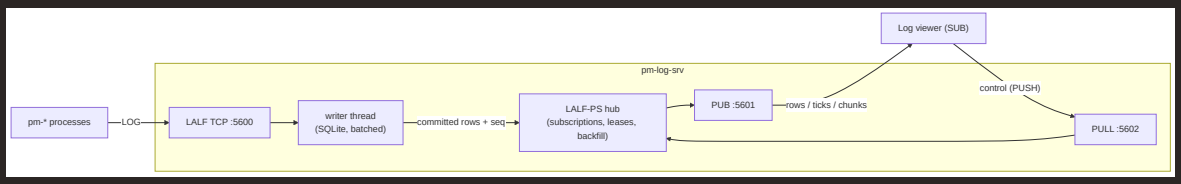
The alternative — having each viewer poll `log.db` on a timer — is what LALF-PS exists to avoid. Polling forces an unpleasant trade: poll often and you burn CPU re-running the same query against a growing table for nothing most of the time; poll rarely and your viewer lags reality by however long the interval is. Neither is a good foundation for a UI whose whole job is to show you what is happening *now*. LALF-PS pushes instead, the moment a row is committed.

32.5.1 Socket topology

`pm-log-srv` binds two ZeroMQ sockets. The shape is deliberately identical to `pm-index`'s own `PUB`/`PULL` pair, so a client written against one needs no new socket vocabulary for the other.

Socket	Bound by	Default address	Carries
`PUB`	`pm-log-srv`	`tcp://…:5601`	Everything outbound: live rows, notification ticks, backfill chunks, control acks, errors, server state
`PULL`	`pm-log-srv`	`tcp://…:5602`	Everything inbound: subscribe, renew, unsubscribe, backfill, status

A subscriber holds the mirror image: a `SUB` connected to `:5601` and a `PUSH` connected to `:5602`.



Every message is the same two-frame envelope used everywhere else on the bus — frame 0 is the topic string, frame 1 is a JSON payload — so ZeroMQ's prefix filter does the routing in the kernel before your process ever sees a byte.

32.5.2 `sub_id` — the routing key

Every control message carries a subscriber-chosen `sub_id`. It plays exactly the role `gateway_id` plays elsewhere on the bus: the server appends it to each reply topic, so a subscriber needs only two `SUBSCRIBE` prefixes to receive everything relevant to it:

```
sub.setsockopt(zmq.SUBSCRIBE, b"log.")              # simplest: everything
# or, more precisely:
sub.setsockopt(zmq.SUBSCRIBE, f"log.event.{sub_id}".encode())
sub.setsockopt(zmq.SUBSCRIBE, f"log.backfill.{sub_id}".encode())
sub.setsockopt(zmq.SUBSCRIBE, b"log.server_state")
```

Pick something stable and unique — `logview-<pid>`, `dashboard-01`. Two live subscribers sharing a `sub_id` will each receive the other's traffic and fight over one lease, because to the server they *are* one subscription.

32.5.3 Two modes: `STREAM` and `NOTIFY`

A subscription declares up front what it wants delivered, because the two plausible kinds of log UI want opposite things.

`STREAM` — full rows are pushed as they are committed, on `log.event.{sub_id}`. This is what a live tail wants: the viewer never touches `log.db` at all and can run on a different host from the database. The cost is bus traffic proportional to the log volume that passes your filter, which is exactly why the filter matters.

`NOTIFY` — the server publishes only a small tick on `log.notify.{sub_id}` saying "n matching rows arrived, the highest is seq X", with a per-level breakdown and no row bodies whatsoever. This suits a UI that already reads `log.db` itself and just needs to know when to refresh, or one that only lights up an indicator on new errors. Ticks are coalesced over `notify_interval_ms` (default 250 ms), so a burst of a thousand rows produces one message rather than a thousand.

You can change your mind at any time by re-sending `log.subscribe` with a different `mode` — see idempotency, below.

32.5.4 Filtering

Both modes take the same optional `filter` object, and the server applies it identically to live rows and to backfill rows. That last point matters more than it sounds: it is what guarantees a viewer sees no gap and no duplication at the seam where its historical window meets the live stream.

Field	Type	Description
`min_level`	string	`DEBUG` / `INFO` / `WARNING` / `ERROR` / `CRITICAL` — rows below are excluded
`processes`	array	Exact process-name match, e.g. `["pm-engine", "pm-md-gwy"]`
`loggers`	array	**Prefix** match, so `["edumatcher.engine"]` also matches `edumatcher.engine.book`
`sessions`	array	Exact LALF session-id match, to follow one connection of one process
`contains`	string	Case-insensitive substring on the message body
`exceptions_only`	boolean	Only rows carrying a traceback

All present fields are ANDed. An omitted field constrains nothing, and an absent `filter` matches everything. A malformed filter is rejected with `INVALID_FILTER` rather than silently degrading to "match everything" — a filter that quietly stops filtering is how a viewer ends up flooded.

Filter server-side rather than client-side wherever you can. A `min_level: "WARNING"` filter on a busy system can be the difference between a few messages a minute and a few thousand.

32.5.5 Backfill — "the last n minutes"

A log viewer is nearly useless if it starts empty and only shows you what happens from now on; the interesting thing usually already happened. Two ways to prime it:

- Pass `backfill_minutes` on `log.subscribe`, and the server starts the replay immediately after acknowledging the subscription.
- Send `log.backfill_request` at any time afterwards — to widen the window, or to re-fetch with a different filter without disturbing the live subscription.

Either way the response arrives as a sequence of `log.backfill.{sub_id}` messages sharing one `request_id`, in ascending `seq` order, with the last one setting `done: true`. **Keep reading until you see `done`.** An empty window still produces exactly one chunk with `rows: []` and `done: true`, so you never wait forever on a quiet system.

Chunking is not a nicety. A busy hour can be hundreds of thousands of rows; sending that as one response would mean both a multi-megabyte ZeroMQ frame and a SQLite scan long enough to stall the server's main loop — which would stall LALF collection, which would back-pressure every `pm-*` process in the system. The server instead emits at most one bounded chunk per loop iteration, so an arbitrarily large window costs a bounded amount of work per pass.

Two limits apply: `max_backfill_minutes` (default 1440, i.e. 24 h) caps the window and rejects anything larger with `INVALID_WINDOW`; `max_backfill_rows` (default 100 000) caps the volume, and when it bites the final chunk sets `truncated: true` so you know history was cut short rather than genuinely exhausted.

32.5.6 Liveness — why leases

This is the part of the design that is not obvious, so it is worth being explicit about the problem.

A ZeroMQ `PUB` socket is **blind**. It never learns who is attached, subscribers come and go without the publisher being told, and publishing into the void succeeds silently. There is no publish-side event that means "my subscriber died" — nothing analogous to the TCP `recv()` returning zero bytes that tells the LALF side a producer has gone. A naive implementation would happily keep filtering rows, formatting messages and accumulating per-subscriber state for a viewer that was killed an hour ago.

LALF-PS therefore makes every subscription an explicit **lease**:

1. `log.subscribe` creates the subscription with a TTL. The server *clamps* your requested `lease_sec` to its `max_lease_sec` rather than rejecting it, and tells you what you actually got in the ack — along with `renew_before_sec`, which is always half the granted lease.
2. The subscriber sends `log.renew` every `renew_before_sec`. This is the *only* signal that keeps the subscription alive. (A `log.backfill_request` also counts as proof of life, so a subscriber pulling a large window cannot expire mid-transfer.)
3. If no renewal arrives before the deadline, the server reaps the subscription: filter state gone, buffered rows discarded, any in-flight backfill job cancelled.
4. A final `log.lease_expired.{sub_id}` is published on the way out. This is for the case where the subscriber is not dead but merely wedged — it tells such a client unambiguously that it must re-subscribe, instead of sitting forever waiting for rows that will never come.

A crashed viewer therefore costs the server at most one lease period of buffering, and the mechanism is transport-independent: it would work identically over any fan-out that lacks peer visibility.

Choosing a lease

Shorter leases reap faster but cost more control traffic; the default 30 s means a dead viewer is forgotten within 30 s at a cost of one small `PUSH` every 15 s. Raise it for a viewer on a flaky link, lower it if you run many short-lived subscribers.

32.5.7 The other failure: a subscriber that is slow, not dead

Leases handle death. They do not handle a subscriber that is alive and renewing but simply cannot read as fast as the system logs — and left alone, that subscriber would grow the server's memory without bound.

Two bounds apply. Each subscription's row buffer is capped at `max_pending_rows` (default 20 000): past that, the **oldest** buffered rows are shed and a running `dropped` counter is reported in every subsequent `log.event`. Separately, the `PUB` socket carries a bounded `pub_sndhwm` (default 10 000) so ZeroMQ's own queue for a wedged-but- connected peer cannot grow indefinitely either.

`dropped` climbing is the signal that a viewer is not keeping up and its view has gaps. The cures, in order of preference: tighten the filter, switch to `NOTIFY`, or re-backfill to close the gap.

32.5.8 Server liveness

`log.server_state` is published on the un-suffixed topic — no `sub_id` — every `heartbeat_interval_sec`, plus once with `state: "DOWN"` at shutdown. Any client can subscribe to it without having subscribed to anything else, which makes it the natural way for a UI to show "log server up/down" and to read the server's counters before deciding what to ask for.

Absence of `log.server_state` for more than a couple of intervals is how a subscriber distinguishes "the *server* went away" from "*my subscription* went away" — two situations that need different responses.

32.5.9 Ordering, durability and the slow-joiner race

Three guarantees worth stating plainly:

Rows are published after commit, never before. The server publishes only once the SQLite transaction that assigned each row's `seq` has committed. A subscriber can therefore never observe a row that a subsequent crash would have rolled back: the live stream and `log.db` can never disagree.

`seq` is the ordering key. It is monotonic and server-assigned; rows arrive in ascending `seq` within and across messages, in both live and backfill paths. Use it, not timestamps — `client_ts` comes from the producing process's own clock and two processes' clocks need not agree.

The first ack can be lost. This is ZeroMQ's classic *slow joiner* behaviour: `connect()` completes asynchronously, so a message published in the moments after your `SUB` socket connects may be dropped before delivery. LALF-PS handles this by making `log.subscribe` **idempotent** — re-sending it for an existing `sub_id` replaces the mode and filter, renews the lease, and re-emits the ack, while preserving the lifetime counters. If you have not seen your ack within a second, simply send `log.subscribe` again.

32.5.10 Error replies

Every rejected request is answered on `log.error.{sub_id}` with a machine-readable `code`:

Code	Meaning	What to do
`BAD_REQUEST`	Unsupported topic or malformed payload	Fix the request; do not retry as-is
`UNKNOWN_SUB`	No such subscription — never created, or already reaped	Send `log.subscribe`
`TOO_MANY_SUBS`	Server is at `max_subscribers`	Back off and retry, or raise the limit
`INVALID_FILTER`	Malformed `filter` object	Fix the filter
`INVALID_MODE`	`mode` was neither `STREAM` nor `NOTIFY`	Fix the mode
`INVALID_WINDOW`	`minutes` absent, non-positive, or above `max_backfill_minutes`	Request a smaller window
`INTERNAL`	Server-side failure, e.g. a backfill query error	Retry; check the server's stdout logging

A control message arriving with no usable `sub_id` is dropped and logged server-side — there is no reply topic to answer on.

32.5.11 A minimal subscriber

The whole interface in one file. This connects, backfills the last five minutes of warnings and above, then tails live while keeping its lease alive.

```
import time
import zmq
from edumatcher.config import LOG_SRV_PUB_ADDR, LOG_SRV_PULL_ADDR
from edumatcher.models.message import (
    decode,
    make_log_renew_msg,
    make_log_subscribe_msg,
    make_log_unsubscribe_msg,
)

SUB_ID = "logview-demo"
ctx = zmq.Context.instance()

sub = ctx.socket(zmq.SUB)
sub.connect(LOG_SRV_PUB_ADDR)
sub.setsockopt(zmq.SUBSCRIBE, b"log.")

push = ctx.socket(zmq.PUSH)
push.connect(LOG_SRV_PULL_ADDR)
time.sleep(0.2)  # let both connections settle before the first publish

push.send_multipart(
    make_log_subscribe_msg(
        SUB_ID,
        mode="STREAM",
        log_filter={"min_level": "WARNING"},
        backfill_minutes=5,
    )
)

renew_every = 15.0          # updated from the ack's renew_before_sec
next_renew = time.monotonic() + renew_every
poller = zmq.Poller()
poller.register(sub, zmq.POLLIN)

try:
    while True:
        for _ in dict(poller.poll(timeout=250)):
            topic, payload = decode(sub.recv_multipart())

            if topic == f"log.subscribe_ack.{SUB_ID}":
                # Always trust the ack over what you asked for: the server
                # clamps the lease and normalises the filter.
                renew_every = payload["renew_before_sec"]
                next_renew = time.monotonic() + renew_every

            elif topic in (f"log.event.{SUB_ID}", f"log.backfill.{SUB_ID}"):
                for row in payload["rows"]:
                    print(f"{row['client_ts']} {row['level']:<8} "
                          f"{row['process']:<12} {row['message']}")
                if payload.get("dropped"):
                    print(f"!! {payload['dropped']} rows dropped — not keeping up")

            elif topic == f"log.lease_expired.{SUB_ID}":
                # We were reaped. Re-subscribing is always safe.
                push.send_multipart(
                    make_log_subscribe_msg(SUB_ID, "STREAM",
                                           {"min_level": "WARNING"})
                )

            elif topic == "log.server_state" and payload["state"] == "DOWN":
                print("pm-log-srv is shutting down")

        if time.monotonic() >= next_renew:
            push.send_multipart(make_log_renew_msg(SUB_ID))
            next_renew = time.monotonic() + renew_every
```

For a `NOTIFY`-mode client, replace the `mode` and handle `log.notify.{SUB_ID}` instead — the payload gives you `count`, a `levels` breakdown and `last_seq`, and you read the actual rows from `log.db` yourself with `WHERE seq > <your watermark>`.

32.5.12 Tuning

All of these live in the `log_server:` block of `engine_config.yaml`; see the config spec for the authoritative table.

Setting	Default	Raise it when…	Lower it when…
`lease_sec`	`30`	Subscribers sit on a flaky or high-latency link	You want dead viewers reaped faster
`max_subscribers`	`32`	You genuinely run many viewers	You want a hard guard against runaway clients
`notify_interval_ms`	`250`	Notify traffic is itself becoming noise	A UI needs snappier refreshes
`backfill_chunk_rows`	`500`	Backfills feel slow and the network is fast	Individual messages are too large for your consumer
`max_backfill_minutes`	`1440`	Users legitimately want multi-day windows	You want to protect the server from expensive scans
`max_pending_rows`	`20000`	Subscribers are bursty but do catch up	Memory matters more than gap-free delivery

32.5.13 Troubleshooting

Symptom	Likely cause	Fix
No `log.subscribe_ack` ever arrives	Slow-joiner race, or wrong `SUBSCRIBE` prefix	Re-send `log.subscribe` (it is idempotent); check you subscribed to `log.` or the exact reply topics
Ack arrives, then nothing	Filter is stricter than you think, or the system really is quiet	Inspect the **normalised** `filter` in the ack; send `log.status_request`; try `min_level: "DEBUG"`
`UNKNOWN_SUB` on renew	Your lease already expired	Re-send `log.subscribe`; renew at `renew_before_sec`, not at `lease_sec`
`dropped` climbing in `log.event`	Subscriber is slower than the log volume	Tighten the filter, switch to `NOTIFY`, or re-backfill
Backfill seems to never end	You stopped reading before `done: true`	Keep consuming chunks until `done`; correlate by `request_id`
`truncated: true` on the last chunk	Window held more rows than `max_backfill_rows`	Narrow the window or the filter, or raise the limit
Nothing at all on `:5601`	LALF-PS disabled or on other ports	Check `--no-pubsub` / `pubsub_enabled`, and the `pub_port`/`pull_port` settings
Server refuses to start, port error	`port`, `pub_port` and `pull_port` are not all distinct	Give each a different port

For the complete normative field tables of every LALF-PS message, see Message Reference — LALF-PS messages.

32.6 pm-log-cli — Query/Troubleshooting Tool

`pm-log-cli` is a read-only command-line tool for querying and troubleshooting `log.db` — no LALF/TCP connection is made, so a busy, slow, or currently-stopped `pm-log-srv` never blocks you from inspecting data it already collected.

```
pm-log-cli [global-options] COMMAND [command-options]
```

32.6.1 Global options

Flag	Default	Description
`--db PATH`	`data/log.db`	SQLite database path
`--format human|json`	`human`	Output format

`--format` is accepted both before and after the subcommand — both `pm-log-cli --format json query` and `pm-log-cli query --format json` work, since a value given after the subcommand always wins if both are present.

Output formats:

Format	When to use
`human`	Interactive terminal — aligned columns, level-colored when the terminal supports it
`json`	Automation, scripts, downstream processing — one JSON object per row for `tail`/`query`/`processes`, a single JSON object for `stats`/`diagnose`

No-result behaviour:

- `human`: prints `No matching log events found.`
- `json`: prints nothing (an empty stream of lines) for row-based commands

32.6.2 Exit codes

Code	Meaning
`0`	Success
`1`	Database error — `log.db` not found or not readable (has `pm-log-srv` ever run?)
`2`	Argument error (invalid flag combination)
`3`	`diagnose` found at least one warning- or error-level finding; `[INFO]` observations alone still exit `0` (only meaningful for scripting; `query`/`tail`/`stats`/`processes`/`prune` never use this code)

32.6.3 `query` — Filtered historical search

The most flexible command — filter by process, level, logger, time range, free text, or exception presence.

```
pm-log-cli query [options]
```

Options:

Flag	Default	Description
`--process NAME`	(all)	Exact match against the process name
`--level LEVEL[,LEVEL...]`	(all)	One or more of `DEBUG,INFO,WARNING,ERROR,CRITICAL`
`--logger PATTERN`	(all)	SQL `LIKE` pattern against the logger name (e.g. `edumatcher.md_gateway.%`)
`--since ISO_TS`	(none)	Start of time range against `client_ts`

Flag	Default	Description
`--until ISO_TS`	(none)	End of time range against `client_ts`
`--grep TEXT`	(none)	Case-insensitive substring search against the message
`--has-exception`	off	Only rows with a formatted exception/traceback attached
`--limit N`	500	Maximum rows returned
`--reverse`	off	Oldest-first instead of newest-first

Output columns: `seq`, `client_ts`, `process`, `instance`, `level`, `logger`, `message`.

Examples:

```
# Every ERROR/CRITICAL from a specific process
pm-log-cli query --process pm-md-gwy --level ERROR,CRITICAL

# Rows with an attached traceback, most recent first
pm-log-cli query --has-exception --limit 20

# Free-text search across every process's messages
pm-log-cli query --grep "Connection reset"

# One process's logging for a specific subsystem
pm-log-cli query --process pm-api-gwy --logger "edumatcher.api_gateway.%"

# A specific time window, oldest first
pm-log-cli query --since 2026-07-28T09:30:00.000Z --until 2026-07-28T09:35:00.000Z --reverse

# Export as JSON for further processing
pm-log-cli --format json query --process pm-engine --limit 5000 > engine_log.json
```

32.6.4 `tail` — Follow new rows in real time

Polls for new rows past the highest sequence number already shown and prints them as they arrive. There is no push-based subscription — `tail` polls on `--interval` seconds.

```
pm-log-cli tail [options]
```

Options:

Flag	Default	Description
`--process NAME`	(all)	Filter by process name
`--level LEVEL[,LEVEL...]`	(all)	Filter by one or more levels
`--logger PATTERN`	(all)	SQL `LIKE` pattern on logger
`--interval SEC`	1.0	Polling interval

Examples:

```
# Follow everything as it arrives
pm-log-cli tail

# Follow only one process's warnings and above
pm-log-cli tail --process pm-md-gwy --level WARNING,ERROR,CRITICAL

# Follow as JSON for piping into another tool
pm-log-cli --format json tail | jq .
```

Stop with Ctrl-C.

32.6.5 `processes` — Connected/recently-connected processes

Lists rows from the connection registry — a quick "what's currently sending logs" view.

```
pm-log-cli processes [options]
```

Options:

Flag	Default	Description
`--active`	off	Only currently-connected sessions (no `disconnected_at`)

Output columns: `process`, `instance`, `pid`, `host`, `connected_at`, `last_seen_at`, `last_heartbeat_at`, `disconnect_reason`, `log_count`.

`last_seen_at` advances only when the process actually *logs* something; `last_heartbeat_at` advances on any contact, including the LALF `HB` frame an idle client sends every few seconds. A process with an old `last_seen_at` and a fresh `last_heartbeat_at` is connected and healthy with nothing to report. Both being stale is the case worth looking into.

`disconnect_reason` says *how* a session ended: `client_exit` means the client sent `EXIT` and shut down deliberately, while `peer_closed`, `io_error` (the connection was reset, typically a crash), `idle_timeout`, `hello_timeout`, or `protocol_error` mean it went away without saying goodbye. It is empty for sessions still open, and for sessions recorded before this column existed.

A session appears here only once it has actually sent a `LOG` or `HB` frame, not merely completed the handshake. Every `pm-*` process probes for a running `pm-log-srv` at startup by opening a throwaway connection, sending `HELLO`, reading `WELCOME` and hanging up again; counting those probes as processes filled this table with tools that were never started. If you are looking for a process you expect to be connected and it is absent, it has either not started or not yet logged anything.

Examples:

```
# Every session seen, connected or not
pm-log-cli processes

# Only processes still connected right now
pm-log-cli processes --active

# Export as JSON for a dashboard
pm-log-cli --format json processes
```

32.6.6 `stats` — Server + database health summary

Reports `pm-log-srv`'s own lifetime counters plus a handful of cheap aggregate queries over `log.db` — a quick "is logging healthy" dashboard.

```
pm-log-cli stats
```

Example output:

```
pm-log-srv Statistics
========================================
  Started at:        2026-07-28T22:26:30.202Z
  Total log events:  3
  Total connections: 1
  Total truncated:   0
  Total errors sent: 0
  Rows in log.db:    3
  DB file size:      45,056 bytes

  By level:
    ERROR      1
    INFO       1
    WARNING    1

  By process:
    pm-md-gwy            3
```

Examples:

```
# Quick health check
pm-log-cli stats

# Machine-readable for monitoring/automation
pm-log-cli --format json stats
```

32.6.7 `diagnose` — Rule-based troubleshooting report

Applies a small, fixed, documented set of rule-based heuristics against stored logs and reports likely causes plus concrete, reproducible recommendations. Every heuristic is a plain SQL aggregate query plus a fixed threshold, not a statistical or ML model — "why did it flag this" always has a one-sentence, inspectable answer.

```
pm-log-cli diagnose [options]
```

Options:

Flag	Default	Description
`--since ISO_TS`	(all)	Restrict to events since this time
`--process NAME`	(all)	Restrict to a specific process

The seven heuristics:

Heuristic	What it detects
Error-rate spike	ERROR/CRITICAL count in the last 5 minutes far exceeds the preceding hour's baseline for a process
Repeated identical warning	The same `(process, logger, message)` triple repeats many times in the queried window
Process silence	A still-connected session has stopped both logging and heartbeating — reported per process. Processes that are merely quiet — no logs, but still heartbeating — are collapsed into a single `[INFO]` observation instead
Clock skew	A process's `client_ts` is consistently ahead of or behind `pm-log-srv`'s own clock
Truncated-message rate	A process sent one or more oversized log messages that were truncated
Exception clustering	Most tracebacks in the window come from one specific logger
Likely fallback-to-file event	A process's *most recent* session ended without a clean `EXIT` and it has not reconnected — it either exited abnormally or failed over to file logging. Deliberate shutdowns are ignored

Each finding cites the exact `pm-log-cli query` invocation that would reproduce it, so you can always drill from "here's what's flagged" to "here are the actual log lines." When no heuristic fires, `diagnose` reports "no issues detected" rather than printing nothing — a clean run is visibly a clean run. Exit code `3` signals at least one finding, useful for scripting a periodic health check.

Findings carry a severity. `[INFO]` findings are observations that need no action — a quiet-but-heartbeating process is the common one — and they do *not* set exit code `3`, so a scripted health check will not be woken by them. Only `[WARNING]` and `[ERROR]` findings do.

Example output (an issue found):

```
[WARNING] repeated_warning: pm-api-gwy logged "ENGINE_TIMEOUT" 47 times in the last hour
  Recommendation: Investigate why edumatcher.api_gateway.engine_client keeps repeating this message; it may indicate a stuck retry lo
  Reproduce with: pm-log-cli query --process pm-api-gwy --logger 'edumatcher.api_gateway.engine_client' --grep 'ENGINE_TIMEOUT'
```

Example output (clean session):

```
No issues detected in the queried window.
```

Examples:

```
# Full-database health check
pm-log-cli diagnose

# Restrict to one process
pm-log-cli diagnose --process pm-md-gwy

# Restrict to the last hour
pm-log-cli diagnose --since 2026-07-28T13:00:00.000Z

# Scripted health check — non-zero exit means something was flagged
pm-log-cli --format json diagnose || echo "logging issues detected, see above"
```

32.6.8 `prune` — Manual retention pruning

Deletes `log_events` rows older than a given number of days — the same operation `pm-log-srv` already runs automatically once per hour, exposed here for on-demand use (e.g. before archiving `log.db`, or to shrink it immediately rather than waiting for the next scheduled pass).

```
pm-log-cli prune [--days N]
```

Options:

Flag	Default	Description
`--days N`	`30`	Delete rows with `client_ts` older than N days

Examples:

```
# Prune using the default 30-day window
pm-log-cli prune

# Aggressive pruning before archiving
pm-log-cli prune --days 7
```

`prune` is the one subcommand that opens `log.db` for writing rather than read-only, since it needs to delete rows; it is still safe to run while `pm-log-srv` is running, since SQLite serializes the two writers.

32.7 Cookbook — Common Log-Server Workflows

Examples below are grouped by the kind of question you're trying to answer.

32.7.1 Watching a live session

```
# Follow everything as it happens
pm-log-cli tail

# Follow just the warnings and errors across every process
pm-log-cli tail --level WARNING,ERROR,CRITICAL

# Follow one specific gateway
pm-log-cli tail --process pm-md-gwy
```

`pm-log-cli tail` polls `log.db`, which is exactly right for a terminal you are watching for a few minutes. For anything long-lived and interactive — a viewer window, a dashboard — subscribe over LALF-PS instead and be pushed rows as they land.

32.7.2 Investigating an error after the fact

```
# All errors in the last hour
pm-log-cli query --level ERROR,CRITICAL --since 2026-07-28T13:00:00.000Z

# All rows with an attached traceback
pm-log-cli query --has-exception

# Full traceback text for one specific event, exported for a support ticket
pm-log-cli --format json query --process pm-md-gwy --has-exception --limit 1
```

32.7.3 Checking whether every process is still reporting in

```
# Everything currently connected
pm-log-cli processes --active

# Everything ever seen, to spot a process that connected once and vanished
pm-log-cli processes

# Automated health check — flags process silence and fallback-to-file events
pm-log-cli diagnose
```

32.7.4 Free-text searching across the whole system

```
# Find every mention of a specific error string, any process
pm-log-cli query --grep "Connection reset"

# Narrow to one subsystem's loggers
pm-log-cli query --grep "timeout" --logger "edumatcher.api_gateway.%"
```

32.7.5 Exporting for external analysis

```
# Export a whole day's logging for one process
pm-log-cli --format json query --process pm-engine \
  --since 2026-07-28T00:00:00.000Z --until 2026-07-28T23:59:59.000Z \
  --limit 100000 > pm-engine_2026-07-28.json

# Export the full stats snapshot for a dashboard
pm-log-cli --format json stats > log_stats.json
```

32.7.6 Routine maintenance

```
# Quick health check before end of day
pm-log-cli stats
pm-log-cli diagnose

# Prune early if log.db is getting large before its scheduled hourly prune
pm-log-cli prune --days 30
```

32.8 Error Handling

Exit code	Meaning
`0`	Success
`1`	Database not found or not readable — has `pm-log-srv` ever run?
`2`	Argument error
`3`	`diagnose` found at least one warning- or error-level finding

Unlike `pm-audit-cli`, `pm-log-cli` has no separate index-building step and no rotated-file discovery — `log.db` is a single SQLite file and every subcommand reads (or, for `prune`, writes) it directly.

32.9 See Also

In this book:

- Configuration — Configuring pm-log-srv — the `log_server:` config block field reference
- Audit Trail — the equivalent dedicated-collector pattern for trading events (`pm-audit`/`pm-audit-cli`)
- Statistics and Reporting — the equivalent dedicated-collector pattern for market data (`pm-stats`/`pm-stats-cli`)

Reference Manual:

- Processes — pm-log-srv / pm-log-cli — startup reference tables in the process overview

Protocols and Clients:

- LALF Protocol Reference — normative wire specification for anyone implementing a LALF client
- Message Reference — LALF-PS messages — normative field tables for every LALF-PS message

CHAPTER 33

Log Operator Console (`pm-log-ui`)

Learning objectives

After reading this page you will understand:

- What the Log Operator Console is for, and the one question it is built to answer
- How it relates to `pm-log-srv` and `pm-log-cli`, and why it replaces neither
- Where LALF and LALF-PS fit — and why the console speaks only one of them
- Every way to start the application, and what each one needs
- What each of the six views shows and how to work with it
- The acknowledgement model, and why it is shared rather than per-browser
- How to configure, secure, and troubleshoot a deployment

33.1 Overview

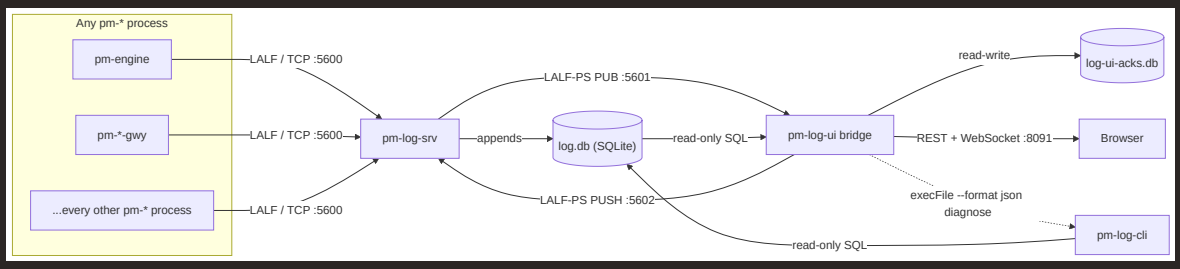
The **Log Operator Console** (`pm-log-ui`) is a browser-based console over the logging collected by `pm-log-srv`. Where `pm-log-cli` answers "what did this process log?", the console answers a different and more operational question:

> **Is anything wrong right now — and has someone dealt with it?**

That second clause is the part no CLI provides. A `diagnose` report tells you something is broken; it cannot tell you that a colleague already looked at it twenty minutes ago and decided it was benign. The console adds a shared acknowledgement layer on top of the same data, so a room full of operators converges on one view of what is outstanding rather than each re-investigating the same recurring error.

Component	Role	Type
`pm-log-srv`	Collector — receives LALF from every `pm-*` process, appends to `log.db`, republishes live rows over LALF-PS	Long-running process
`pm-log-cli`	Query/troubleshooting CLI — reads `log.db` directly	One-shot CLI
`pm-log-ui` bridge	Backend — one LALF-PS subscription, read-only `log.db` access, WebSocket fan-out, acknowledgement store	Long-running process
`pm-log-ui` web	Frontend — six views over the bridge's REST + WebSocket API	Browser application

The console is **strictly additive**. It never writes to `log.db`, never alters `pm-log-srv`'s behaviour, and can be stopped at any time without affecting log collection. Everything it displays remains available through `pm-log-cli` whether the console is running or not.



33.2 How it relates to the log server

The console is a **consumer** of `pm-log-srv`, and it consumes it through two independent channels that fail independently — a distinction that matters because the UI reports them separately rather than collapsing them into one "connected" light.

	Live channel	History channel
Source	LALF-PS (ZeroMQ) from `pm-log-srv`	`log.db` read directly from disk
Carries	New rows as they land, server heartbeats	Search, aggregation, counts, export
Works when `pm-log-srv` is stopped	No	**Yes**
Works when `log.db` is unreachable	**Yes**	No
Latency	Pushed on commit	Query-time

Because history comes from the file rather than the wire, the console deliberately **never sends `log.backfill_request`**. Backfill exists in LALF-PS for subscribers with no filesystem access to `log.db`; this bridge has that access, so replaying history over ZeroMQ would be strictly worse — slower, bounded by the server's backfill limits, and competing with live delivery.

Three consequences worth internalising before you deploy:

- **Stop `pm-log-srv` and the console still works** for everything historical. The top bar shows *log server down*, the live tail stops, and search, aggregation and export continue unaffected.
- **Move `log.db` out from under the bridge and the live tail still works.** The top bar shows *log.db unavailable*, and history views return a clean 503 rather than a blank page.
- **The bridge holds exactly one LALF-PS subscription** no matter how many browser tabs are open. Ten operators watching the console cost `pm-log-srv` one subscriber, not ten. Fan-out to tabs happens over the bridge's own WebSocket.

The console is not a substitute for pm-log-cli

`pm-log-cli` remains the tool of record for scripting, for exporting large ranges, for hosts with no browser, and for any situation where you need the log data without a running service. The console is a *viewing* layer; `pm-log-cli` is the *access* layer. The Diagnostics view makes this explicit by shelling out to `pm-log-cli` rather than reimplementing it.

33.3 LALF, LALF-PS, and where the console sits

This is the single most common source of confusion, so it is worth stating plainly:

The Log Operator Console does not speak LALF.

Protocol	Direction	Transport	Who speaks it
LALF	`pm-*` process → `pm-log-srv`	TCP `:5600`, line-oriented text	Every process that logs. Normative spec: LALF Protocol Reference
LALF-PS	`pm-log-srv` → subscriber	ZeroMQ `PUB :5601` / `PULL :5602`, JSON	Live viewers, including this console. Reference: LALF-PS

LALF is a *producer* protocol: a process opens one long-lived TCP connection, does a HELLO/WELCOME handshake, and streams its own LOG records until it exits. It is unidirectional and subscription-free — a LALF client can only ever send its own logging, never read anyone else's.

That property is exactly why the console cannot use it. A viewer needs to *read* everything, which is the opposite of what LALF offers. So pm-log-srv exposes a second, separate interface — LALF-PS — and the console subscribes to that.

Read LALF Protocol Reference when you are instrumenting a process to send its logs. Read it for this console only to understand the shape of the rows that eventually arrive: the LOG header field set in §"LOG" is what becomes a row in every table and detail pane described below.

33.3.1 What the bridge does on the LALF-PS wire

The bridge implements the subscriber side in web-apps/log-gui/apps/bridge/src/lalf-ps-uplink.ts. The sequence, in order:

1. Connect SUB to :5601 and PUSH to :5602.
2. Subscribe to the ZeroMQ topic prefix **log.** — not log.<sub_id>. The server publishes per-subscriber topics as log.<msgtype>.<sub_id> (plus the broadcast log.server_state), so the shared prefix is the only filter that catches them all. The bridge then discards anything not addressed to its own sub_id.
3. Wait 200 ms before sending anything. ZeroMQ's **slow-joiner** behaviour silently drops messages published before a SUB connection finishes establishing; without this settle the first subscribe_ack can vanish.
4. Send log.subscribe with mode: STREAM and a lease.
5. Retry the subscribe if no ack arrives within one second. Subscribe is idempotent by sub_id, so a duplicate is harmless.
6. Renew at **half the lease interval**, using renew_before_sec from the ack when the server supplies one.
7. On log.lease_expired or a log.error with UNKNOWN_SUB, re-subscribe immediately rather than assuming the old lease survived.

The sub_id is pm-log-bridge-<pid> by default, so two bridges on one host do not collide. It is shown in the Health view — quote it if you need to correlate the console against pm-log-srv's own subscriber list.

The bridge tracks four uplink states, surfaced in the top bar:

Uplink state	Top bar	Meaning
CONNECTING	reconnecting…	Subscribe sent, no ack yet
ACTIVE	log-srv up	Lease held, rows flowing
RECONNECTING	reconnecting…	Lease expired or unknown; re-subscribing
SERVER_DOWN	log server down	log.server_state said DOWN, or heartbeats stopped

33.4 Running the application

Four supported ways, in increasing order of setup effort. If you are not sure, use the first.

Method	Best for	Needs	URL
The whole stack	Almost everyone — it starts the exchange too	Podman/Docker + Make	http://localhost:8091
This app alone, in a container	An exchange that runs elsewhere	Podman/Docker + Make	http://localhost:8091
Pre-built image artifact	Offline hosts, no repo clone	Podman/Docker only	http://localhost:8091
Local development	Contributing to the console itself	Node.js + npm	http://localhost:8191

33.4.1 Prerequisites

Whichever method you choose, the console needs to reach **both** of its sources:

- **pm-log-srv's LALF-PS ports** — 5601 (PUB) and 5602 (PULL) by default. Note these are *not* the LALF port 5600; the console has no use for that one.
- **A filesystem path to log.db** — read access is sufficient, and read-*only* is preferred.

Any modern browser (Chrome/Edge 110, Firefox 115, Safari 16). No extensions are required.

The console needs two things from the exchange, and this is the whole of it:

It needs	For	Without it
pm-log-srv's LALF-PS sockets (5601/5602)	The live tail	History still works; nothing streams
The file log.db	Search, history, alerts, acknowledgements	The live tail works; the history views are empty

Note the second one: this console reads a **file**, not just a socket. Any way of running it has to give it a path to the same log.db that pm-log-srv writes.

33.4.2 Option 1 — The whole stack (recommended)

Starts the exchange and the other applications together, with the console already pointed at the exchange's own data directory:

A released install:

```
cd ~/.edumatcher
./edumatcher.sh start
```

From a source checkout:

```
cd deployment/docker
make up-all
```

Open **http://localhost:8091**. Nothing needs configuring: the console reaches pm-log-srv at the hostname edumatcher on the shared Compose network, and the exchange's data directory is mounted read-only at /backend-data, so log.db is the very file the collector is writing.

Everyday commands from deployment/docker (or ./edumatcher.sh in a released install):

```
make status      # the exchange's process table
make mounts      # which host directory is behind each container path
make down-all    # stop everything
```

make mounts is worth knowing about here specifically. The console's health page reports *container* paths — it will say its database is /backend-data/log.db, which tells you nothing about whose data that is. If you ever doubt which exchange you are looking at, that command answers it.

33.4.3 Option 2 — This app alone, in a container

Use this when pm-log-srv runs somewhere else. From web-apps/log-gui/:

```
make up
```

make up auto-detects Podman or Docker (preferring Podman), starts the Podman machine on macOS if needed, builds the image, and starts the stack detached. Open **http://localhost:8091**.

```
make down      # stop and remove the stack
make restart   # restart it
make logs      # follow the container log
make ps        # show stack status
make help      # list every target
```

Two things need pointing at the exchange, because this container is not beside it. Set LOG_SRV_HOST to wherever pm-log-srv is reachable *from inside the container* — host.docker.internal on Docker Desktop, host.containers.internal on Podman, or a real hostname:

```
LOG_SRV_HOST=host.containers.internal make up
```

CALF_HOST names where pm-md-gwy is reachable in the same way; the top bar's session phase comes from it (see below).

And mount the directory that holds log.db at ./logsrv-data. This app's compose file mounts two volumes, deliberately:

```
volumes:
  # pm-log-srv's directory, mounted READ-ONLY
  - ./logsrv-data:/app/logsrv-data:ro
  # this project's own ack store, read-write
  - ./ack-data:/app/ack-data
```

The :ro on the first mount is not decoration. "This project never writes to log.db" is enforced by the kernel rather than by application code, so a bug in the console cannot corrupt the collector's database.

33.4.4 Option 3 — A pre-built image artifact

For a host with no repository clone, build a self-contained OCI archive:

```
make cdist
```

This produces dist/edumatcher-log-gui-<version>.tar.xz containing the Node runtime, the compiled frontend and the bridge — no external dependencies. On the target host:

```
# Podman
podman load --input edumatcher-log-gui-<version>.tar.xz
podman run -d --name log-gui -p 8091:8091 \
  -e LOG_SRV_HOST=logsrv.example \
  -e CALF_HOST=exchange.example \
  -v /srv/logsrv-data:/app/logsrv-data:ro \
  -v /srv/log-ui-acks:/app/ack-data \
  edumatcher-log-gui:<version>

# Docker — identical, substituting `docker` for `podman`
```

No Node.js or npm required

The archive is fully self-contained. The host needs only Podman or Docker.

33.4.5 Option 4 — Local development

```
cd web-apps/log-gui
make install      # npm ci from the lockfile; once, and after a dependency change
make dev          # web dev server (8191) + bridge (5191) together
```

Open **http://localhost:8191**. The Vite dev server hot-reloads on save and proxies both /api/* and /ws/* to the bridge on 127.0.0.1:5191, so you only ever open the 8191 URL.

Running against the container stack works with no addresses to set — the bridge defaults to 127.0.0.1, which is where the stack publishes — with one exception: **the live tail needs ZMQ=1**, because 5601/5602 are ZeroMQ sockets that the stack does not publish by default.

```
cd deployment/docker && make down-all && make up-all ZMQ=1
eval "$(make -s dev-env GUI=log-gui)"     # exports LOG_DB_PATH and ACK_STORE_PATH
cd ../../web-apps/log-gui && make dev
```

Without ZMQ=1 the history views still work — they read log.db directly — and only the live stream is missing. See The Development Loop.

For separate logs, run the two halves in two terminals:

```
make dev-bridge   # Fastify: LALF-PS + log.db + WebSocket
make dev-web      # Vite dev server only
```

This assumes pm-log-srv is running locally on its default ports and that log.db exists. The bridge resolves log.db's location by mirroring edumatcher.config._resolve_data_dir()'s priority order exactly — EDUMATCHER_DATA_DIR, then <repo>/src/data, then the installed-package home directory fallback — so in a normal checkout neither side needs configuring to agree with the other. It logs the resolved absolute paths at startup; check that line first if a view is unexpectedly empty.

A known npm dedup bug

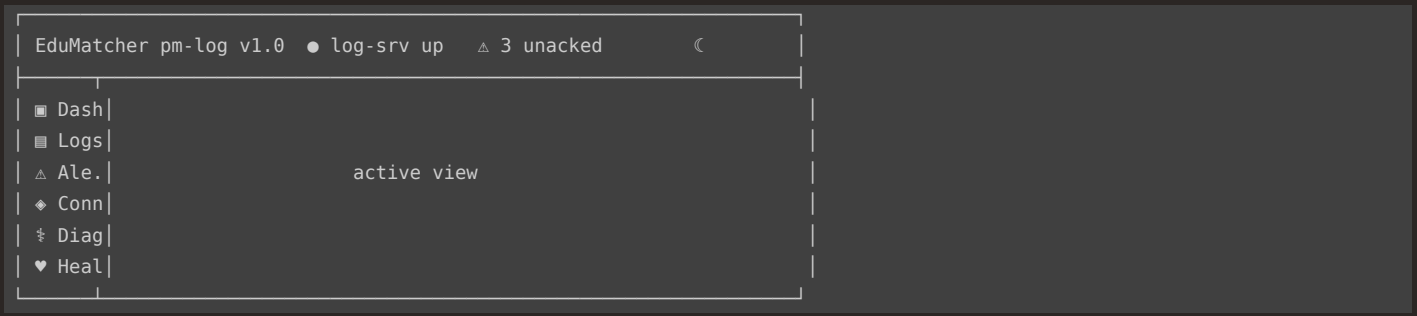
If `package-lock.json` is ever deleted and regenerated, some npm versions (observed on 11.17.0) crash with `TypeError: Invalid Version:` from `@npmcli/arborist`'s dedup step. `make install` already works around this by running `npm install --no-dedupe` for the first resolution. If you hit the crash by calling `npm install` directly, rerun with `--no-dedupe`. See `web-apps/log-gui/README.md` for the full explanation.

33.4.6 Developer commands

```
make test        # Vitest: log-query, fingerprinting, issue index, ack store
make typecheck   # type-check every workspace
make build       # production frontend bundle
make lint        # type-check (lint alias)
make format      # Prettier
make clean       # remove build artifacts and dist/
```

33.5 A tour of the interface

The window is a fixed shell: a **top bar** carrying global state, a **six-item navigation rail** on the left, and the active view filling the rest.



The top bar is deliberately sparse. After the name and version (`EduMatcher pm-log v<version>`) it holds four things, always visible:

- **Source state dot** — green *log-srv up*, yellow *reconnecting…*, red *log server down* or *log.db unavailable*. The two red states are distinct because they mean different things and have different fixes.
- **Unacked count** — a link to Alerts, styled red when non-zero. Also mirrored as a badge on the Alerts rail item.
- **Session indicator** — the exchange session phase as a coloured badge, the next phase and the time left to it (`Continuous → Closing Auction in 03:08`), and the clock in the exchange session timezone, exactly as the Trading GUI shows it. The bridge reads it from `pm-md-gwy` over CALF, which needs no login or API key. The badge reads `Unknown` until `pm-md-gwy` has reported the phase, and whenever it cannot be reached.
- **Theme toggle** — light/dark. The icon shows the theme a click switches *to*, not the one currently active.

The AI-trader simulation after a `stats.db` lock made pm-stats drop records. The top bar shows the source state (`log-srv01 log-srv up`) and the red `4 unacked` link; the Alerts rail item carries the same count. On the Dashboard: the banner naming the most recent affected process, the five meter tiles, events over time, top processes by error count, and the five most recent errors.

33.5.1 The six views at a glance

View	Answers	Refresh
Dashboard	Is anything wrong right now?	Live counters + 15–30 s polls
Logs	What exactly happened?	Live tail or on-demand query
Alerts	What is outstanding, and who has it?	10 s
Connections	Is everything still reporting in?	10 s
Diagnostics	What would `pm-log-cli diagnose` say?	60 s
Health	Which component is degraded?	10–30 s

View	Answers	Refresh

33.5.2 Dashboard

The at-a-glance view, designed to be readable from across a room.

Top of the view is an **alert banner**, shown only when unacknowledged issues exist. It pulses, names the most recent affected process, and links straight to Alerts. When nothing is outstanding the banner is absent entirely — the absence of red is the signal.

Below it, five meter tiles: total events, current error rate per minute, warnings in the last 60 s, process count, and log-server state. Then two charts — events over time (1-minute buckets over the last hour) and top processes by error count — and a compact list of the five most recent errors.

The **error-rate tile is banded** — it carries a colour and a word (`normal`, `elevated`, `high`, `SEVERE`) rather than a bare number, because "18 errors a minute" means nothing without knowing what this venue considers busy. The boundaries are the `ERROR_RATE_*` thresholds.

The error rate and the 60-second warning count come from the **live** WebSocket stream and update every second. Everything else is polled from `log.db`. This is why the tiles can briefly disagree during a burst: the live counters have already seen rows the aggregate query has not yet been re-run for.

33.5.3 Logs (the Explorer)

The workhorse view: a virtualised log table that runs in one of two modes, toggled by the **LIVE / history** button.

- **LIVE** streams from the WebSocket into an in-memory ring buffer of the **most recent 2 000 rows**, newest first. Rows flash briefly as they arrive.
- **history** queries `log.db` through the bridge and shows a match count.

The filter bar applies to both modes, and every filter is **kept in the URL query string** — so a filtered view is a link you can paste into a ticket or a chat message and a colleague opens exactly what you were looking at.

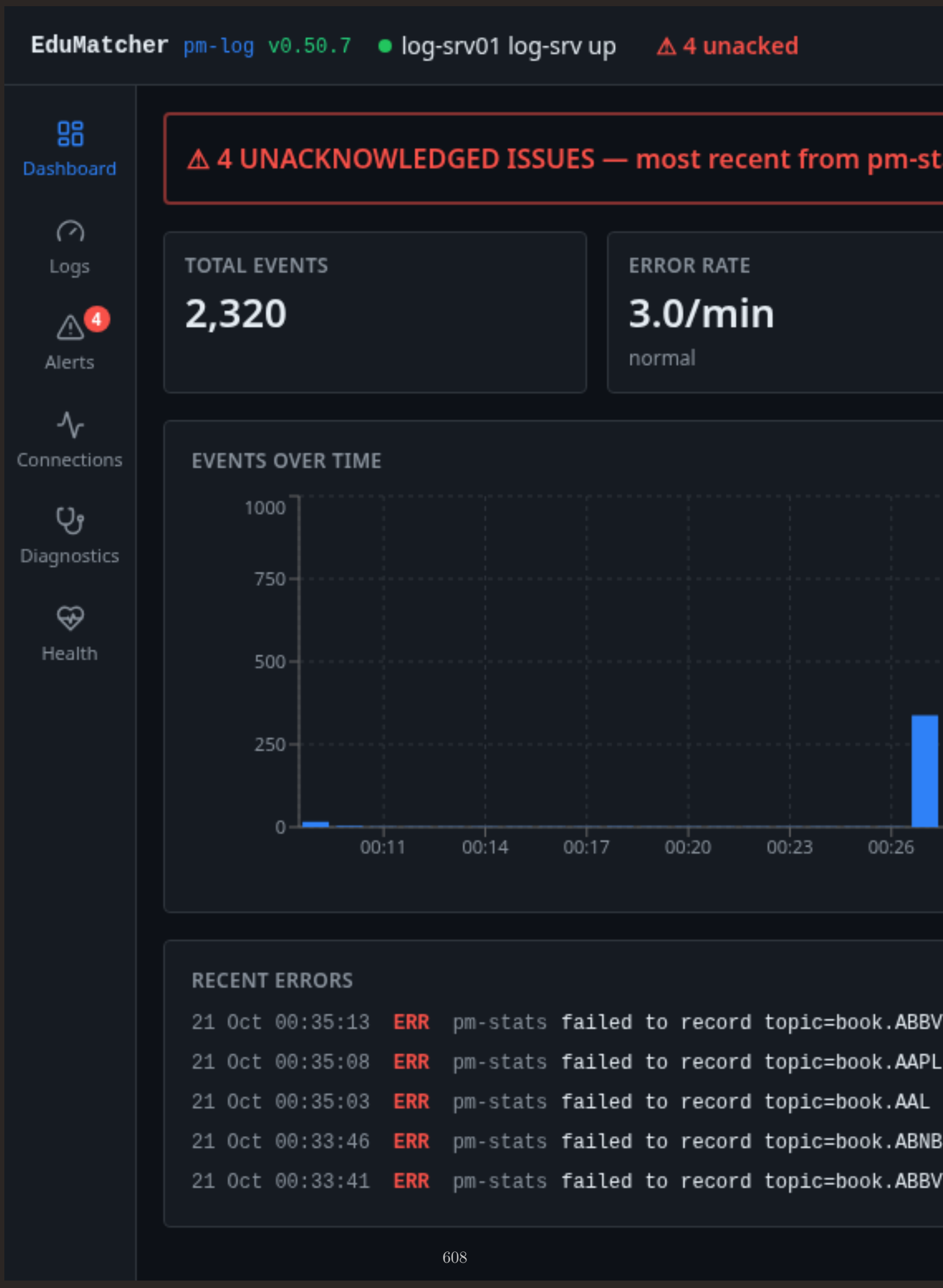
Control	Effect
`message contains…`	Substring match on the message body
Level dropdown	Minimum severity (`WARNING+`, `ERROR+`, …)
`exceptions only`	Restrict to rows carrying a traceback
LIVE / history	Switch between the live buffer and `log.db`

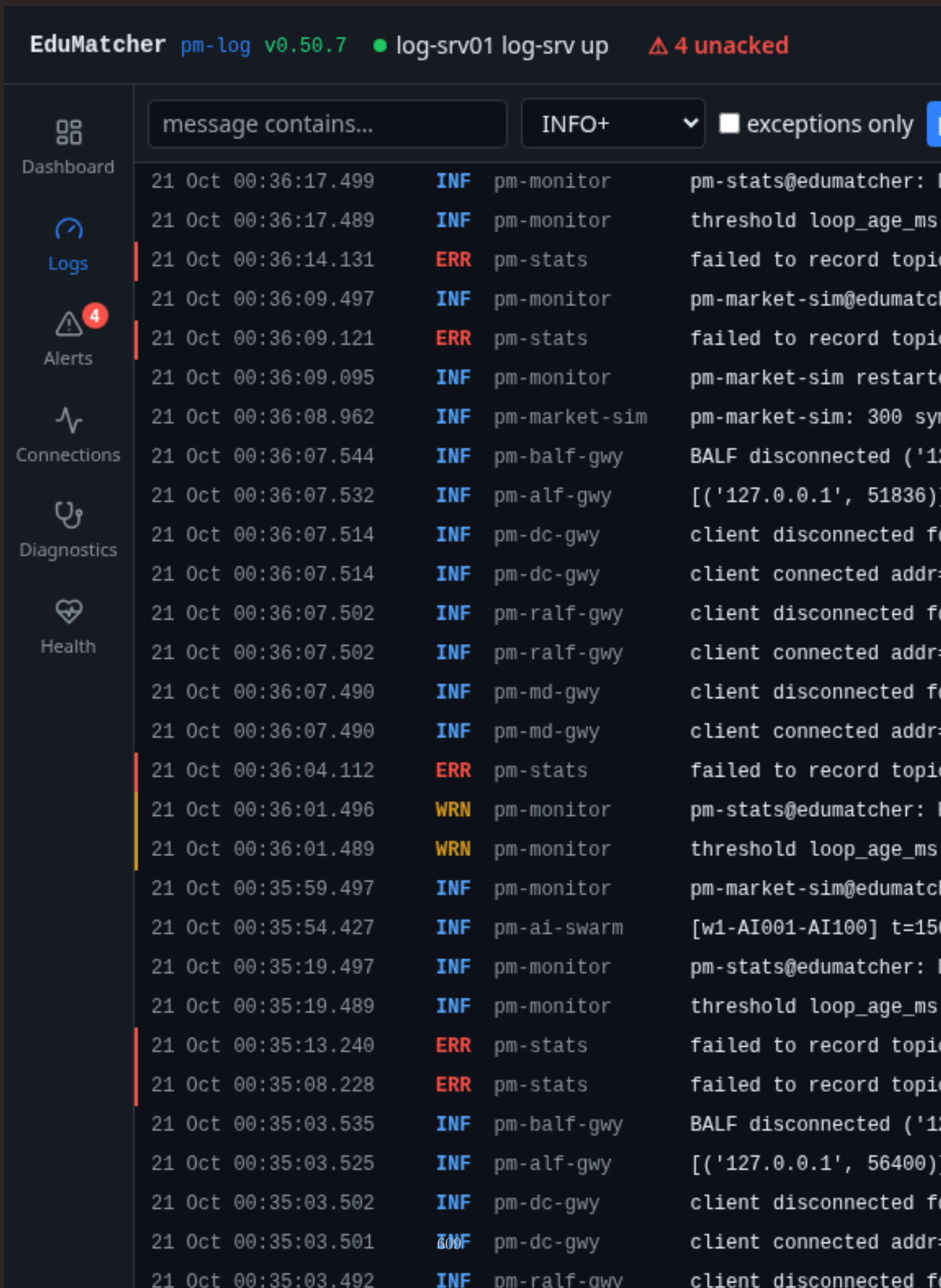
Scrolling down in LIVE mode pauses the tail. This is intentional — you cannot read a fast-moving log that keeps jumping. A banner appears showing how many rows arrived while paused; clicking it jumps back to the top and resumes. The rows are not lost, they simply were not rendered.

Delivery adapts to the rate

Under `LIVE_BATCH_THRESHOLD_PER_SEC` rows per second, each row is sent as its own WebSocket frame, which is what makes the tail feel immediate. Above it the bridge coalesces rows into batched frames flushed every 100 ms.
This is automatic in both directions and needs no operator action. It matters during a storm: at a few thousand rows a second, one frame per row is thousands of encodes, socket writes and React updates per second *per tab*, and the browser stops keeping up long before the bridge does. Ordering is preserved unconditionally — once anything is buffered, later rows queue behind it even if the rate drops back.
Clicking any row opens a **detail pane** on the right with the full `LOG` header field set — logger, module and line, pid and host, session and sequence number — plus the untruncated message. Two actions there:

- **Copy** — the message text to the clipboard.
- **Filter to this process** — rewrites the URL filter, which is usually the next thing you want after spotting one bad row.


EduMatcher pm-log v0.50.7
log-srv01 log-srv up
4 unacked
Dashboard
Logs
Alerts
4
Connections
Diagnostics
Health
4 UNACKNOWLEDGED ISSUES — most recent from pm-st
TOTAL EVENTS
2,320
ERROR RATE
3.0/min
normal
EVENTS OVER TIME
1000
750
500
250
0
00:11
00:14
00:17
00:20
00:23
00:26
RECENT ERRORS
21 Oct 00:35:13 ERR pm-stats failed to record topic=book.ABBV
21 Oct 00:35:08 ERR pm-stats failed to record topic=book.AAPL
21 Oct 00:35:03 ERR pm-stats failed to record topic=book.AAL
21 Oct 00:33:46 ERR pm-stats failed to record topic=book.ABNB
21 Oct 00:33:41 ERR pm-stats failed to record topic=book.ABBV


EduMatcher pm-log v0.50.7
log-srv01 log-srv up
4 unacked
Dashboard
Logs
Alerts
4
Connections
Diagnostics
Health
message contains...
INFO+
exceptions only
21 Oct 00:36:17.499 INF pm-monitor pm-stats@edumatcher:
21 Oct 00:36:17.489 INF pm-monitor threshold loop_age_ms
21 Oct 00:36:14.131 ERR pm-stats failed to record topi
21 Oct 00:36:09.497 INF pm-monitor pm-market-sim@edumatc
21 Oct 00:36:09.121 ERR pm-stats failed to record topi
21 Oct 00:36:09.095 INF pm-monitor pm-market-sim restart
21 Oct 00:36:08.962 INF pm-market-sim pm-market-sim: 300 sy
21 Oct 00:36:07.544 INF pm-balf-gwy BALF disconnected ('1
21 Oct 00:36:07.532 INF pm-alf-gwy [('127.0.0.1', 51836)
21 Oct 00:36:07.514 INF pm-dc-gwy client disconnected f
21 Oct 00:36:07.514 INF pm-dc-gwy client connected addr
21 Oct 00:36:07.502 INF pm-ralf-gwy client disconnected f
21 Oct 00:36:07.502 INF pm-ralf-gwy client connected addr
21 Oct 00:36:07.490 INF pm-md-gwy client disconnected f
21 Oct 00:36:07.490 INF pm-md-gwy client connected addr
21 Oct 00:36:04.112 ERR pm-stats failed to record topi
21 Oct 00:36:01.496 WRN pm-monitor pm-stats@edumatcher:
21 Oct 00:36:01.489 WRN pm-monitor threshold loop_age_ms
21 Oct 00:35:59.497 INF pm-monitor pm-market-sim@edumatc
21 Oct 00:35:54.427 INF pm-ai-swarm [w1-AI001-AI100] t=15
21 Oct 00:35:19.497 INF pm-monitor pm-stats@edumatcher:
21 Oct 00:35:19.489 INF pm-monitor threshold loop_age_ms
21 Oct 00:35:13.240 ERR pm-stats failed to record topi
21 Oct 00:35:08.228 ERR pm-stats failed to record topi
21 Oct 00:35:03.535 INF pm-balf-gwy BALF disconnected ('1
21 Oct 00:35:03.525 INF pm-alf-gwy [('127.0.0.1', 56400)
21 Oct 00:35:03.502 INF pm-dc-gwy client disconnected f
21 Oct 00:35:03.501 INF pm-dc-gwy client connected addr
21 Oct 00:35:03.492 INF pm-ralf-gwy client disconnected f

*Rows are coloured by level — `INF`, `WRN`, `ERR` — with a matching left edge on warnings and errors. The filter (`minLevel=INFO`) is in the URL. The detail pane shows the selected row's logger, module and line, pid and host, session and sequence number, the full message, and the **Copy** and **Filter to this process** actions.*

33.5.4 Alerts

Where the console earns its keep. `WARNING`-and-above rows are **fingerprinted** — normalised so that the same fault recurring a thousand times collapses into one entry with a count, rather than a thousand lines to scroll past.

Each issue card carries the level, process and logger, the occurrence count, a sample message, and first/last-seen times. Three tabs filter by state: **unacked**, **acked**, **all**.

The acknowledgement model:

- **Acknowledge** marks the issue handled, attributed to the name in the operator field, with an optional free-text note recording *what was done*.
- **Un-ack** reverses it.
- **`recurred since ack`** is flagged prominently when a fingerprint fires again after being acknowledged. This is the case that matters most: it means the fix did not hold, and it is visually distinct from a fresh issue.

Acknowledgements are shared, not per-browser

Acks live in the bridge's own SQLite file, so every operator sees the same state. That is the entire point — it is how the console prevents two people investigating the same error in parallel. It also means an ack is a **statement to your colleagues**, not a personal "hide this" toggle. Set your name in the operator field before acknowledging anything; unnamed acks are attributed to `unknown` and are much less useful during a post-mortem.
Issues with no activity beyond the retention window (7 days by default) are pruned hourly. The index is rebuilt from `log.db` at startup, so restarting the bridge loses nothing except issues that had already aged out.

*Each card is one fingerprint: level, process and logger, the occurrence count, a sample message and first/last seen. The second card was acknowledged by `instructor` with a note, then fired again — hence `recurred since ack` and **Un-ack** instead of **Acknowledge**. The operator name is set once, top right.*

33.5.5 Connections

A registry of every connection a process has made to `pm-log-srv`, one row per connection, with a status glyph:

Glyph	Meaning
●	Connected and logging
⚠	Connected but silent for **`PROCESS_SILENCE_SEC`** or more (30 s by default)
○	Disconnected

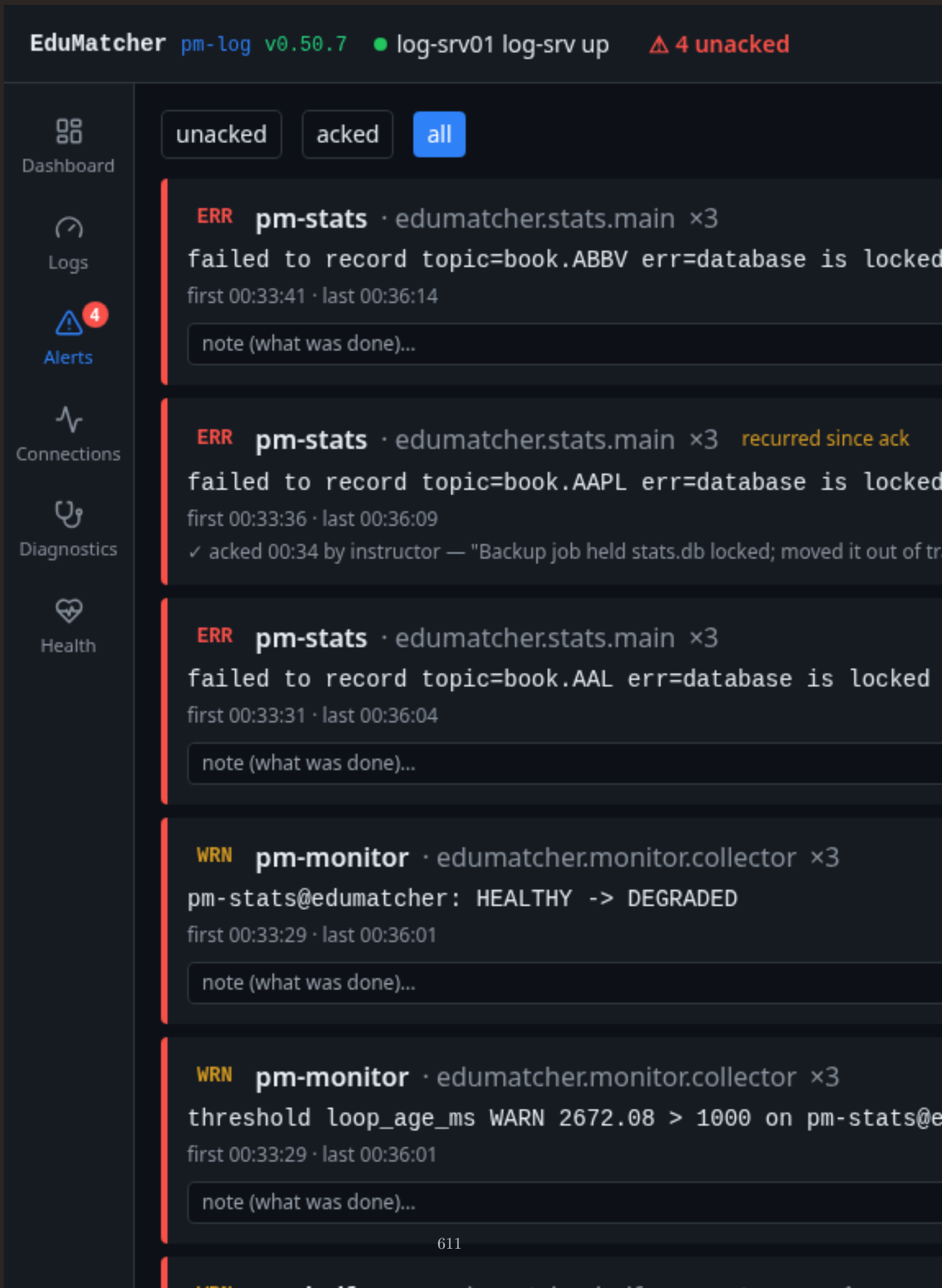
The ⚠ state is the one to watch. A process that connected and then went quiet is not obviously broken from any single log line — there is nothing to see, which is precisely the problem. Alongside each row: pid, host, connect time, idle time, total log count and error count.

The threshold is `PROCESS_SILENCE_SEC` (30 s by default) and is stated in the legend above the table, so the glyph is never ambiguous.

Hide probes (on by default) hides each process's start-up probe: before it starts logging, a process opens a short connection only to find out whether `pm-log-srv` is reachable, then closes it and opens the one it logs over. That probe would otherwise appear as a second row for the same PID — connected and disconnected in the same second, having logged nothing — which looks alarmingly like a process flapping. The checkbox shows how many it hides.

Clicking any row navigates to the Explorer pre-filtered to that process.

*All three glyphs: ● logging (the gateways and the AI traders), ⚠ silent for 30 s or more, and ○ for the two earlier `pm-market-sim` connections, which ended when it was restarted. pm-stats shows its 10 errors. **Hide probes** is on, so the two start-up probes are hidden.*


EduMatcher pm-log v0.50.7
log-srv01 log-srv up
4 unacked
Dashboard
Logs
Alerts
Connections
Diagnostics
Health
unacked
acked
all
ERR pm-stats · edumatcher.stats.main ×3
failed to record topic=book.ABBV err=database is locked
first 00:33:41 · last 00:36:14
note (what was done)...
ERR pm-stats · edumatcher.stats.main ×3 recurred since ack
failed to record topic=book.AAPL err=database is locked
first 00:33:36 · last 00:36:09
✓ acked 00:34 by instructor — "Backup job held stats.db locked; moved it out of tr
ERR pm-stats · edumatcher.stats.main ×3
failed to record topic=book.AAL err=database is locked
first 00:33:31 · last 00:36:04
note (what was done)...
WRN pm-monitor · edumatcher.monitor.collector ×3
pm-stats@edumatcher: HEALTHY -> DEGRADED
first 00:33:29 · last 00:36:01
note (what was done)...
WRN pm-monitor · edumatcher.monitor.collector ×3
threshold loop_age_ms WARN 2672.08 > 1000 on pm-stats@e
first 00:33:29 · last 00:36:01
note (what was done)...

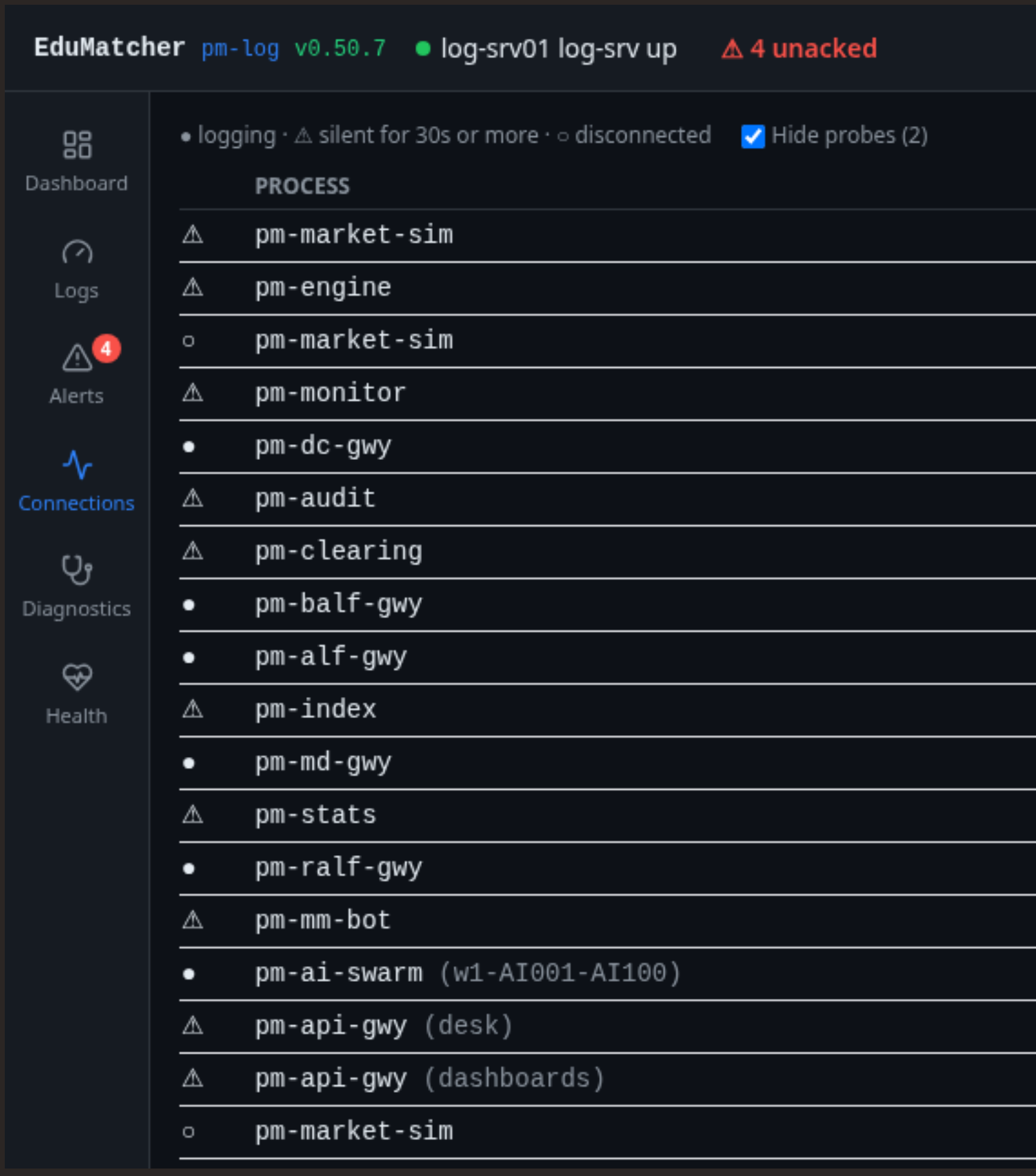


Figure 33.4: The Connections board.

33.5.6 Diagnostics

Surfaces `pm-log-cli`'s rule-based heuristics on a 60-second schedule, with a manual **Re-run** button.

Each finding shows the heuristic name, what was found, a recommendation, and — importantly — the **exact `pm-log-cli` command to reproduce it**. Heuristics that found nothing are listed compactly at the bottom, because "we checked and it was fine" is information too.

This view calls pm-log-cli, it does not reimplement it

`GET /api/diagnostics` shells out to `pm-log-cli --format json diagnose` so there is exactly one implementation of the heuristics to keep correct. The consequence: `pm-log-cli` must be installed and on `PATH` wherever **the bridge** runs. The container image installs the `edumatcher` package for exactly this reason; a bridge run some other way needs it too. If it is missing, this one view reports it and every other view keeps working. Override the command with `LOG_CLI_COMMAND`, e.g. `LOG_CLI_COMMAND="poetry run pm-log-cli"`.
See diagnose for what each heuristic actually checks.

An error-bordered `error_rate_spike` (pm-stats), a warning-bordered `repeated_warning` (the AI traders' `SESSION_LOST` after an engine restart) and an informational `process_silence`, each with its recommendation and the `pm-log-cli` command that reproduces it; the heuristics that found nothing are listed at the bottom.

33.5.7 Health

Four panels for four components that **fail independently**, kept separate rather than merged into a single status light:

Panel	Shows
`pm-log-srv`	Up/unknown, server name, lifetime events, connections, truncations, errors sent
LALF-PS	State, subscriber count, active backfills, last sequence, inbox drops, default lease
`log.db`	Path, size, row count, oldest retained row
Bridge	WebSocket client count, fingerprints indexed, acks stored, `sub_id`, `log.db` reachability

Start here when something looks wrong but you do not yet know what. The resolved `log.db` **path** in particular answers the most common misconfiguration — a bridge reading a different database than the one `pm-log-srv` is writing.

`inbox dropped` is worth a periodic glance: a rising count means `pm-log-srv` is shedding inbound LALF-PS messages, which is a capacity signal rather than a console problem.

The four independently failing components side by side. The resolved `log.db` path is `/backend-data/log.db` — the container's read-only mount of the exchange's data directory — and the bridge's LALF-PS subscription is `pm-log-bridge-26`.

33.6 Exporting

`GET /api/logs/export` returns the current filter's matches as CSV (default) or JSON:

```
curl -o logs.csv \
  'http://localhost:8091/api/logs/export?minLevel=ERROR&processes=pm-engine&format=csv'
```

The CSV carries `seq,client_ts,level,process,logger,message`. Export is capped at `EXPORT_MAX_ROWS` (1 000 000 by default), and the whole result set is materialised in memory before it is sent — a full-cap export is a large allocation in the bridge, so prefer a narrowed filter.

If the cap is reached the response carries `X-Export-Truncated: true` and `X-Export-Max-Rows`, and the bridge logs a warning. Check that header before treating an export as complete: a silently short file is the failure mode that matters here, because nothing about the CSV itself reveals that rows are missing. For anything larger — or for scripted extraction — use `pm-log-cli query --format json` instead; it reads the same database with no HTTP layer in between.

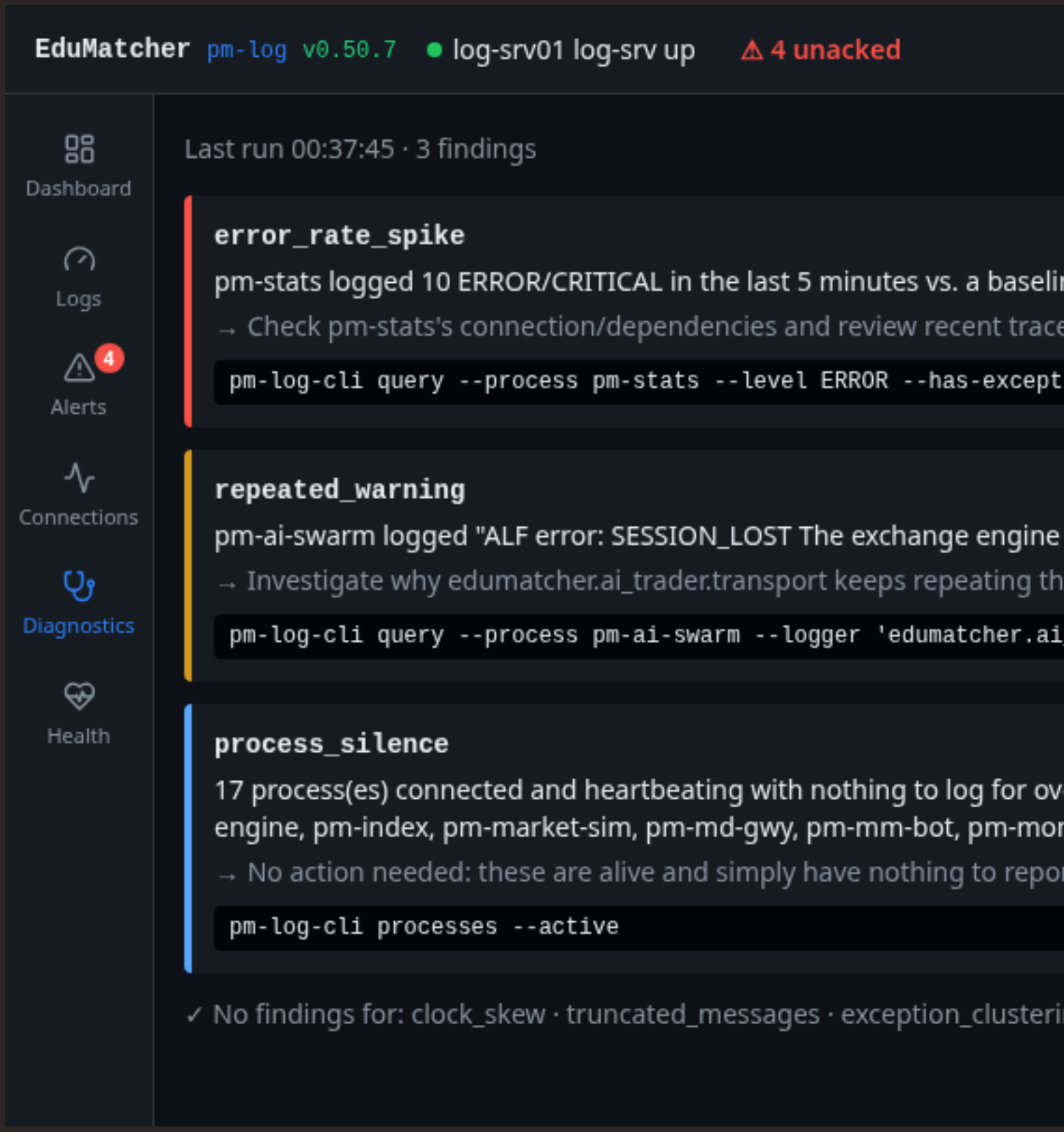


Figure 33.5: Diagnostics with three findings.

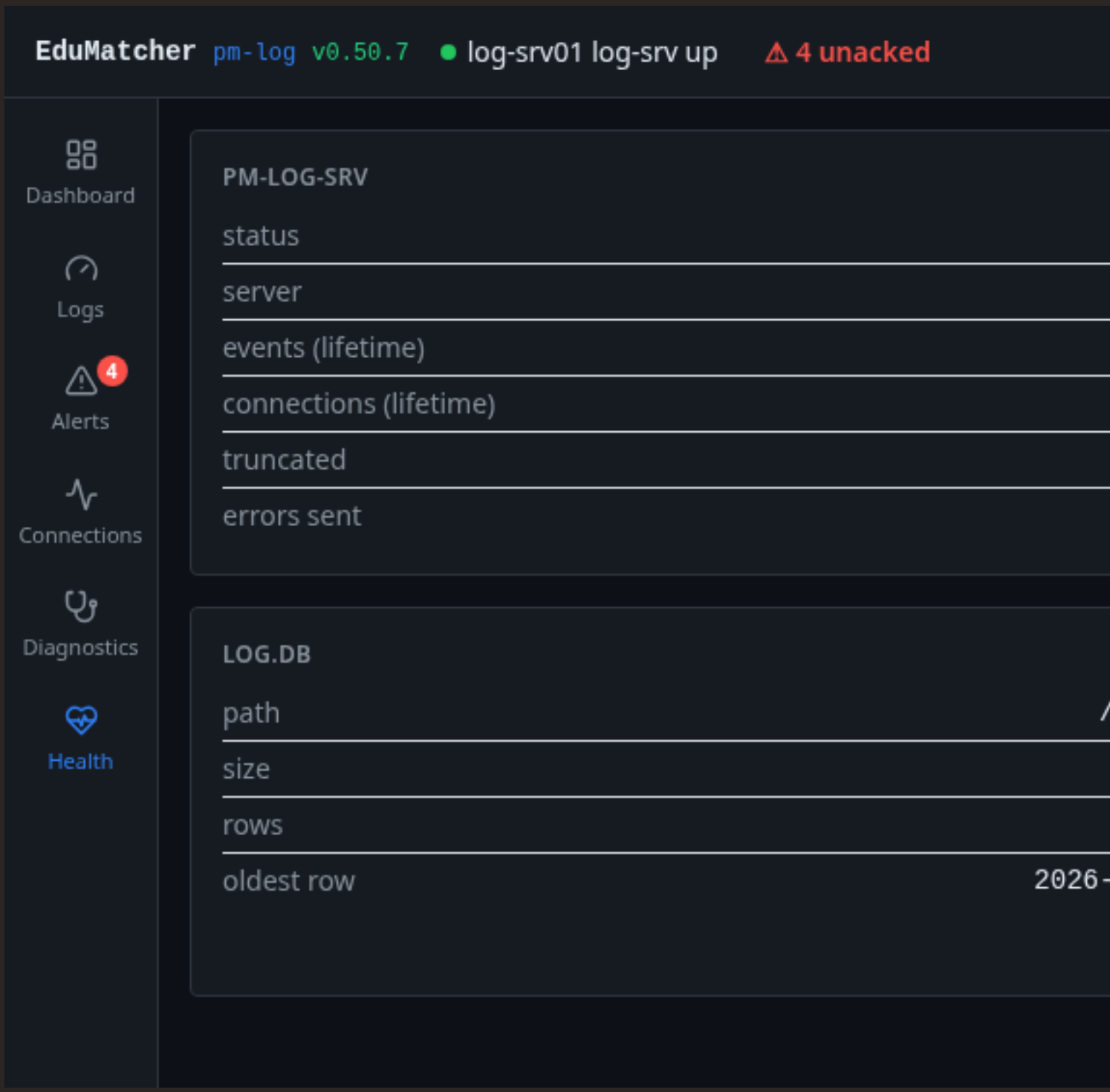


Figure 33.6: The Health view.

33.7 Configuration reference

All bridge configuration is environment variables.

33.7.1 Network and serving

Variable	Default	Purpose
HOST	127.0.0.1	Bind address. The container sets 0.0.0.0.
PORT	5191	HTTP/WebSocket port. The container sets 8091
CORS_ORIGIN	*	Allowed origin. **Restrict this** when exposed beyond localhost.
STATIC_DIR	*(unset)*	Serve the built frontend from here (single-container mode)
LOG_LEVEL	info	The bridge's own logging verbosity

33.7.2 Sources

Variable	Default	Purpose
LOG_SRV_HOST	127.0.0.1	Where pm-log-srv is reachable
LOG_SRV_PUB_PORT	5601	LALF-PS PUB port
LOG_SRV_PULL_PORT	5602	LALF-PS PULL port
SUB_ID_PREFIX	pm-log-bridge	Subscriber id prefix; -<pid> is appended
LEASE_SEC	30	Subscription lease; renewed at half this
CALF_HOST / CALF_PORT	127.0.0.1 / 5570	pm-md-gwy, for the session phase in the top bar
CALF_CLIENT_ID	pm-log-bridge	Name the bridge gives in CALF HELLO
LOG_DB_PATH	*resolved data dir* /log.db	The collector's database, read-only
ACK_STORE_PATH	*resolved data dir* /log-ui-acks.db	This project's own store, read-write
LOG_CLI_COMMAND	pm-log-cli	Command backing the Diagnostics view

33.7.3 Issues

Variable	Default	Purpose
ISSUES_RETENTION_DAYS	7	How long an inactive issue is kept, and how far back the startup scan reaches
ISSUES_MIN_LEVEL	WARNING	Minimum level that becomes an issue — applied identically to the startup scan and to live rows
ISSUES_ALERT_LEVEL	ERROR	Minimum issue level counted as "unacked" in the top bar and the Alerts badge

Both level variables are validated against the real log-level enum, so a typo yields the default rather than a silently broken filter. Setting ISSUES_ALERT_LEVEL=WARNING makes the top-bar count include warnings; setting ISSUES_MIN_LEVEL=ERROR stops warnings being fingerprinted at all, which also empties them from Alerts.

The two levels are independent, and order matters

`ISSUES_MIN_LEVEL` decides what is *tracked*; `ISSUES_ALERT_LEVEL` decides what is *counted as demanding attention*. Setting `ISSUES_ALERT_LEVEL` below `ISSUES_MIN_LEVEL` is legal but pointless — nothing below the tracking floor exists to be counted.

33.7.4 Thresholds

Variable	Default	Purpose
`PROCESS_SILENCE_SEC`	`30`	Seconds of silence after which a connected process is flagged ⚠ on the Connections board
`ERROR_RATE_NORMAL_PER_MIN`	`5`	Error rate at which the Dashboard meter leaves **normal**
`ERROR_RATE_ELEVATED_PER_MIN`	`20`	…and enters **high**
`ERROR_RATE_SEVERE_PER_MIN`	`100`	…and enters **severe**

Three thresholds, four bands. Each is the rate at which the *next* band begins, so they read as lower bounds:

Errors/min	Band	Meter colour
`< 5`	`normal`	default
`5 – 19`	`elevated`	warning
`20 – 99`	`high`	error
`>= 100`	`severe`	critical

The classification lives in `@edumatcher/log-types` (`classifyErrorRate`) so the bridge and the browser cannot drift apart on where a band starts.

33.7.5 Limits

Variable	Default	Purpose
`QUERY_MAX_ROWS`	`5000`	Hard cap on any single history query
`EXPORT_MAX_ROWS`	`1000000`	Hard cap on an export
`LIVE_BATCH_THRESHOLD_PER_SEC`	`50`	Rows/sec above which live delivery switches from per-row frames to coalesced batches

Invalid numeric values fall back to the default rather than failing startup.

33.7.6 How the frontend learns these

`GET /api/ui-config` returns `alertLevel`, `issuesMinLevel`, `processSilenceSec` and the three error-rate thresholds. The browser fetches it once and holds it for the session — it has **no local defaults**, deliberately, because a fallback constant in the UI is exactly how these settings came to be ignored in the first place. Views render a neutral placeholder until the config arrives rather than guessing.

One visible consequence: for the first moment after a page load the Connections board shows no ⚠ glyphs, because flagging every healthy process as silent would be worse than a fraction of a second's delay.

33.8 Security notes

There is no authentication

`pm-log-ui` has **no login, no user accounts, and no authorisation**. Anyone who can reach the port can read every log line the exchange has produced — which routinely includes internal hostnames, gateway identifiers, order ids and tracebacks. Treat the console as an internal operations tool:

- Bind to `127.0.0.1` (the default) or an internal interface only.
- Put it behind an authenticating reverse proxy if it must be reachable beyond the host.
- Set `CORS_ORIGIN` to your specific site origin rather than leaving `*`.
- Never expose port `8091` to an untrusted network.

Acknowledgement attribution is a **convention, not an identity claim** — the operator name is free text typed by whoever is at the keyboard. It is there to help a post-mortem reconstruct who looked at what, not to prove it.

Beyond that the posture is deliberately narrow: Helmet security headers are applied, the only writable resource is the ack store, and `log.db` is opened read-only — enforced by the `:ro` mount in the container deployment.

33.9 Troubleshooting

Symptom	Cause	Fix
Top bar shows "log server down", history works	`pm-log-srv` is stopped, unreachable, or its heartbeats stopped	Start the server; check `LOG_SRV_HOST` and that `5601/5602` are reachable *from the bridge* (inside the container, if containerised).
Top bar shows "log.db unavailable", live tail works	The bridge cannot open the database file	Compare the path in **Health → log.db** with where `pm-log-srv` writes. Set `LOG_DB_PATH` explicitly.
Every view is empty, no error shown	Bridge is reading a *different, valid* `log.db`	Check the resolved-paths line the bridge logs at startup — a valid but empty database looks exactly like a working one.
Live tail stuck, history fine	Lease not being renewed; slow-joiner drop	Check **Health → LALF-PS** state and `sub_id`. Restart the bridge; the subscribe path retries automatically but a wedged socket does not recover.
Live tail stopped after scrolling	Expected — scrolling pauses the tail	Click the "*n* new rows — jump to top" banner.
"Diagnostics unavailable"	`pm-log-cli` not on `PATH` where the bridge runs	Expected in the default container. Set `LOG_CLI_COMMAND`, or run the bridge where the CLI is installed. Every other view is unaffected.
Acks vanish after a restart	`ACK_STORE_PATH` points at ephemeral storage	Mount a persistent volume for the ack store — it is the only state the console owns.
Acks are not shared between operators	Each is running their own bridge	Run one bridge and have everyone open the same URL. Acks live in that bridge's store.
`make up` fails: neither podman nor docker found	No container runtime	Install one, or use local development.
`make up` on macOS: podman machine not running	The Podman VM is not started	`make up` starts it automatically; otherwise `podman machine init && podman machine start`.

Symptom	Cause	Fix
Container cannot reach `pm-log-srv` on `localhost`	`localhost` inside a container is the container	Use `host.docker.internal` (Docker Desktop) or `host.containers.internal` (Podman), or a real hostname.
`npm install` crashes: `TypeError: Invalid Version:`	npm dedup bug	Use `make install`, or rerun with `--no-dedupe`.
Port `8091` or `8191` already in use	Another process holds it	For the whole stack, set `LOG_GUI_PORT` in `.env`; for this app's own `make up`, set `WEB_PORT` (e.g. `WEB_PORT=8101 make up`). For the dev server, change `server.port` in `web-apps/log-gui/apps/web/vite.config.ts`.
Blank page in production, API responds	Frontend not built or `STATIC_DIR` wrong	`make build`, then point `STATIC_DIR` at an **absolute** path to `apps/web/dist`.
A client route 404s in production	Static host with no SPA fallback	Let the bridge serve the UI via `STATIC_DIR`; it falls back to `index.html`.
Explorer shows fewer rows than expected	`QUERY_MAX_ROWS` cap (5 000)	Narrow the filter, use a time range, or export instead.
An export looks complete but is short	`EXPORT_MAX_ROWS` cap reached	Check for `X-Export-Truncated: true` on the response. Narrow the filter, raise the cap, or use `pm-log-cli query`.
Top-bar unacked count differs from the Alerts list	Expected — the badge counts `ISSUES_ALERT_LEVEL` and above; Alerts lists `WARNING` and above	Set `ISSUES_ALERT_LEVEL=WARNING` if you want them to agree.
Warnings never appear in Alerts	`ISSUES_MIN_LEVEL` raised above `WARNING`	Lower it. Note it gates tracking, so raising it also empties existing warnings on restart.
Numbers on the Dashboard briefly disagree	Live counters update at 1 Hz; aggregates poll at 15–30 s	Expected. Health and the Explorer are authoritative.

33.10 Keeping the console in sync with `pm-log-srv`

Three files re-implement pieces of `pm-log-srv`'s Python surface in TypeScript. Each carries a maintenance comment naming its counterpart, and all three must be revisited when the server's schema, LALF-PS message set, or query shape changes:

TypeScript	Mirrors
`packages/log-query/src/filter-to-sql.ts`	`edumatcher.log_srv.pubsub.LogFilter.sql_where()` / `edumatcher.log_cli.queries.query_events()`
`apps/bridge/src/lalf-ps-uplink.ts`	The LALF-PS wire messages implemented in `edumatcher.log_srv.pubsub`
`apps/bridge/src/config.ts`	`edumatcher.config._resolve_data_dir()`'s priority order

`apps/bridge/src/fingerprint.ts` has no Python counterpart — issue fingerprinting is new surface this project introduces. Run `make test` after any of these changes.

33.11 See Also

In this book:

- Centralized Log Server — `pm-log-srv` and `pm-log-cli`, the data this console displays

- Configuration — Configuring pm-log-srv — the `log_server:` block, including the LALF-PS ports
- Configuration GUI (`config-gui`) — the sibling browser application, same deployment pattern

Participant Guide:

- Trader Information Terminal (TapeDeck) — the other operator-facing UI

Reference Manual:

- Processes — pm-log-srv — where `pm-log-srv` sits among the other processes

Protocols and Clients:

- Appendix: LALF Protocol Reference — normative wire spec for the producer protocol
- Message Reference — LALF-PS messages — normative field tables for every message the bridge sends and receives

CHAPTER 34

Health Monitoring (`pm-monitor`)

Learning objectives

After reading this page you will understand:

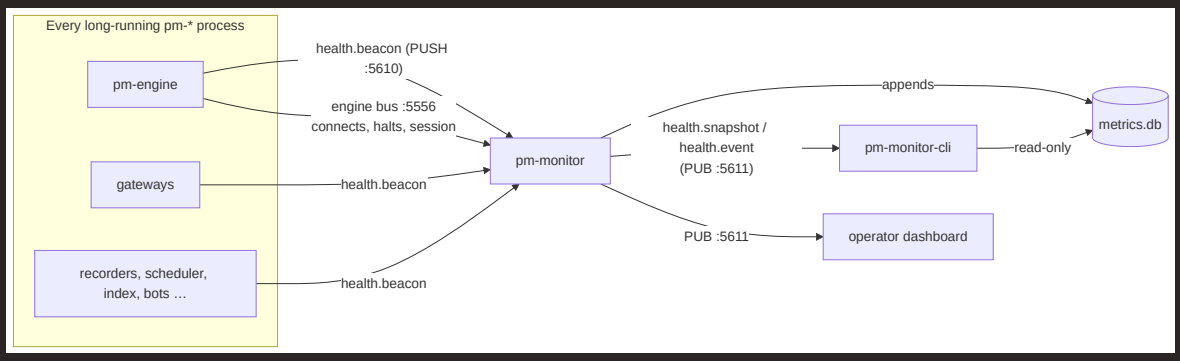
- What a health beacon is and which processes send one
- How `pm-monitor` judges each process — HEALTHY, DEGRADED, HUNG, SILENT, DEAD, MISSING, STOPPED — and the exchange as a whole
- How to start `pm-monitor`, configure it, and where `metrics.db` lives
- What the thresholds watch and how breaches and events are reported
- Every `pm-monitor-cli` command, its options and exit codes
- How to troubleshoot the common situations

34.1 Overview

`pm-opctl-cli list` answers "is the process there?": a PID and a TCP or health-ping probe. It cannot tell a process whose main loop is stuck from one that is working, it knows nothing about processes on other hosts, and it has no history. `pm-monitor` answers "is the exchange working?".

Every long-running `pm-*` process sends a small **health beacon** every two seconds: its memory, CPU, threads, how long ago its main loop last completed an iteration, and its own counters (orders in, messages out, clients connected, …). `pm-monitor` collects them, judges every process, turns the counters into rates, watches thresholds, keeps history, and publishes a live snapshot once a second.

Component	Role	Type
Health beacon	Sent by a background thread in every long-running process to `pm-monitor` (ZeroMQ PUSH to port `5610`)	Built into each process
`pm-monitor`	Collector — judges health, computes rates, evaluates thresholds, stores history, publishes `health.snapshot` and `health.event` on port `5611`	Long-running process
`metrics.db`	History: samples, 1-minute roll-ups, daily totals, events	SQLite file in the data directory
`pm-monitor-cli`	Query tool — health, exchange status, event log	One-shot CLI
Operator Dashboard	The browser view of the same snapshot and history (`pm-dash`)	Web app



The beacon is sent from a background thread, not the main loop — that is the point. When a main loop hangs, the thread keeps reporting with a growing *loop age*, so `pm-monitor` can tell a hung loop (beacons arrive, loop stuck) from a stuck interpreter (no beacons, process alive) from a dead process (no beacons, no process). A beacon sent from the main loop would simply stop, and silence cannot make that distinction.

A process never waits for `pm-monitor`: beacons are sent without blocking, at most ten queue up, and when `pm-monitor` is down they are dropped and counted. Starting the exchange without `pm-monitor` costs nothing.

34.1.1 Which processes beacon

`pm-engine`, `pm-alf-gwy`, `pm-balf-gwy`, `pm-md-gwy`, `pm-ralf-gwy`, `pm-dc-gwy`, `pm-api-gwy` (one per `--instance`), `pm-log-srv`, `pm-stats`, `pm-audit`, `pm-clearing`, `pm-scheduler`, `pm-index`, `pm-market-sim`, `pm-monitor` itself, and the participant bots `pm-mm-bot` and `pm-ai-swarm` (shown in their own group; they never affect the exchange status). Short-lived tools — the CLIs, consoles, viewers — do not.

34.2 Starting `pm-monitor`

Every built-in `pm-opctl-cli` profile except `micro` starts `pm-monitor`, right after `pm-log-srv` (so it can log) and before `pm-engine` (so the engine's first beacons are received). Its profile entry checks it with `pm-monitor-cli health -q`, so `pm-opctl-cli list` shows `healthcheck passed` only when its collector loop is actually publishing.

To start it by hand:

```
pm-monitor [--log-level LEVEL] [-v|-vv] [-q]
           [--log-target server|stdout|file] [--log-file PATH]
           [--log-failover-timeout SEC] [--version]
```

There are no other flags: everything comes from the `monitor` section of the deployed configuration (Configuration). It binds `127.0.0.1:5610` (beacons in) and `127.0.0.1:5611` (snapshots and events out) by default, subscribes to the engine bus, and opens `<data dir>/metrics.db`.

At start-up it logs, at INFO, the session timezone it judges "today" by — the engine's `locale.session_timezone`, exactly as `pm-stats` does, so "orders today" here and the statistics' daily rollup always agree.

It exits with code 1 when it cannot bind its ports, or in two cases when it cannot use `metrics.db`:

- **Another `pm-monitor` already writes this `metrics.db`** — one `pm-monitor` per data directory; two would each add their own counts to the same daily totals.
- **`metrics.db` was recorded with another session timezone** — its daily totals would mean two different things. Put `locale.session_timezone` back, or move `metrics.db` aside to start a new history.

Any other failure to open `metrics.db` — a read-only or full data directory — costs history, not monitoring: `pm-monitor` logs the error and runs on without history, and the `metrics_store_errors` rule (below) shows it DEGRADED until it is restarted with a usable data directory. A write that fails later is counted the same way.

With `monitor.enabled: false` it prints that monitoring is disabled and then **idles** until stopped (exit 0), binding nothing and writing nothing. It does not exit at once on purpose: the profiles run it, and the container restarts dead profile processes every minute.

On SIGTERM it keeps collecting for 1.5 seconds, because `pm-opctl-cli stop` and `docker stop` signal every process at once and

their final beacons arrive just after its own signal; it then records the last picture (every process STOPPED) in `metrics.db` and exits.

34.2.1 In the container

The image `EXPOSE`s 5610 and 5611, but no compose file publishes them: every process that beacons runs inside the container, and the operator dashboard reaches 5611 over the compose network. `EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0` (set by the compose file) makes `pm-monitor` bind all interfaces inside the container. Run `pm-monitor-cli` inside the container (`make shell`).

34.3 How health is judged

Every second `pm-monitor` judges each process. The first matching rule wins:

State	Means	Rule
STOPPED	Stopped cleanly	Its last beacon was the final one a process sends when it shuts down
MISSING	Expected but never seen	The active `pm-opctl-cli` profile runs it, and it has not sent a beacon since `pm-monitor` started (judged once `pm-monitor` has run for one silence period, 6 s by default)
DEAD	Gone	No beacon for `silent_after_beacons` × its beacon interval (6 s by default), and its process no longer exists. For a process on another host, which cannot be checked, after twice that
SILENT	Stuck, still there	No beacon for that long, but the process still exists: the whole interpreter is stuck, or beacons cannot reach `pm-monitor`
HUNG	Main loop stuck	Beacons arrive, but the main loop has not completed an iteration for more than `hung_after_sec` (5 s). While a process is still starting up (its loop has not run once yet) it gets 120 s instead. Time the whole process stood still — reported SILENT, after a `kill -STOP` or a suspended host — does not count
DEGRADED	Working, but something is off	A threshold on this process is in breach, or its loop ran more than 1 s late since the previous beacon
HEALTHY	—	None of the above

A process that restarted (a new incarnation of the same name, instance and host) shows a *restarted* badge for ten minutes. A process killed or stopped by hand that the profile does not expect is forgotten after an hour without a beacon.

Exchange status — the one-line answer:

Status	When
DOWN	No `pm-engine` is running: it is DEAD, SILENT, HUNG, MISSING or STOPPED, or `pm-monitor` has never heard of one

Status	When
IMPAIRED	Another expected exchange process is DEAD, SILENT, HUNG or MISSING — or STOPPED while the active profile expects it
DEGRADED	An exchange process is DEGRADED, or a threshold that belongs to no process (host CPU, memory or disk, the reject ratio, a book) is in breach
OPERATIONAL	Otherwise

The participant bots never affect the exchange status. Without an active profile (processes started by hand) every beaconing exchange process counts, and nothing is MISSING.

34.4 Thresholds and events

Rule	Subject	Warn	Critical
`host_cpu_pct`	`host`	> 80 %	> 95 %
`host_mem_pct` (total − available)	`host`	> 80 %	> 92 %
`disk_free_pct` (data directory)	`host`	< 15 %	< 5 %
`engine_handle_p99_us`	the engine	> 5 000 µs	> 20 000 µs
`reject_pct_5m` — rejected over all inbound requests (orders, cancels, amends, quotes), judged from 20 requests in 5 min	`exchange`	> 10 %	> 30 %
`rss_growth_pct_1h` — judged once a process has been watched for 10 min, only when the hour added at least 100 MB and RSS rose at least 1 % in the last 15 min (a step that has levelled off, such as at the market open, is not a leak)	a process	> 50 %	> 100 %
`heap_bloat_ratio` — a book's heap size over the size compaction tolerates; 1 is the limit	a symbol	> 1.5	> 3
`loop_age_ms` (fixed)	a process	> 1 000 ms	(HUNG)
`slow_client_disconnects`, `api_dropped_events`, `stats_feed_gaps` (fixed)	a process	any increase in 5 min	—
`audit_archive_errors` (fixed) — `pm-audit`'s archiver exited with an error	`pm-audit`	any increase in 5 min	—
`metrics_store_errors` (fixed) — history `pm-monitor` could not write	`pm-monitor`	any increase in 5 min	—

The configurable rules take their levels from `monitor.thresholds`. A breach is raised after five consecutive seconds over a warn level (a critical level is raised at once), so one noisy sample is not a breach. It clears (or eases from critical to warning) only after thirty consecutive seconds at the lower level, each with the value back inside 80 % of the threshold it crossed, so a value hovering around a threshold produces one breach and one clear, not a flood. Each breach and each clear is one `THRESHOLD` event and one log line through the log server (`threshold host_cpu_pct WARN 81 > 80 on host` at WARNING, critical at ERROR, a clear at INFO), so the log console's alerting already covers notification.

Events — the event log `pm-monitor-cli events` shows:

Kind	From	Subject · detail
`PROCESS_STATUS`	a state change	process key · `HEALTHY -> HUNG` (a process first seen HEALTHY is not news)
`THRESHOLD`	a breach or clear	the subject · `host_cpu_pct WARN 81 > 80`
`CONNECT` / `AUTH_REJECT`	engine bus	gateway id · description or reason
`DISCONNECT`	engine bus	gateway id · reason (`heartbeat_timeout` is a WARN)
`DIAGNOSTIC`	engine bus	component · `detail (xN): error`
`SESSION`	engine bus	`session · CLOSED -> PRE_OPEN`
`HALT` / `RESUME`	engine bus	symbol · level and source, or reason

34.5 Rates and "today"

Processes report counters as totals since they started; `pm-monitor` turns them into increments, so a process restart neither loses nor double-counts (a value that drops means a restart). For orders and trades it shows:

- **today** — the sum since local midnight in the session timezone;
- **per minute now** — the last 60 seconds;
- **average per minute** — the last 60 minutes;
- **average per hour** — today divided by the hours since the session opened today (else since midnight);
- **average per day** — the mean daily total over the last 7 trading days with data.

When `pm-monitor` restarts it resumes from `metrics.db`: today's totals come back, and the first beacon of a process that kept running counts exactly what happened while `pm-monitor` was down. Only the very first start of a new `metrics.db` begins the day at zero for processes that started earlier.

34.6 `metrics.db`

`<data dir>/metrics.db` (SQLite, WAL mode) holds:

Table	Content	Kept
`series`	One row per (source, metric, label); the sample tables refer to it	—
`samples_raw`	The latest value of every unlabelled series, every 10 s	`retention.raw_hours` (6 h)
`samples_1m`	Per series and minute: min, average, max, last, samples, and for counters the increment (a counter only in the minutes it moved)	`retention.minute_days` (7 days)
`daily_totals`	Per trading date and series, the day's increments	For good

Table	Content	Kept
counter_last	The last value of every counter, for an exact restart	minute_days
events	Every event; its seq is the event's sequence number	minute_days
metrics_meta	The session timezone the file was recorded under	—

The views v_samples_raw and v_samples_1m show the samples with their source, metric name and label spelled out, for SQL by hand:

```
SELECT ts, value FROM v_samples_raw
WHERE source LIKE 'pm-engine@%' AND name = 'proc.rss_bytes'
ORDER BY ts DESC LIMIT 10;
```

Sources are process keys (pm-api-gwy/desk@host) or host. A labelled series — one per participant, symbol, channel or core — is kept by the minute only. Of the engine's, only those the dashboard reads are kept (orders, rejects and the other request counters, fills, trades and traded value, resting orders, and the books' resting orders, memory and heap bloat, plus engine.rejects_by_code and engine.diagnostics); daily_totals keeps every counter. Besides each process's own metrics, pm-monitor stores every process's vitals as proc.rss_bytes, proc.cpu_pct, proc.threads, proc.fds, proc.loop_age_ms and proc.loop_lag_ms, and the host as host.* and files.size_bytes (labelled metrics.db, log.db, stats.db, audit, clearing.db, gtc).

Retention runs hourly and returns freed pages, so the file stops growing once it holds its window. Measured on the AI-trader simulation (300 symbols, 112 participants, some 3 000 orders a minute) it grows about 0.2 MB a minute while trading, which comes to about 0.75 GB at the default retention with its seven trading hours a day; a 30-symbol exchange holds a fraction of that. Most of it is the minute rows of the per-symbol and per-participant series; a shorter retention.minute_days shrinks it in proportion. pm-opctl-cli clear --all deletes it (it is history, like log.db), together with its metrics.db-wal and -shm; clear --state keeps it.

34.7 Configuration

The optional top-level monitor block of engine_config.yaml; an absent block means all defaults, with monitoring on. Every process reads it — the beaconing processes for where and how often to send, pm-monitor for everything.

```
monitor:
  enabled: true
  bind_address: 127.0.0.1
  host: 127.0.0.1
  pull_port: 5610
  pub_port: 5611
  beacon_interval_sec: 2
  hung_after_sec: 5
  silent_after_beacons: 3
  book_metrics_interval_sec: 10
  retention:
    raw_hours: 6
    minute_days: 7
  thresholds:
    host_cpu_pct: {warn: 80, crit: 95}
    disk_free_pct: {warn: 15, crit: 5}
```

Field	Default	Description
enabled	true	When false no process sends beacons and pm-monitor idles

Field	Default	Description
bind_address	127.0.0.1	Interface pm-monitor binds; EDUMATCHER_GATEWAY_BIND_HOST overrides it
host	127.0.0.1	Where every process sends its beacons, and where pm-monitor-cli and the dashboard connect. Must reach bind_address (M031)
pull_port	5610	Beacons in
pub_port	5611	Snapshots and events out
beacon_interval_sec	2	How often each process reports
hung_after_sec	5	Loop age above which a process is HUNG; must exceed the beacon interval (M030)
silent_after_beacons	3	Missed beacons before SILENT or DEAD
book_metrics_interval_sec	10	How often the engine recomputes its per-book figures
retention.raw_hours	6	Raw sample history
retention.minute_days	7	1-minute buckets, events and counter state
thresholds.<rule>	see Thresholds	{warn, crit}; warn below crit, except disk_free_pct (M029). Omit a rule to keep its default

pm-config-gen --monitor writes the whole block, and the --monitor-* options set its fields (see "Monitor options" in Generate Configs with pm-config-gen). pm-cverifier checks the block's shape (S124–S131), its cross-field rules (M029–M031) and its ports against every other listener (M018); see Config Verifier. Changing locale.session_timezone on an exchange with history makes pm-monitor refuse to start (see above).

34.8 pm-monitor-cli

A read-only query tool: it never writes metrics.db and holds no read open, so it never disturbs pm-monitor.

```
pm-monitor-cli [--host HOST] [--port PORT] [--db PATH]
               [--format table|json] [--timeout SEC] COMMAND [options]
```

Option	Default	Description
--host	monitor.host	Where pm-monitor publishes
--port	monitor.pub_port (5611)	Its snapshot/event port
--db	<data dir>/metrics.db	For events and health --extended
--format	table	json for scripts
--timeout	3	Seconds to wait for a snapshot

34.8.1 health

```
pm-monitor-cli health [-q] [--extended] [-v]
```

Healthy when a snapshot arrives within --timeout. Snapshots come from pm-monitor's main loop, so this proves the loop runs — the same guarantee the recorders' health pings give, without a port of its own.

- `-q` prints nothing; exit 0 when healthy, 255 otherwise (what `pm-opctl-cli` uses).
- `--extended` also checks `metrics.db`: integrity (`PRAGMA quick_check`), that it is a `pm-monitor` database, that its pinned session timezone matches the deployed one (`pm-monitor` would refuse to start otherwise), and that the file and its directory are writable.
- `-v` adds the `pm-monitor` PIDs and endpoint, and with `--extended` the database size, the size of its WAL file, and row counts. A WAL that keeps growing means a reader is holding a read open and stopping checkpoints.

```
$ pm-monitor-cli health --extended -v
pm-monitor snapshot: OK - snapshot seq 1520 at 2026-10-19T14:33:22.114Z, 16 processes, exchange OPERATIONAL
metrics.db:          OK - integrity, schema, timezone and write access ok
pm-monitor processes (pgrep): 258541
endpoint: tcp://127.0.0.1:5611
database: /home/me/.local/share/edumatcher/metrics.db
  size: 41.2MB (+ WAL 3.9MB)
  ...
health: OK
```

34.8.2 `status`

```
pm-monitor-cli status
```

One snapshot: the exchange status and session, today's orders and trades, the reject ratio and engine latency, every process (state, PID, uptime, RSS, CPU, loop lag, seconds since its last beacon), and active breaches. `--format json` prints the snapshot exactly as `pm-monitor` published it.

The exit code is the exchange status, for scripts:

Exit	Meaning
`0`	OPERATIONAL
`1`	DEGRADED
`2`	Usage error (as every `pm-*` CLI)
`3`	IMPAIRED
`4`	DOWN
`5`	`pm-monitor` unreachable

34.8.3 `events`

```
pm-monitor-cli events [--since ISO_TS] [--kind KIND] [--subject TEXT]
                      [--limit N] [--follow [--count N]]
```

The newest `--limit` (50) matching events from `metrics.db`, oldest first. `--kind` is one of the kinds above; `--subject` matches part of the subject. `--follow` then prints live events as they happen until Ctrl-C (`--count N`: stop after N). No event is lost or repeated between the history and the live feed: a gap in the sequence numbers is filled from `metrics.db`.

34.8.4 Cookbook

```
# Is the exchange OK? (exit code for a script or a cron job)
pm-monitor-cli status > /dev/null || echo "exchange not OPERATIONAL ($?)"

# What happened to the engine today?
pm-monitor-cli events --subject pm-engine --since 2026-10-19T00:00:00Z

# Watch disconnects live
pm-monitor-cli events --kind DISCONNECT --follow

# Every process state as JSON
pm-monitor-cli --format json status | jq '.processes[] | {key, state}'
```

34.9 Troubleshooting

Symptom	Likely cause	What to do
`health` fails: no snapshot	`pm-monitor` not running, another `--host`/`--port`, or monitoring disabled	`pm-opctl-cli list`; `pm-monitor-cli health -v`; check `monitor.enabled`
A process MISSING	The profile runs it but it never sent a beacon: it failed to start, or it sends to the wrong place	Its log in `<data dir>/emo/<name>.log`; check `monitor.host`/`pull_port`
Every process MISSING or SILENT, `pm-monitor` itself fine	Beacons cannot reach `pm-monitor`: `host` does not reach `bind_address`	`pm-cverifier` reports `M031`; align `host` and `bind_address`
A process SILENT	Its process exists but sends nothing: the whole interpreter is stuck	A Python stack dump (for example `py-spy dump --pid PID`, if installed) shows where; then restart it
A process HUNG	Its main loop is stuck while its beacon thread runs	Its log around the time; restart it
A process DEGRADED	A threshold on it, or its loop ran late	`status` lists the breach; `events --subject <key>`
`pm-monitor` exits: another `pm-monitor` writes `metrics.db`	Two started on one data directory	Stop the other one
`pm-monitor` exits: recorded with another timezone	`locale.session_timezone` changed	Restore it, or move `metrics.db` aside
Today's counts start from zero after a `pm-monitor` restart	First start with this `metrics.db` (or it was cleared)	Expected once; later restarts resume exactly
`pm-monitor` DEGRADED by `metrics_store_errors`	It cannot write `metrics.db`: a read-only or full data directory, or a broken file	Its log names the error; free space or fix the permissions, then restart `pm-monitor`
The exchange DEGRADED by `reject_pct_5m`	Many rejected requests	The dashboard's "Rejects today by reason" says why; `ORDER_NOT_FOUND` is mostly cancels of orders that had just filled, an ordinary race for an active trader

34.10 See also

- Operator Dashboard (`pm-dash`) — the same snapshot and history in a browser
- Running the Exchange — `pm-opctl-cli` profiles
- Centralized Log Server — where `pm-monitor`'s breach and state-change lines go
- Config Verifier — `S124`–`S131`, `M029`–`M031`
- Message Reference — `health.beacon`, `health.snapshot`, `health.event`
- SQLite Schemas — `metrics.db` in full

CHAPTER 35

Operator Dashboard (`pm-dash`)

Learning objectives

After reading this page you will understand:

- What the Operator Dashboard is for, and how it relates to `pm-monitor`
- How to start it with the rest of the exchange, on its own, or for development
- What each of the eight rows shows, and how to read its colours
- How the time picker, the refresh setting and the session clock work
- How to open a process, participant or order book in detail, and share that view
- How to configure it, and what to do when it shows nothing

35.1 Overview

The **Operator Dashboard** (`pm-dash`) is a browser view of the whole exchange, in the style of a Grafana dashboard: every process's health, trading activity, participants, order books, gateways and events, live and over time. It answers the question an operator asks first, from across the room:

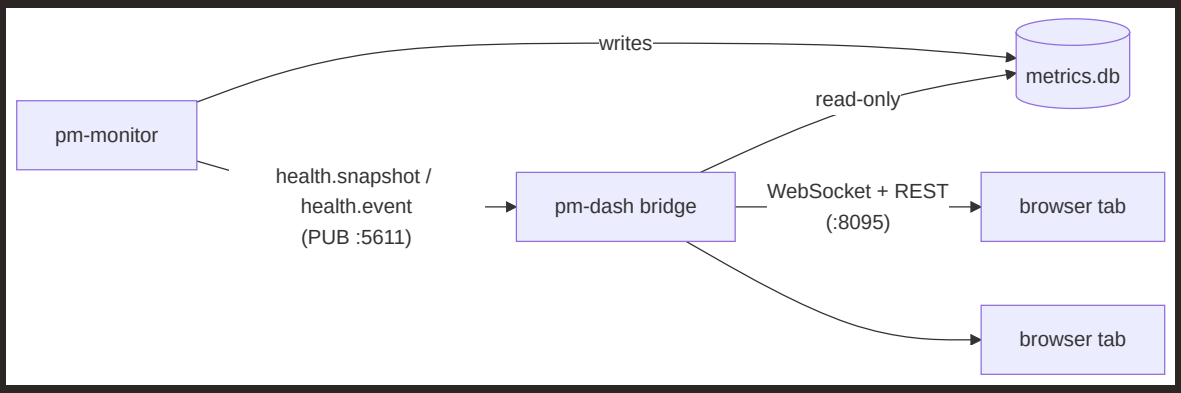
> **Is the exchange OK — and if not, where?**

It shows what `pm-monitor` knows. `pm-monitor` judges every process and the exchange, watches the thresholds and keeps the history in `metrics.db`; the dashboard draws its live snapshot (once a second) and that history. It judges nothing itself, so it never disagrees with `pm-monitor-cli status`.

The overview row says whether the exchange is OK at a glance: processes up, participants connected, today's orders and trades, the order rate and engine latency, the host's CPU, memory and disk, and the exchange's memory over the picked time range.

Component	Role	Type
`pm-monitor`	Judges health, computes rates, evaluates thresholds, publishes `health.snapshot` and `health.event` on port `5611`, writes `metrics.db`	Long-running process
`pm-dash` bridge	One subscription to `pm-monitor`, read-only `metrics.db` access, the WebSocket to every open tab, the history API	Long-running process
`pm-dash` web	The dashboard in the browser	Browser application


EduMatcher pm-dash v0.50.7
• OPERATIONAL
CONTINUOUS · 2026-10-20
OVERVIEW
PROCESSES
10 / 10
all healthy
PARTICIPANTS
0 / 4
connected
ORDERS TO
42
~0.70 /min
2.8%
host
0%
100%
CPU
8.9%
11.2 GiB
0%
100%
MEMORY
99.0%
62.1 GiB
0%
100%
DISK FREE
PROCESSES
STATE ▲
PROCESS
• HEALTHY pm-alf-gwy
• HEALTHY pm-api-gwy/desk
• HEALTHY pm-dc-gwy
• HEALTHY pm-engine
• HEALTHY pm-log-srv
• HEALTHY pm-md-gwy
• HEALTHY pm-monitor
• HEALTHY pm-ralf-gwy
• HEALTHY pm-scheduler



The dashboard is **strictly read-only**. It never writes `metrics.db`, sends nothing to `pm-monitor` or any other process, and can be stopped at any time without affecting the exchange. Everything it shows is also available from `pm-monitor-cli`.

35.2 Running the dashboard

The dashboard needs `pm-monitor`, which every built-in `pm-opctl-cli` profile except `micro` starts (see Starting `pm-monitor`).

35.2.1 Option 1 — The whole stack (recommended)

The dashboard starts with the exchange and the other web GUIs, and opens on http://localhost:8095:

- **A released install** (no checkout, no toolchain): the one-command install in Installation starts it; `./edumatcher.sh urls` lists it as "Operator Dashboard".
- **From a source checkout**: `make up-all` in `deployment/docker/` (see Container Deployment Walkthrough).

In both, the dashboard reaches `pm-monitor` over the container network and reads `metrics.db` and the deployed configuration from the exchange's data directory, mounted read-only. There is nothing to configure.

35.2.2 Option 2 — This app alone, in a container

When the exchange runs on this machine outside a container:

```
cd web-apps/dash-gui
export EDUMATCHER_DATA_DIR=~/.local/share/edumatcher   # where metrics.db lives
make up          # builds the image and starts it on http://localhost:8095
make logs        # follow the bridge's output
make down
```

The data directory is mounted read-only. `pm-monitor` binds `127.0.0.1` by default, which a container cannot reach; start the exchange with `EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0` (or set `monitor.bind_address`), and set `MONITOR_HOST` if `pm-monitor` runs on another machine.

35.2.3 Option 3 — Local development

```
cd web-apps/dash-gui
make install
make dev         # bridge on 127.0.0.1:5195, Vite on http://localhost:8195
```

Against the containerised exchange, publish `pm-monitor`'s port and take the bridge's environment from the stack:

```
cd deployment/docker
make up-all ZMQ=1                       # also publishes 5611 on the host
eval "$(make -s dev-env GUI=dash-gui)"
cd ../../web-apps/dash-gui && make dev
```

35.3 Reading the dashboard

35.3.1 The top bar

From left to right: the version, the **exchange status** (OPERATIONAL, DEGRADED, IMPAIRED or DOWN — `pm-monitor`'s verdict, see How health is judged), the **session indicator**, then the **time range**, the **refresh** setting, the theme, the settings, and the connection: **LIVE** while the browser is connected to the bridge.

The session indicator is the one every EduMatcher GUI shows, as in the Trading GUI: the session phase as a coloured badge, the next phase and the time left to it (`Continuous → Closing Auction in 03:08`), and the clock in the exchange session timezone. Hover it for the trading date. The bridge reads the phase from `pm-md-gwy` (CALF, `CALF_HOST`/`CALF_PORT`), not from `pm-monitor`; the badge reads `Unknown` until `pm-md-gwy` has reported it, and whenever it cannot be reached.

When the figures stop being live, a red banner says why and since when — "pm-monitor down — data frozen at 11:42:05", or the bridge is unreachable and the browser is reconnecting — and the panels dim. The last figures stay on screen; they are not cleared. The footer always shows whether the bridge receives `pm-monitor`'s snapshots.

35.3.2 Time

- **The time range** (`Last 5m` to `Last 7d`) sets the window of every chart. It is in the address bar as `?from=now-1h`, so a link opens on the same window.
- **Live figures** — every tile, gauge and table — arrive once a second, whatever the time range says.
- **Refresh** (off, 10 s — the default — 30 s, 1 min, 5 min) is how often the charts ask for their history again. 10 s is the finest history there is.
- **Every time shown is on the exchange's session clock** (`locale.session_timezone`), not your browser's: an operator in another timezone sees the same times as the trading date and the logs.

Hovering a chart draws the same vertical line on every chart, so you can line up a CPU spike with a burst of orders. Process state changes, threshold breaches, session changes, halts and resumes are thin coloured markers on every chart; hover one for what happened.

35.3.3 Colours

Every colour is **`pm-monitor`'s verdict**: a tile, gauge or table cell turns amber (WARN) or red (CRIT) when `pm-monitor` reports that rule in breach, and the processes tile follows the exchange status. The coloured bands drawn on gauges and charts are the rules themselves, as configured in `monitor.thresholds`; hover a tile or gauge for its definition.

The two can differ for a moment, on purpose: `pm-monitor` raises a breach at once but clears it only after three readings below the threshold, and judges some rules only above a minimum (the reject ratio from 20 requests, heap garbage from a minimum book size). A value inside a band with no colour is a value `pm-monitor` does not consider a breach — yet. See Thresholds and events.

Every state is also shown in words and a glyph (HEALTHY, DEGRADED, HUNG, DEAD …), never by colour alone.

35.3.4 Rates and their windows

An order or trade rate is always one of four, and each tile or column says which: *now* (the last 60 seconds), *avg / min* (the last 60 minutes), *avg / hour* (today's total over the hours the session has been open) and *avg / day* (the last 7 trading days). They answer different questions, so "0.70 /min" beside "3.78 /h" is not a contradiction: the last hour was quieter than the day.

35.3.5 The nine rows

Each row folds away with its title; the browser remembers which are folded.

Row	What it shows
Overview	Processes up (and which are not), participants connected, orders and trades today, orders per second and engine p99 latency (order entry) with their last minute; the host's CPU, memory and free disk; the exchange's memory by process over the range
Session schedule	The same timetable as TapeDeck's Session view: this week's hours with bank holidays, today's phases with the current one lit and the exchange's time on the day, and the next bank holiday that closes the exchange or shortens its hours. Read from `pm-api-gwy` with the read-only key (`PM_DASH_API_KEY`); without one the row says the timetable is unavailable
Engine latency	How long the matching engine takes per message over the last minute: order p50 and p99 with the number of samples behind them, the p99 with garbage-collection time taken out, the p99 of every other message, the longest GC pause and the share of time in GC. Below, the order latency and the longest GC pause over the range. See Reading engine latency
Processes	Every process: state, uptime, memory and CPU with their recent trend, loop age and lag. Participant bots are listed after the exchange's own and never affect its status. Memory per process, and CPU per process over the range
Trading activity	Orders per minute by participant (the top eight and "other"), trades and traded value per minute, the order averages, the reject ratio as `pm-monitor` judges it, and today's rejects by reason — what the engine said no to (`ORDER_NOT_FOUND`, `MARKET_CLOSED`, `MM_MAX_SPREAD` …)
Participants	How many participants were connected over the range, with heartbeat timeouts as red lines: a drop of many at once points at a gateway or the network. Then the table: role, connection, orders today and their rates, reject %, resting orders, connects / disconnects / heartbeat timeouts today, and the participant's own connection lane over the range (sort that column to bring the most disconnected to the top). "avg/day" shows "—" until there is a full day of history
Order books	The number of books and their estimated memory; per symbol: resting and stop orders, price levels, memory, heap bloat, trades today, and whether it is halted
Gateways & infrastructure	Clients per gateway against its limit, API WebSocket sessions and dropped events, slow-client disconnects, log server intake, feed gaps, the data files' sizes, and the engine's internal diagnostics
Event log	Every health event, newest first, filterable by kind: connects and disconnects, state changes, threshold breaches, session changes, halts

35.3.6 Reading engine latency

The engine times **1 message in 16** and reports percentiles over the **last minute**, at most 1024 timed messages. Three things keep the figures honest:

- **Order entry is timed apart from everything else.** New orders, cancels, amends, quotes, combos and OCOs are

order latency. Heartbeats, connects, queries and admin commands are *other messages*: a book snapshot of a deep book can take milliseconds, and mixed in it would make the order p99 look like matching is slow.
- **The p99 shows how many samples it stands on.** Below 100 samples a p99 is simply the slowest of them; the row header then says so. At a few orders a minute this is normal.
- **Garbage collection is timed separately.** Every collection is timed, not only the sampled messages, so *Longest GC pause* never misses a stall. *p99 without GC* is the order p99 with the collection time inside each message taken out: when it sits far below the order p99, the slow messages were slow because Python was collecting garbage, not because of matching.

The maximum is the slowest *timed* message, so it can miss outliers that were not sampled; the longest GC pause and the process's loop lag cover the worst stalls.

Per-minute bars count a minute once it is over, so the current minute appears at the next refresh after it closes.

35.3.7 Drawers: one process, participant or book

Click a row in the processes, participants or order books table (or select it and press Enter) to open its **drawer** on the right:

- **A process**: its state and figures, its memory, CPU, threads, file descriptors, loop age and loop lag over the range, every application metric it reports (for the engine, those not per participant or symbol — those are in the participant and book drawers), its events, and **Logs in log-gui** — its log records in the Log Operator Console.
- **A participant**: its connection and figures, its orders, rejects and fills per minute, its resting orders, and its connects and disconnects.
- **An order book**: its figures, trades and traded value per minute, resting orders, memory and heap bloat, its halts and resumes, and **Book in book-gui** — the book in the Order Book Viewer.

Escape or the close button closes it; the dashboard stays live behind it.

Each drawer has its own address — `/process/<key>`, `/participant/<id>`, `/book/<symbol>`, with the time range — so in an incident you can paste the link to exactly what you are looking at, for example `http://localhost:8095/process/pm-engine%40edumatcher?`

35.3.8 Settings

The gear icon holds the theme (dark or light), the font size (for a wall display across the room), and whether a stat tile shows its level by colouring the value or its whole background. All three are remembered per browser.

35.4 Configuration reference

The bridge is configured from its environment. In both container stacks these are already set; you need them only for Option 2 or 3.

Variable	Default	Meaning
`HOST` / `PORT`	`127.0.0.1` / `5195`	Bridge bind address (the container uses `0.0.0.0:8095`)
`MONITOR_HOST`	`127.0.0.1`	Where `pm-monitor`'s PUB socket is reachable (`monitor.host`)
`MONITOR_PUB_PORT`	`5611`	`monitor.pub_port`
`MONITOR_DOWN_AFTER_SEC`	`3`	No snapshot for this long and `pm-monitor` is shown down
`CALF_HOST` / `CALF_PORT`	`127.0.0.1` / `5570`	`pm-md-gwy`, for the session phase in the top bar
`API_GATEWAY_URL`	`http://127.0.0.1:8081`	`pm-api-gwy`'s dashboards instance, for the Session schedule row
`PM_DASH_API_KEY`	unset	The read-only API key (the credential with `gateway_id: null`); the stacks resolve it at start

Variable	Default	Meaning
METRICS_DB_PATH	<data dir>/metrics.db	pm-monitor's history, opened read-only
CONFIG_ARTIFACT_PATH	<data dir>/ref_data/engine_config.json	The deployed configuration, for the threshold bands
LOG_GUI_URL / BOOK_GUI_URL	unset (http://localhost:8091 / :8094 in the stacks)	log-gui's and book-gui's addresses as your browser reaches them; set, the drawers link there
CORS_ORIGIN	*	Restrict when exposed beyond localhost
MAX_WS_CLIENTS	200	Browser tabs beyond this are refused
LOG_SRV_HOST / LOG_SRV_PORT	127.0.0.1 / 5600	The bridge's own logging to pm-log-srv (optional)

In the container stacks, DASH_GUI_PORT in .env moves the dashboard off 8095, and LOG_GUI_URL / BOOK_GUI_URL there change the drawers' links — needed when you open the GUIs from another machine.

35.5 Security notes

- The dashboard has **no login**. It shows no credentials and changes nothing, but it does show the exchange's participants and activity.
- The stacks publish it on 127.0.0.1 only. Setting BIND_ADDR=0.0.0.0 opens it — with every other GUI — to your network; do that only on a network you control, and set CORS_ORIGIN to the dashboard's own address.
- The bridge holds the read-only API key for the Session schedule row; it never sends it to the browser.
- pm-monitor's ports are not published by the stacks; the dashboard reaches it over the container network. ZMQ=1 (or EM_ZMQ=1) publishes the snapshot feed (5611) for tools on the host, never the beacon intake (5610), which would let anything report fake health.

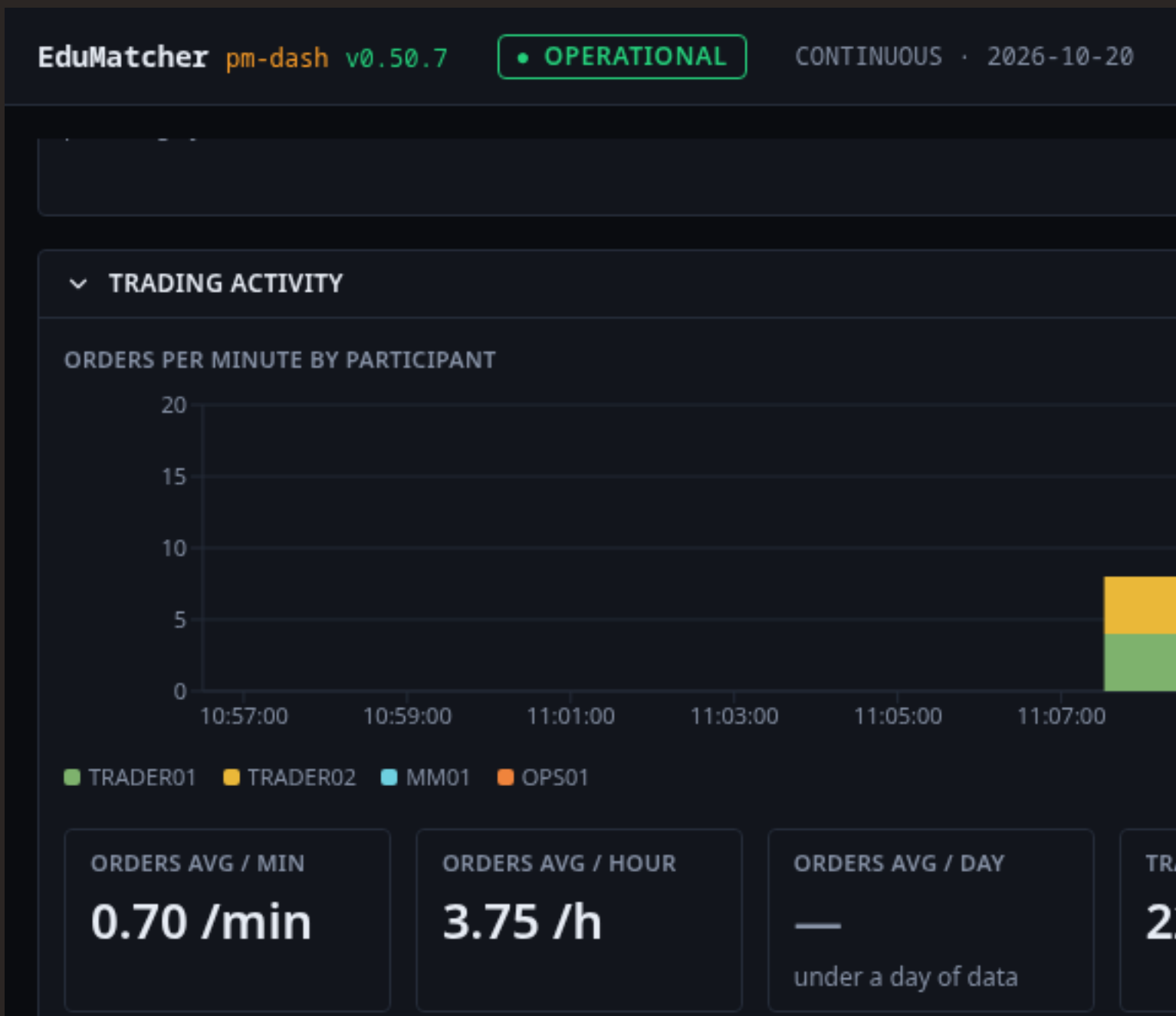
35.6 Troubleshooting

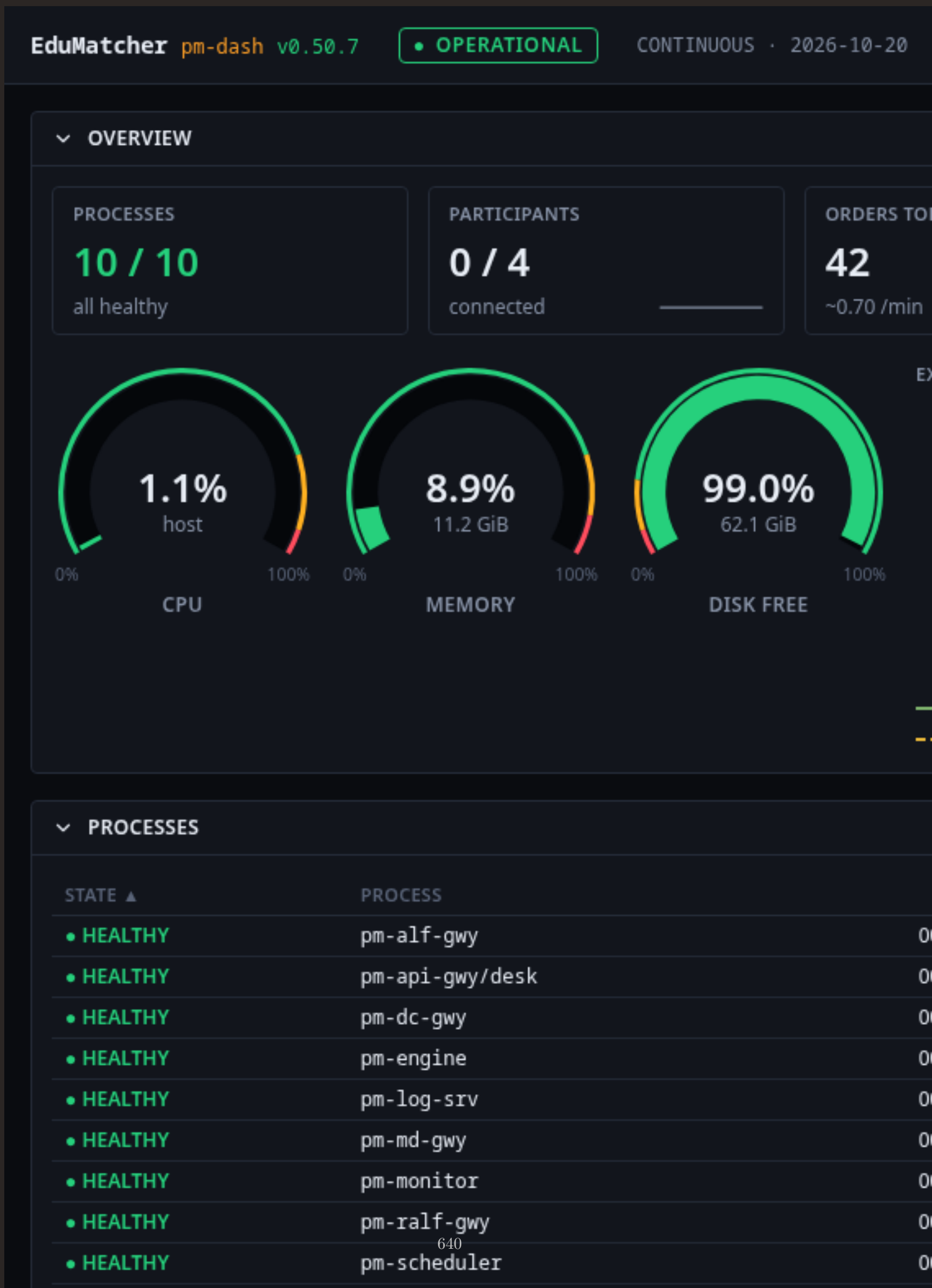
Symptom	Likely cause	What to do
Banner "pm-monitor down" and nothing ever shown	pm-monitor not running, or not reachable at MONITOR_HOST:MONITOR_PUB_PORT	pm-opctl-cli list; pm-monitor-cli status; outside a container, open it with EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0
Banner "pm-monitor down" after it worked	pm-monitor stopped or hung	pm-opctl-cli list; restart it. The dashboard recovers by itself
Live figures, but every chart says "No data in this range"	The bridge cannot read metrics.db, or the exchange has only just started (the first history row is written after 10 s)	curl localhost:8095/api/bridge/status shows metricsDb; check METRICS_DB_PATH and the mount
Per-minute bars stay empty	The minute is not over yet, or refresh is off	Wait for the minute to close, or set refresh to 10 s
Session schedule row says "Timetable unavailable"	No read-only key, or pm-api-gwy not reachable at API_GATEWAY_URL	curl localhost:8095/api/reference/schedule shows the gateway's answer; set PM_DASH_API_KEY (the stacks do it for you)
Bands differ from monitor.thresholds	The bridge does not see the deployed configuration	Check CONFIG_ARTIFACT_PATH; curl localhost:8095/api/thresholds says "source": "artifact" when it does

Symptom	Likely cause	What to do
Changed `monitor.thresholds`, colours did not follow	`pm-monitor` reads its thresholds at start	Restart `pm-monitor`. The bands show the new values within one refresh interval
No "Logs in log-gui" / "Book in book-gui" link	`LOG_GUI_URL` / `BOOK_GUI_URL` unset or not `http(s)://…`	Set them as your browser reaches those GUIs
A pasted link shows "No process … in the latest snapshot"	That process, participant or book is not in this exchange (another host name, a stopped bot)	Open the dashboard and pick it from its table
`pm-monitor` DEGRADED, a `metrics_store_errors` breach	`pm-monitor` cannot write its history (read-only or full data directory)	See Health Monitoring; the live figures are unaffected
Container cannot read `metrics.db` on Linux	An SELinux-enforcing host and a mount without `:z`	The supplied compose files carry `:ro,z`; keep it in your own

35.7 Where to go next

- Health Monitoring (`pm-monitor`) — the health states, thresholds, rates and `metrics.db` the dashboard draws
- Log Operator Console (`pm-log-ui`) — what a process logged
- Order Book Viewer (`pm-book`) — one book in full depth
- Running the Exchange — `pm-opctl-cli` profiles


EduMatcher pm-dash v0.50.7
• OPERATIONAL
CONTINUOUS · 2026-10-20
TRADING ACTIVITY
ORDERS PER MINUTE BY PARTICIPANT
20
15
10
5
0
10:57:00
10:59:00
11:01:00
11:03:00
11:05:00
11:07:00
TRADER01
TRADER02
MM01
OPS01
ORDERS AVG / MIN
0.70 /min
ORDERS AVG / HOUR
3.75 /h
ORDERS AVG / DAY
—
under a day of data
PARTICIPANTS · 0 / 4 connected
MM01
OPS01
TRADER01
TRADER02
connected
heartbeat timeout


EduMatcher pm-dash v0.50.7
• OPERATIONAL
CONTINUOUS · 2026-10-20
OVERVIEW
PROCESSES
10 / 10
all healthy
PARTICIPANTS
0 / 4
connected
ORDERS TO
42
~0.70 /min
1.1%
host
0%
100%
CPU
8.9%
11.2 GiB
0%
100%
MEMORY
99.0%
62.1 GiB
0%
100%
DISK FREE
PROCESSES
STATE ▲
PROCESS
• HEALTHY pm-alf-gwy
• HEALTHY pm-api-gwy/desk
• HEALTHY pm-dc-gwy
• HEALTHY pm-engine
• HEALTHY pm-log-srv
• HEALTHY pm-md-gwy
• HEALTHY pm-monitor
• HEALTHY pm-ralf-gwy
• HEALTHY pm-scheduler

CHAPTER 36

FAQ: Setup and Operation

Questions about running an exchange beyond the first start: seed quotes, the scheduler, watching the message bus, participant allowlists, and where the configuration and data files live. The first-day questions are in the Quick Start Guide's FAQ & Common Mistakes; questions about orders and positions are in the Participant Guide's FAQ: Orders, Sessions and Positions.

36.1 Starting the exchange

36.1.1 Why did the seed MM quote disappear after the second day?

By default, `market_maker_quotes` entries have `seed_once: true`. This means the seed is skipped whenever that market maker already has a live quote on the symbol that was restored from the previous run. A quote that was traded away or that expired is not restored, so its seed fires again; otherwise the real market maker is expected to keep submitting its own quotes.

To inject the seed on every startup (e.g. for a repeatable classroom demo):

```
market_maker_quotes:
  - gateway_id: MM01
    bid_price: 149.00
    ask_price: 151.00
    bid_qty: 500
    ask_qty: 500
    tif: DAY
    seed_once: false    # inject on every startup
```

To force a day-1 re-injection without changing the config:

```
rm data/book_stats.json    # or $EDUMATCHER_DATA_DIR/book_stats.json
```

36.1.2 Why did `pm-scheduler` exit immediately?

Because no configuration is deployed. The scheduler exits with a fatal error rather than falling back to its built-in default times — running a timetable the engine has never seen is worse than not starting at all.

```
pm-config-deploy engine_config.yaml
```

36.2 Seeing what is happening

36.2.1 How do I see what messages are flowing between processes?

Run `pm-audit --terminal` in a separate window. It subscribes to **all** ZeroMQ messages and prints them to stdout, giving a full real-time trace of every event in the system.

36.3 Participants

36.3.1 Why is my order rejected with "Gateway not authenticated"?

Your gateway ID is not in the engine's allowlist. Gateway IDs must be pre-configured in `engine_config.yaml` under `participants`:

```
participants:
    - id: GW01
      role: TRADER
      description: My first trader
```

Deploy the configuration (`pm-config-deploy`) and restart the engine after editing it.

36.4 Configuration and data files

36.4.1 The engine says it cannot find its configuration. Where should the file be?

In exactly one place, always:

```
<EDUMATCHER_DATA_DIR>/ref_data/engine_config.json
```

Mode	Data directory
pipx installed	`~/.local/share/edumatcher`
Source checkout	`<repo>/src/data/`
Either	`EDUMATCHER_DATA_DIR`, if set

The message means nothing has been deployed there yet. Install a configuration and the error goes away:

```
pm-config-deploy ~/my-session/engine_config.yaml
pm-config-deploy --show        # print the path it deployed to
```

There is no `--config` flag and no `EDUMATCHER_CONFIG` variable. Both used to exist, and the reason they no longer do is that they let one process be started against a configuration the rest of the exchange had never seen — the resulting symptom (a gateway reporting no instruments while the engine happily traded ten) gave no hint of the cause.

36.4.2 Where are my data files stored?

Data files (`gtc_orders.json`, `stats.db`, `audit.log`, etc.) go to:

Mode	Data directory
pipx installed	`~/.local/share/edumatcher`
Source checkout	`<repo>/src/data/`
Either	`EDUMATCHER_DATA_DIR` environment variable (highest priority)

To isolate two sessions from each other, run each in its own directory and set `EDUMATCHER_DATA_DIR` to a different path:

```
export EDUMATCHER_DATA_DIR=~/sessions/morning
pm-engine
```