AI001 BUY · AI017 SELL · AI042 BUY · MM01 QUOTE · AI063 SELL · NEWS +4.0% · AI088 BUY · RUMOUR −2.5%




AUTONOMOUS MARKET SIMULATION

# AI TRADERS AND MARKET SIMULATION

*Run an exchange that trades itself: AI traders, a hidden true value, news and rumours*

AGENT SWARM · 100 TRADERS · 300 SYMBOLS

BUY SELL WAIT



**EduMatcher AI Traders and Market Simulation**

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.

# AI Traders and Market Simulation

`edumatcher`, Version: v0.77.0

Run an autonomous exchange: AI traders, a hidden true value, news and rumours

Johan Persson <johan162@gmail.com>

Generated on November 8, 2026

# Chapters


1 How to Use This Book 1

2 A Market That Runs Itself 5

3 Ten-Minute Start 9

4 The True Value 13

5 News and Rumours 19

6 How the AI Traders Decide 27

7 Tuning the Traders 33

8 AI Traders 43

9 Running the AI-Trader Swarm 49

10 The Market Model (`pm-market-sim`) 55

11 A Complete Set-up by Hand 61

12 The Simulation Container 67

13 Accelerated Time 75

14 Watching the Market 79

15 AI Traders & News 83

**16 Running It for Weeks** **89**

**17 Troubleshooting and FAQ** **95**

**18 Simulation Glossary** **99**

# Contents


**1 How to Use This Book** **1**
- 1.1 What you will be able to do . . . 1
- 1.2 How the book is organized . . . 2
- 1.3 The set-up this book runs . . . 2
- 1.4 Conventions . . . 3
- 1.5 Where this book sits . . . 3

**2 A Market That Runs Itself** **5**
- 2.1 An exchange with nobody at the desks . . . 5
- 2.2 The idea underneath: a value nobody can see . . . 5
- 2.3 The parts . . . 6
- 2.4 The twelve kinds of trader . . . 7
- 2.5 What it is good for . . . 8
- 2.6 What it is not . . . 8
- 2.7 See also . . . 8

**3 Ten-Minute Start** **9**
- 3.1 Install and start it . . . 9
- 3.2 Open the browser applications . . . 10
- 3.3 Look at the hidden value . . . 11
- 3.4 See a trading day now . . . 11
- 3.5 Stop it, start it again . . . 12
- 3.6 What you have now . . . 12
- 3.7 If something does not work . . . 12

**4 The True Value** **13**
- 4.1 Price and value . . . 13
- 4.2 How the values move . . . 13
- 4.3 Who pulls, who pushes . . . 15
- 4.4 How close does the market get? . . . 15
- 4.5 Changing the model . . . 16
- 4.6 The market maker's anchor . . . 17
- 4.7 See also . . . 18

**5 News and Rumours** **19**
5.1 Why news . . . 19
5.2 Anatomy of a headline . . . 19
5.3 Generated news . . . 20
5.4 Rumours . . . 23
5.5 Who reacts, and how . . . 23
5.6 Steering the news . . . 24
5.7 Try it . . . 26
5.8 See also . . . 26

**6 How the AI Traders Decide** **27**
6.1 The loop . . . 27
6.2 Strategies: what to trade . . . 28
6.3 Execution styles: how to trade it . . . 28
6.4 Tempo: how often and how big . . . 29
6.5 Risk: what it will not do . . . 30
6.6 Writing a preset . . . 30
6.7 The other bot: `pm-mm-bot` . . . 31

**7 Tuning the Traders** **33**
7.1 Where the settings live . . . 33
7.2 What to watch . . . 34
7.3 The swarm . . . 34
7.4 The presets . . . 36
7.5 The market maker . . . 39
7.6 The model . . . 40
7.7 Recipes . . . 40
7.8 Measuring a change . . . 41

**8 AI Traders** **43**
8.1 What are the AI trader processes? . . . 43
8.2 Presets . . . 44
8.3 Running one AI trader . . . 45
8.4 Running a swarm . . . 46
8.5 Your own preset . . . 47
8.6 Reading its output . . . 47
8.7 See also . . . 47

**9 Running the AI-Trader Swarm** **49**
9.1 What runs . . . 49
9.2 Before you start . . . 49
9.3 Starting it . . . 50
9.4 Logs and daily summaries . . . 52
9.5 When something goes down . . . 53
9.6 Stopping . . . 53
9.7 Compressed soak . . . 53

**10 The Market Model (`pm-market-sim`)** **55**
10.1 What it does . . . 55

10.2 The model . . . 56
10.3 `market_sim.yaml` . . . 56
10.4 Running it . . . 57
10.5 Anchoring a market maker . . . 57
10.6 News . . . 57

**11 A Complete Set-up by Hand** **61**
11.1 Install and start . . . 61
11.2 Accelerated time by hand . . . 62
11.3 What runs . . . 62
11.4 Where everything is . . . 63
11.5 Your own variant . . . 64
11.6 Running the pieces one at a time . . . 65

**12 The Simulation Container** **67**
12.1 Two images, one exchange . . . 67
12.2 What happens when it starts . . . 68
12.3 `compose.ai-sim.yaml` . . . 68
12.4 Changing what it runs . . . 71
12.5 Opening it to a classroom . . . 72
12.6 Resources . . . 72
12.7 Upgrading . . . 73
12.8 Building the image yourself . . . 73

**13 Accelerated Time** **75**
13.1 One setting . . . 75
13.2 Choosing a speed . . . 75
13.3 What speeds up, and what does not . . . 76
13.4 Traps . . . 77
13.5 A lesson plan at speed 10 . . . 77

**14 Watching the Market** **79**
14.1 The five windows . . . 79
14.2 The command line . . . 80
14.3 Who made money . . . 81
14.4 For the instructor: the values through the API . . . 82
14.5 See also . . . 82

**15 AI Traders & News** **83**
15.1 Objective . . . 83
15.2 Prerequisites . . . 83
15.3 Background . . . 83
15.4 Exercise 1: Deploy an Example with AI Traders . . . 84
15.5 Exercise 2: One AI Trader . . . 85
15.6 Exercise 3: The Swarm . . . 85
15.7 Exercise 4: Price and Value . . . 85
15.8 Exercise 5: An Earnings Surprise . . . 86
15.9 Exercise 6: A Rumour That Turns Out True . . . 86
15.10 Exercise 7: A Rumour That Is Retracted . . . 86

15.11 Exercise 8: A Breakthrough and a Fire . . . 87
15.12 Exercise 9: Stop the Swarm . . . 87
15.13 Classroom Tips . . . 87
15.14 Reflection . . . 88
15.15 Further Reading . . . 88

**16 Running It for Weeks** **89**
16.1 Before you leave it . . . 89
16.2 What the container repairs by itself . . . 89
16.3 What grows . . . 90
16.4 A minute a day . . . 91
16.5 The nightly backup . . . 91
16.6 Backups . . . 92
16.7 Upgrading during a run . . . 92
16.8 The evidence . . . 92

**17 Troubleshooting and FAQ** **95**
17.1 When something does not work . . . 95
17.2 FAQ . . . 96

**18 Simulation Glossary** **99**

CHAPTER 1

# How to Use This Book

This book is about an exchange that trades on its own. A hundred AI traders, a market maker and a hidden model of what every share is really worth keep 300 order books busy all day, every trading day, for as long as you leave it running. You watch it, teach with it, trade against it, and change how it behaves.

| | |
|---|---|
| **Who it is for** | Instructors who want a live market for a class; students who want a market to trade in; anyone curious how prices form |
| **What you need** | A computer with Docker or Podman for the container route, or Python 3.14 and `pipx` for the hand-built route; a terminal and a browser |
| **What you do not need** | Any finance background, any Python, or a copy of the source code |
| **Time** | Ten minutes to a running market (Part I); an afternoon to understand and tune it (Part II) |

## 1.1 What you will be able to do

After this book you can:

1. start a complete simulated exchange — 300 shares, 100 AI traders, a market maker, news — with one command, and leave it running for weeks;
2. explain how a price on the exchange can be *wrong*, what the true value behind it is, and how trading discovers it;
3. read the news and rumours the market model publishes, and publish your own to move the market in front of a class;
4. change how the traders behave — calmer, wilder, faster to react, more or less informed — and know which setting does what;
5. run the calendar faster than real time, so a month of trading passes in a day;
6. watch all of it with the browser applications and the command-line tools.

## 1.2 How the book is organized

| Part | Chapters | What you get |
|---|---|---|
| **I. Start here** | A Market That Runs Itself; Ten-Minute Start | What the simulation is, and a running one |
| **II. How it works** | The True Value; News and Rumours; How the AI Traders Decide; Tuning the Traders | The model behind the prices, the traders, and every setting that changes them |
| **III. Run it** | AI Traders; Running the AI-Trader Swarm; The Market Model; A Complete Set-up by Hand; The Simulation Container; Accelerated Time | The reference for each part, and two ways to run it all |
| **IV. Watch and teach** | Watching the Market; AI Traders & News; Running It for Weeks; Troubleshooting and FAQ | What to look at, classroom exercises, and keeping it healthy |
| **Appendices** | Simulation Glossary | The words this book uses |

Part I is enough to get a market running. Part II is the heart of the book: read it before you change anything. Part III is reference — dip into it when you need a flag or a file. Part IV is for the days and weeks after.

**Chapters shared with other books**

Five chapters of this book also appear in the book they were written for: *AI Traders* and *How the AI Traders Decide* in the Participant Guide, *Running the AI-Trader Swarm* and *The Market Model* in the Operator's Guide, and *AI Traders & News* in the Training Guide. They are the same text, kept in one place. Their links point into those books, and their examples use the smaller `s150-*` examples, started by hand. What they explain holds for the `s300-sim` set-up this book runs; their commands do not all fit the container. In the container, skip the steps that deploy an example (the container has deployed `s300-sim`, and the AI traders need it), and remove the market maker as Tuning the Traders shows rather than with Ctrl+C: the container would restart it. Scenario files are in the image under `/opt/edumatcher/.venv/lib/python3.14/site-packages/docs/examples/news/`.

## 1.3 The set-up this book runs

Everything here uses one bundled example, `s300-sim`, and one process profile, `ai-sim`:

| | |
|---|---|
| **Symbols** | 300 shares in 11 industry sectors |
| **AI traders** | `AI001`–`AI100`, twelve kinds (presets), each trading 30 symbols |
| **Market maker** | `MM01`, a bot quoting every symbol, nudged toward the true values |

| | |
|---|---|
| **People** | `TRADER01`–`TRADER10` (desks for students), `OPS01` (the instructor) |
| **Market model** | A hidden true value per share, with random news and rumours |
| **Hours** | 09:00–16:05, Monday to Friday, on the Swedish calendar and clock |

The container image `edumatcher-ai-sim` is exactly this set-up, ready to run.

## 1.4 Conventions

- A bare command in a `bash` block is typed in a normal terminal. Commands that run *inside* the simulation container say so, or start with `docker exec edumatcher-sim`. From an installed simulation (`cd ~/.edumatcher-sim`), `./edumatcher.sh shell <command>` is the same thing and works with Podman too; `./edumatcher.sh` also wraps `start`, `stop`, `status` and `logs`. Where the book says `./data-sim`, an installed simulation's data directory is `~/.edumatcher-sim/data`.
- `[OPS01|ADMIN]>` is the prompt of the operator console (`pm-admin`); `[TRADER01]>` the prompt of a trader console (`pm-alf-console`).
- Output shown in a plain block is what you should see. Prices, times and counts will differ on your machine: the market is random, and so is the moment you look at it.
- Every number this book quotes about the simulation — how far prices stray, how much disk a day takes, how much CPU it needs — was measured on the set-up it describes. Where a number depends on your machine, the book says on what it was measured.
- `!!! tip`, `!!! note` and `!!! warning` boxes hold advice you can skip on a first reading; **Checkpoint** paragraphs tell you how to know a step worked.

## 1.5 Where this book sits

This is one of nine EduMatcher books. If the exchange itself is new to you, read the Quick Start Guide first — an hour, and every chapter here will be easier. If markets are new to you, How an Exchange Works explains order books, auctions and market makers from scratch.

CHAPTER 2

# A Market That Runs Itself

**Learning objectives**

After reading this chapter you will be able to:

- Say in a few sentences what the simulation is and what it is for
- Name its parts and what each one does
- Explain the one idea the whole simulation rests on: a price that is trying to find a value nobody can see

## 2.1 An exchange with nobody at the desks

An exchange on its own is an empty room. Order books fill only when someone sends orders, and a class of students can keep a few symbols busy for an hour at most. The simulation fills the room: AI traders send orders into every one of 300 order books, all day, every trading day, whether anyone is watching or not. Students who log on find a market already moving — prices, spreads, queues, news — and trade in it next to the bots.

The traders are not scripted. Each one follows a simple rule of its own — buy what looks cheap, follow a trend, fade a move, react to a headline, or just trade at random — and the prices you see are what those rules add up to. Nobody decides that AAPL should trade at 182.40. It does because a value investor thought it was cheap at 182.30 and a noise trader happened to sell.

## 2.2 The idea underneath: a value nobody can see

Real shares have a value — what the company is actually worth — that nobody knows exactly. Prices wander around it: traders who have a good idea of the value buy when the price is below it and sell when it is above, and so pull the price toward it, while everyone else pushes it about. When news arrives the value jumps, and the price follows as fast as traders understand the news. That is *price discovery*.

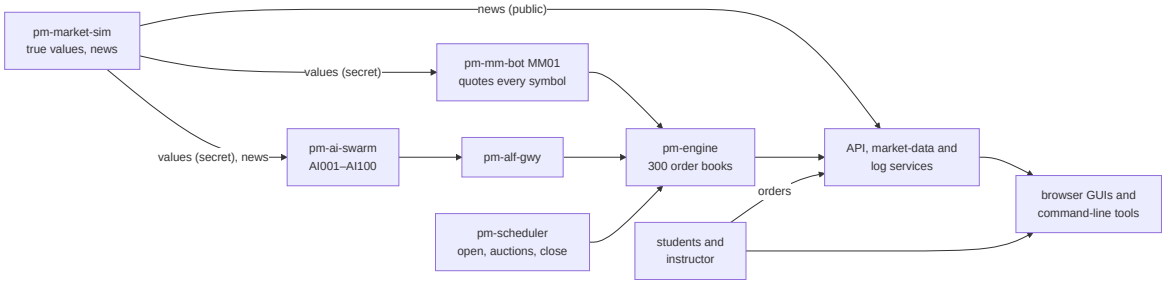
The simulation makes this concrete. A process called `pm-market-sim` keeps a **true value** for every share: a number that moves the way company values move, together with the market, with its sector, on its own, and in sudden jumps when news comes out. The true values are secret. Students never see them; only some of the AI traders, the market maker and the instructor do. The price on the exchange is whatever the traders

agree on, and the gap between price and value is what makes the market interesting:

- a well-informed market keeps the gap small, a noisy one lets it grow;
- a headline moves the value at once and the price a little later;
- a rumour moves the price but not the value — until it is confirmed or retracted.

The True Value explains the model; News and Rumours the news.

## 2.3 The parts




| Part | Process | What it does |
|---|---|---|
| The exchange | `pm-engine` and its gateways | Matches orders in 300 order books; publishes trades and prices |
| The clock | `pm-scheduler` | Opens and closes the market: pre-open at 09:00, opening auction, continuous trading 09:30–16:00, closing auction, closed at 16:05; weekends and Swedish holidays off |
| The market model | `pm-market-sim` | Keeps the hidden true values, moves them during trading, publishes news and rumours |
| The market maker | `pm-mm-bot` as `MM01` | Quotes a buy and a sell price in every symbol, around a price it moves part of the way toward the true value every second |
| The AI traders | `pm-ai-swarm` as `AI001`–`AI100` | A hundred traders of twelve kinds, each trading 30 of the 300 symbols |
| The windows | the GUIs and `pm-*` tools | Show the books, the trades, the news, the logs; let people trade |

Everything else — the audit trail, clearing, statistics, the API — is the ordinary exchange, running exactly as it does in a classroom without any bots. The AI traders are participants like any other: they log on, send

orders and receive fills through the same gateway a student's trading program uses. Their orders are in the books you look at, their trades in the tape.

## 2.4 The twelve kinds of trader

Each AI trader follows a *preset*: a strategy (what to trade), an execution style (how), a tempo (how often, how big) and risk limits. The hundred traders are a mix:

| Preset | How many | What it does |
|---|---|---|
| `noise-retail` | 15 | Small random orders resting near the best prices — the background flow |
| `value-investor` | 22 | Buys below its view of the true value, sells above it |
| `trend-follower` | 10 | Buys what is rising, sells what is falling |
| `contrarian` | 10 | The opposite: fades moves, patiently |
| `institutional` | 8 | Big orders toward the value, sliced over minutes |
| `scalper` | 8 | Fast momentum trades with tight trailing stops |
| `news-trader` | 5 | Trades headlines within seconds |
| `iceberg-seller` | 4 | Large orders showing only a fifth of their size |
| `market-taker` | 4 | Impatient: takes liquidity with market orders |
| `block-taker` | 2 | Takes whole blocks with fill-or-kill orders |
| `auction-player` | 2 | Trades mostly in the opening and closing auctions |
| `patient-liquidity` | 10 | Small orders waiting deep in the book — the depth behind the best prices |

Only the value, institutional and news traders know anything about the true value or the news. Everyone else trades on prices alone — and that is what keeps the market from being perfectly efficient. How the AI Traders Decide has the details.

## 2.5 What it is good for

- **A live market for a class.** Students log on as `TRADER01`–`TRADER10` and trade in books that already have depth, a spread and a tape.
- **Teaching price discovery.** Inject a headline in front of the class and watch the price find the new value; inject a rumour and watch it move a price that should not move.
- **Teaching microstructure.** Auctions, queues, icebergs, stops, market orders sweeping a book — all of them happen every day, unprompted.
- **A long-running test bed.** Leave it for a month and use the history: daily statistics, the audit trail, the traders' daily profit and loss.

## 2.6 What it is not

The simulation is a teaching market, not a forecast of a real one. Its values follow a textbook model, its traders a handful of simple rules, and its hundred participants are far fewer than a real exchange's. The prices look and behave like prices — they trend, revert, jump on news, and have the fat-tailed returns of real prices — but no real share is behind them.

## 2.7 See also

- Ten-Minute Start — get it running
- The True Value — the model behind the prices
- AI Traders — the trader processes and presets

CHAPTER 3

# Ten-Minute Start

**Learning objectives**

After this chapter you will have:

- A simulated exchange running, with its AI traders, market maker and news
- The five browser applications open on it
- Checked that it is healthy, and know how to stop and restart it

**Prerequisites**: Docker (Docker Desktop on macOS and Windows) or Podman with `podman compose`; about 2 GB of free memory and 4 GB of free disk (for a month's running).

There are two ways to run the simulation. The **container** is the one this chapter uses: one command, the browser applications included. The **hand-built** way runs the same processes straight on your machine with `pipx`; it is in A Complete Set-up by Hand.

## 3.1 Install and start it

One command downloads the installer, which fetches the compose file and the control script, pulls six images and starts everything:

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

Everything lands in `~/.edumatcher-sim` — a compose file, a `.env`, the control script `edumatcher.sh`, and `./data` where the exchange writes. There is nothing installed anywhere else; delete the directory and it is gone.

The first start downloads six images. Then:

```
 ✔ Container edumatcher-sim               Started
 ✔ Container edumatcher-sim-terminal-gui  Started
 ✔ Container edumatcher-sim-book-gui      Started
 ✔ Container edumatcher-sim-dash-gui      Started
 ✔ Container edumatcher-sim-log-gui       Started
 ✔ Container edumatcher-sim-trader-gui    Started
```

The exchange deploys its configuration and starts its processes; give it half a minute.

**Podman, or a different directory**

The installer uses Podman when it is installed and Docker otherwise. `--dir PATH` puts the install somewhere else, `--version X.Y.Z` pins a release, `--no-start` downloads everything, images included, but stops short of starting it (`./edumatcher.sh start` then needs no network).

**From a clone of the repository**

`deployment/docker/compose.ai-sim.yaml` is the same stack built from source: `make up-sim` starts it, `make down-sim` stops it.

**Checkpoint.** Ask the exchange for its process table:

```
./edumatcher.sh status
```

Every row should say `running` (a gateway may say `not responding` for the first few seconds while it binds its port). The last three rows are the simulation: `mm-bot`, `market-sim` and `ai-swarm`.

**Is the market open?**

The simulation keeps real trading hours: 09:00–16:05 Stockholm time, Monday to Friday, closed on Swedish public holidays. Outside those hours everything runs but nothing trades; the books show the last day's closing state. To see trading right now, whatever the time, use See a trading day now.

## 3.2 Open the browser applications

| Application | Address | Shows |
|---|---|---|
| **terminal-gui** | http://localhost:8090 | A market-data terminal: watchlists, quotes, charts, the news |
| **book-gui** | http://localhost:8094 | One symbol's order book, level by level, live |
| **log-gui** | http://localhost:8091 | The log of every process, searchable |
| **trader-gui** | http://localhost:8093 | A trading screen — log in as a student desk |
| **dash-gui** | http://localhost:8095 | The exchange's health: every process, trading activity, participants and events |

The trader-gui asks for an API key. The simulation's desks and their keys:

| Participant | Role | API key |
|---|---|---|
| `TRADER01` | student desk | `key-trader01-rwlr0v` |
| `TRADER02` | student desk | `key-trader02-uzytnc` |

| Participant | Role | API key |
|---|---|---|
| TRADER03 | student desk | key-trader03-rwqo7n |
| TRADER04 | student desk | key-trader04-3etdg0 |
| TRADER05 | student desk | key-trader05-k0c8fm |
| TRADER06 | student desk | key-trader06-xcv14m |
| TRADER07 | student desk | key-trader07-8b5rdy |
| TRADER08 | student desk | key-trader08-8ndd87 |
| TRADER09 | student desk | key-trader09-b3bj7h |
| TRADER10 | student desk | key-trader10-m1men1 |
| OPS01 | instructor (ADMIN) | key-ops01-3e34rs |

**These keys are public**

They are in the bundled example and in this book. The compose file publishes the exchange on `127.0.0.1` only, so nobody else can reach it. Before you open it to a network, read The Simulation Container.
**Checkpoint.** During trading hours, terminal-gui's watchlist ticks and book-gui's book for `AAPL` changes every second or two.

## 3.3 Look at the hidden value

The true values are secret from students, but not from the instructor. In the container, as `OPS01`:

```
./edumatcher.sh shell pm-market-sim --status --id OPS01
```

During trading this lists the symbols whose price is furthest from their true value:

```
pm-market-sim: RUNNING seq=1843 symbols=300 open_rumours=1
symbol        value        mid  mid/value
DXCM          84.12      85.31     +1.41%
...
```

Outside trading hours it says `PAUSED` and shows no values. The gap column is the whole point of the simulation; The True Value explains it.

`./edumatcher.sh shell` with no command puts you in the container with every `pm-*` tool on the PATH.

## 3.4 See a trading day now

To watch a day without waiting for 09:00, run the simulation with an accelerated clock that starts just before the open:

```
./edumatcher.sh speed 10 2026-11-02T08:58
./edumatcher.sh restart
```

`2026-11-02` is a Monday morning; speed `10` runs the calendar ten times faster than the clock on the wall. The pre-open starts within seconds, the opening auction two and a half minutes later, continuous trading half a minute after that, and the whole trading day takes about 42 minutes. Accelerated Time explains what changes and what does not.

To go back to real hours:

```
./edumatcher.sh speed 1
./edumatcher.sh restart
```

## 3.5 Stop it, start it again

```
./edumatcher.sh stop     # stop and remove the containers
./edumatcher.sh start    # start again where it left off
```

Everything the exchange has recorded — trades, statistics, logs, the model's values, the traders' daily results — is in `~/.edumatcher-sim/data` and survives `stop`. The containers restart on their own after a reboot (`restart: unless-stopped`). To start from scratch, stop it and delete `./data`.

## 3.6 What you have now

A market that will keep trading every weekday for as long as the machine is on. The rest of the book explains what you are looking at and how to change it. If you only read one more chapter, read The True Value.

## 3.7 If something does not work

| You see | Do |
|---|---|
| `pull access denied` or `manifest unknown` | The image for that version is not published yet: pin one that is with `--version <version>`, or build it — The Simulation Container |
| `port is already allocated` | Another EduMatcher stack (or anything else) holds 5560–5600, 8080/8081 or 8090–8095. Stop it first; only one stack can run at a time |
| The process table shows `dead` | Wait a minute: the container restarts dead processes once a minute. If it stays dead, see Troubleshooting |
| No trading during the day | Check the time zone: the clock is Stockholm's. `./edumatcher.sh shell date` |

CHAPTER 4

# The True Value

**Learning objectives**

After reading this chapter you will be able to:

- Explain price discovery with the simulation as the example: a price looking for a value nobody on the exchange can see
- Describe how the true values move, in words and in numbers
- Say who pulls prices toward the values and who pushes them away
- Measure how far the market is from its values, and change that
- Read and change every setting of the market model

**Prerequisites**: A Market That Runs Itself.

## 4.1 Price and value

A share's **price** is the last price two traders agreed on. Its **value** is what the share is really worth — on a real exchange, what the company's future earnings are worth today. Nobody knows the value exactly. Some traders have a good estimate of it and trade on the difference: they buy when the price is below their estimate, sell when it is above. Their orders pull the price toward the value. Everyone else — traders who follow momentum, who need cash, who simply guess — pushes the price around. The price you see is the outcome of that tug of war, and the process by which it finds the value is called **price discovery**.

On a real exchange nobody can check how well price discovery works, because the value is never revealed. In the simulation it exists: `pm-market-sim` keeps a true value for every share and moves it the way company values move. You, as the instructor, can see it; the students cannot; the AI traders that trade on value see it through errors of their own. So you can watch the gap between price and value open and close — and show it to a class after the fact.

## 4.2 How the values move

Every second of continuous trading, each share's value takes a small random step. The step is a mix of four things:

| Ingredient | What it is | In the `s300-sim` example |
|---|---|---|
| **The market** | One random move shared by every share | each share follows it with weight `beta_market`, 0.35–0.60 by sector |
| **The sector** | One random move per industry sector | weight `beta_sector`, 0.35–0.60 |
| **The share itself** | Its own random move | the rest of its volatility |
| **Jumps** | Now and then a sudden move: an unannounced event | 0.05 a day per share, size ±4 % ( ) |

Plus a slow **drift** (5 % a year), news (next chapter) and an **overnight** move when the market reopens. The size of the random steps is the share's **volatility**: 30 % a year by default, 18 % for the calmest sectors (consumer staples, utilities), 35 % for technology and energy, and spread a little around its sector for each share.

What that means in a day:

| | Typical size (one standard deviation) |
|---|---|
| A whole trading day, volatility 30 % | 1.9 %, about 2.1 % with the jumps |
| One hour of continuous trading | 0.7 % |
| One minute | 0.1 % |
| The overnight move (20 % of a day's variance) | 0.85 % |
| A jump (once in 20 trading days per share) | 4 % |

So on a typical day a share's value ends within about ±2 % of where it started, one day in twenty more than ±4 %; across the 300 shares, about 15 jump somewhere every day.

**Shares move together.** Two shares in the same sector move together with a correlation of about `beta_market²` + `beta_sector²` — 0.56 for two technology shares — and two shares in different sectors with about the product of their market betas, around 0.3. A bad day for the market is a bad day for most shares; a bad day for energy is a bad day for all 16 energy shares.

**The day has a rhythm.** Real markets are busiest after the open and before the close; so is the model. The values move about twice as fast (in variance) right after the open and before the close as at midday, averaging one day's worth over the day.

**Only during trading.** The values move only while the exchange trades continuously — not in the auctions, not at night. The night's move arrives in one step when the market reopens, so the opening auction has a new value to find.

**Where the values start**

On the first day each share's value starts at the engine's previous close, or, on a brand-new exchange, at the share's last price in the configuration — so price and value start equal. From then on `pm-market-sim` saves the values every 10 seconds (`market_sim_state.json`) and continues from them after any restart.

## 4.3 Who pulls, who pushes

| Pulls the price toward the value | Pushes it away |
|---|---|
| `MM01`, the market maker: every second it moves its own price 30 % of the remaining way to the value (`--anchor-sim 0.3`) and re-quotes around it | `noise-retail`, `market-taker`, `iceberg-seller`: orders in random directions |
| 22 `value-investors`: buy when the price is more than 0.4 % below their view of the value, sell when above | `trend-followers` and `scalpers`: buy what rose, sell what fell — they amplify moves |
| 8 `institutionals`: the same with a 0.6 % threshold, in big orders worked over three minutes | `contrarians` and `auction-players`: fade moves; they pull toward the recent *price*, not the value |
| | Every value trader's own error: each sees the value through a bias of its own (about ±1 %) |

The value traders do not see the value exactly. Each one has, per share, a persistent error drawn at random (standard deviation 1 % for value investors, 0.8 % for institutions) and redrawn now and then. That is why two value traders disagree about a price, and trade with each other — and why the market as a whole has a better estimate of the value than any single trader.

## 4.4 How close does the market get?

The yardstick is the gap `ln(price / value)` — for small gaps the same as the percentage difference — across all 300 shares, summarised as its root mean square (RMS). Measured on the `s300-sim` set-up at real speed on a 2-vCPU machine, minute by minute from the fifth to the sixteenth minute of continuous trading:

| Measure | Value |
|---|---|
| RMS gap, all shares | about 0.3 % (0.21–0.43 % from one minute to the next) |
| Median absolute gap | about 0.04 % |
| Shares more than 2 % away | 2 of the 300 on average, never more than 5 |

In other words half the prices sit within a few hundredths of a percent of their value, and a handful of shares — usually ones that just jumped, or a thinly traded one — are a percent or two away; those few make up most of the RMS. The gap is what keeps the value traders busy: if it were zero, they would have nothing to trade.

To see the gap yourself:

```
# From the host
docker exec edumatcher-sim pm-market-sim --status --id OPS01

# From the container
pm-market-sim --status --id OPS01
```

```
pm-market-sim: RUNNING seq=4120 symbols=300 open_rumours=0
symbol        value        mid  mid/value
LHX           62.63      65.63     +4.79%
GPN           28.51      27.61     -3.19%
...
```

It lists the shares furthest from their value. For every share, ask the API with the instructor's key:

```
curl -s -H "Authorization: Bearer key-ops01-3e34rs" http://localhost:8080/api/v1/admin/sim
```

The answer holds the model's state and the latest value of every share. The same data streams on the WebSocket `/api/v1/ws/admin/sim`. Only ADMIN keys get an answer; a student's key is refused.

## 4.5 Changing the model

The model's settings are in `market_sim.yaml`, deployed next to the engine configuration (in the container `/data/ref_data/market_sim.yaml`; how to deploy your own copy: The Simulation Container or A Complete Set-up by Hand).

```
version: 1
seed: 301
step_sec: 1.0
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}
  # ... ten more
symbols:
  AAPL: {sector: TECH, vol: 0.392}
  # ... 299 more
news: { ... }   # next chapter
```

A share's settings are `defaults`, overridden by its sector's, overridden by its own line.

| Setting | Range | Raise it and | Lower it and |
|---|---|---|---|
| `vol` | 0–3 | values wander further each day; prices lag them more; value traders trade more | a calm market where prices hug the values |
| `drift` | −1–1 | values creep up (a bull market); at 0.05 it is invisible within a day | negative: a slow bear market |
| `jump_rate` | 0–10 | more sudden moves the news never announced; more large gaps | 0: values only diffuse |
| `jump_std` | 0–1 | bigger surprises | |

| Setting | Range | Raise it and | Lower it and |
|---|---|---|---|
| `jump_mean` | −1–1 | jumps mostly up (positive) or down (negative) | |
| `beta_market` | 0–1 | shares move together; a market-wide day is visible in every book | shares independent |
| `beta_sector` | 0–1 | sectors move as blocks | |
| `overnight_fraction` | 0–5 | bigger gaps at the open; busier opening auctions | 0: the open continues the close |
| `step_sec` | 0.01–60 | coarser steps, fewer value messages | smoother values, more messages |
| `seed` | any integer 0 | a different but repeatable path | |
| `initial` (per symbol) | $> 0$ | start a share at a chosen value: the market finds it from its old price | |

`beta_market²` + `beta_sector²` must not exceed 1 for any share; the rest of its variance is its own.

### 4.5.1 Recipes

| You want | Change in `market_sim.yaml` |
|---|---|
| A calm market for a first lesson | `defaults: {vol: 0.15, jump_rate: 0.0}`, and the sectors' `vol` down too |
| A volatile market | every `vol` × 2, `jump_rate: 0.2` |
| A crash day | inject a market headline with impact −0.08 (next chapter) rather than changing the model |
| Mispriced shares to find | `initial` for a few shares, 10 % away from their last price: watch the value traders close the gap |
| Independent shares | `beta_market: 0` and `beta_sector: 0` everywhere |
| The same day again | the same `seed`, a fresh data directory |

Restart after a change: the model reads `market_sim.yaml` when it starts. A share's values carry on from where they were; only how they move from now on changes (except `initial`, which applies only to a share without a saved value).

## 4.6 The market maker's anchor

The strongest single force toward the value is the market maker. With `--anchor-sim 0.3` it moves its own price 30 % of the way to the value every second, so that price reaches a new value within a few seconds; its

quotes follow as soon as the price has moved more than 15 ticks (`--drift-ticks`). The value traders then trade against prices that are already about right.

Without it, the value traders do all the pulling. That is a more interesting market to watch — prices wander further, rumours move them — but with one caveat: a market maker that is *not* anchored, and simply re-quotes around its own last mid, is worse than none. It refills the same prices whatever happens, and informed traders cannot move it. In a test the gap grew from under 1 % to 2–4 % within minutes. Either anchor the market maker or stop it; Tuning the Traders shows how.

## 4.7 See also

- News and Rumours — the other way values move
- The Market Model — the reference: the formula, the sockets, `--init`
- How the AI Traders Decide — the `value` strategy in detail

CHAPTER 5

# News and Rumours

**Learning objectives**

After reading this chapter you will be able to:

- Read a headline the way the simulation means it: scope, kind, sentiment and the secret impact behind it
- Follow a rumour from its first headline to its confirmation or retraction
- Predict how the AI traders and the market maker react to news
- Turn generated news up, down or off, and publish news of your own in front of a class

**Prerequisites**: The True Value.

## 5.1 Why news

In the simulation, as on a real exchange, the true value of a share does two things: it drifts a little all the time, and now and then it jumps because something happened. The jumps are what news is for. A headline tells the market that the value has moved — and roughly which way — before the price has caught up. Who reads it first, who believes it and who ignores it is what turns a headline into a price move you can watch.

## 5.2 Anatomy of a headline

| Field | Public? | Meaning |
|---|---|---|
| **Scope** and targets | yes | One symbol, one sector, or the whole market |
| **Kind** | yes | `EARNINGS`, `GUIDANCE`, `MNA`, `REGULATORY`, `PRODUCT`, `LEGAL`, `MANAGEMENT`, `MACRO`, `OPERATIONS`, `RESEARCH`, `CONTRACT` or `ESG` |
| **Headline** | yes | The text, e.g. *"NVDA raises its outlook"* |

| Field | Public? | Meaning |
|---|---|---|
| **Sentiment** | yes | How good it sounds, from −1 (very bad) to +1 (very good) |
| **Status** | yes | `CONFIRMED` (it happened), `RUMOUR`, then `CONFIRMED` or `RETRACTED` |
| **Credibility** | yes, rumours only | How believable the rumour looks, 0 to 1 |
| **Impact** | **no** | The log move it causes in the true value: 0.05 is about +5.1 %, −0.10 about −9.5 % |

Students see everything but the impact: in terminal-gui's news view, the API (`GET /api/v1/news`) and `pm-ticker`. The impact is the market model's secret, exactly like the true value.

**How far a headline moves values.** A symbol headline moves that symbol by its impact. A sector headline moves each symbol of the sector by `beta_sector × impact`, a market headline every symbol by `beta_market × impact` — so with the `s300-sim` betas a sector headline moves its symbols by about 35–60 % of the impact, and a market headline every symbol by 35–60 % of it.

## 5.3 Generated news

Left alone, `pm-market-sim` writes news itself, at random moments during continuous trading. The `s300-sim` example keeps the defaults:

```
news:
  rate_symbol: 6.0         # symbol headlines per trading day, whole exchange
  rate_sector: 0.5         # sector headlines per trading day
  rate_market: 0.3         # market headlines per trading day
  rumour_share: 0.3        # share that starts as a rumour
  rumour_confirm: 0.6      # chance a rumour is confirmed rather than retracted
  rumour_delay_min: 0.02   # trading days before a rumour resolves (uniform)
  rumour_delay_max: 0.15
```

What that means on a 300-symbol exchange:

| | On average |
|---|---|
| Symbol headlines | 6 a day — one every hour or so; any one share gets news about once every 50 trading days |
| Sector headlines | one every two trading days |
| Market headlines | one every three or four trading days |
| Rumours | about 2 a day; they resolve 8 to 60 minutes of continuous trading later, 3 in 5 confirmed |

A "trading day" here is the 6.5 hours of continuous trading; the rates are per trading day whatever speed the calendar runs at (see Accelerated Time).

### 5.3.1 What kind of news, how big

Each headline is drawn from a catalogue of *stories*, and each story has its own size: the impact is normally distributed with the story's mean and spread. There are two sorts.

**Everyday news** can go either way. Its headline follows the sign of the impact: *"NVDA beats earnings expectations"* or *"NVDA misses earnings estimates"*.

| Kind | Scopes | Share of symbol news | Impact mean | Impact spread ( ) | Typical move |
|---|---|---|---|---|---|
| `EARNINGS` | symbol | 24 % | 0 | 5 % | ±5 % |
| `GUIDANCE` | symbol | 12 % | 0 | 3 % | ±3 % |
| `PRODUCT` | symbol | 12 % | +0.5 % | 2 % | ±2 % |
| `REGULATORY` | symbol, sector | 6 % | −1 % | 4 % | ±4 %, a little more often bad |
| `LEGAL` | symbol | 6 % | −1 % | 3 % | ±3 %, a little more often bad |
| `MANAGEMENT` | symbol | 6 % | 0 | 2 % | ±2 % |
| `MNA` | symbol | 3 % | +6 % | 6 % | a takeover bid: mostly up, sometimes a lot |
| `MACRO` | sector, market | — | 0 | 1.5 % | ±1.5 % before the betas |

**Events** go one way only, and they are big: a fire is always bad news, a breakthrough always good. Together they are about a third of symbol news, so with the default rates a class sees two or three a day, which is far more often than in real life, so that students get to watch what such a shock does to a price.

| Kind | Examples | Typical move |
|---|---|---|
| `OPERATIONS` (15 %) | *"Fire ravages XOM's main refinery"*, flooding, a cyber-attack, a strike, a parts shortage; good: a site reopening early, a new oil find | −2.5 % to −5 %; +2 % to +5 % |
| `ESG` (7 %) | *"Environmental protests delay TSLA's new battery plant by 12 months"*, a fine for breaking environmental rules, an oil spill, a wildfire blamed on a utility's lines; good: a permit won after a long battle | −2 % to −8 %; +2.5 % |

| Kind | Examples | Typical move |
|---|---|---|
| `CONTRACT` (5 %) | *"CAT signs a $5 bn long-term deal with a major new customer"*, losing the largest customer, a defence contract | +4.5 % to +5 %; −5 % |
| `RESEARCH` (4 %) | *"NVDA breakthrough lifts next-generation chip performance by 35 %"*, a late-stage drug trial that succeeds or fails, a flagship R&D project written off, a chip that slips a year | +5 % to +8 %; −5 % to −10 % |
| `PRODUCT` (2 % of the 14 %) | *"GM recalls 300,000 vehicles over a brake defect"*, a product recalled over a safety defect, deliveries halted | −6 % |

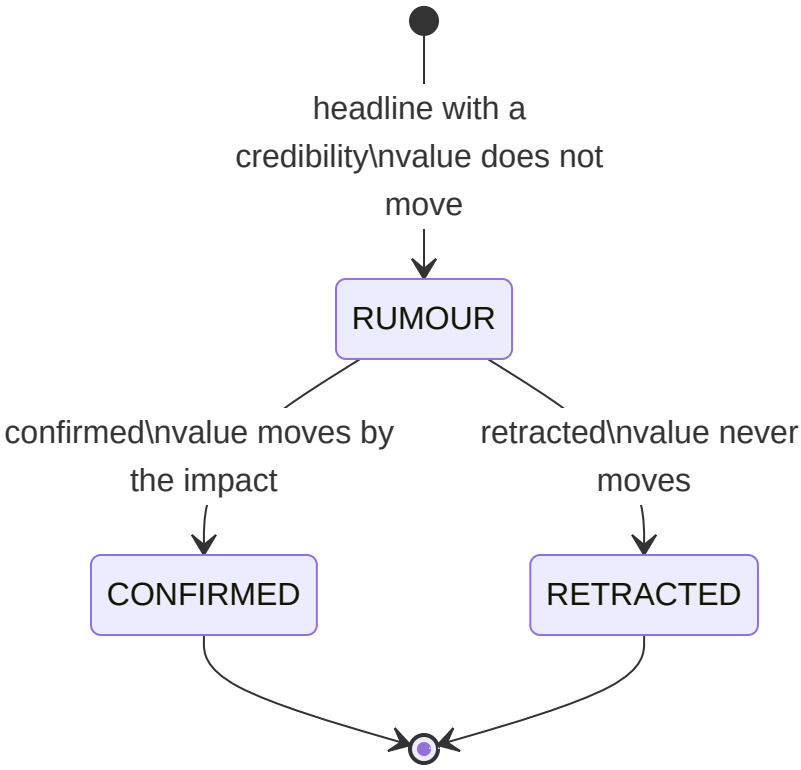
Some events only happen in their own industry: chip stories to chipmakers, trials to drug makers, recalls of vehicles to carmakers, groundings to airlines, defence contracts to aerospace. Other events need a fitting sector: a bank has nothing to recall, and a utility has no largest customer. A number in the headline grows with the impact: a chip that is 40 % faster moves the value more than one that is 20 % faster.

Sector headlines are `REGULATORY` (42 %, including export curbs on chips and drug-pricing reform), `MACRO` (40 %) or `OPERATIONS` (18 %: a supply-chain disruption, a port strike, a hurricane that shuts refineries). Market headlines are always `MACRO`.

### 5.3.2 Sentiment: a hint, not the answer

The sentiment students see is the impact made public, but blurred: for everyday news roughly impact ÷ (2 × the story's spread), plus a little noise ( 0.15), clipped to ±1. So an earnings beat of one spread (+5 %) reads about +0.5, and a big miss of two spreads (−10 %) about −1. For an event it is impact ÷ (|mean| + spread) plus the same noise, so a typical fire reads about −0.7 and a bad one −1. An event's sentiment always has the right sign. For everyday news the sign is almost always right. For both, the size is only a guide. Two headlines with the same sentiment can move the value by quite different amounts, which is why value traders, who see the value itself, keep correcting what news traders start.

## 5.4 Rumours




A rumour is a headline that may not be true. It carries a **credibility** and moves nothing in the model. Later it is confirmed — a second headline, and only then does the value move — or retracted, and the value never moves at all. Prices often do move on a rumour, though: that is traders betting on it, and when a retracted rumour's price move unwinds, someone has lost money.

Credibility is informative but not reliable. A generated rumour that will be confirmed gets a credibility between 0.4 and 0.9; one that will be retracted between 0.1 and 0.6. A credibility of 0.8 is almost surely true, one of 0.2 almost surely false, and anything in 0.4–0.6 is a coin toss.

## 5.5 Who reacts, and how

| Who | Reacts to | How |
|---|---|---|
| `news-trader` (5 agents) | every headline touching its symbols | Hears it after a lag of its own (median 5 s), then trades in the headline's direction while its conviction — \|sentiment\|, times credibility for a rumour, halving every 2 minutes — stays above 0.15; pays up to `0.08 × conviction` beyond the price it first saw |
| `value-investor`, `institutional` (30 agents) | the true value, which news moves | Buy or sell toward the new value as soon as it moves. On a rumour, which does not move the value, they shift their own view by `0.05 × sentiment × credibility` |
| `MM01`, the market maker | the true value | Moves its own price 30 % of the way toward the value every second and re-quotes once that has moved 15 ticks — so a confirmed headline moves its quotes within seconds, and a rumour does not move them at all |
| Everyone else (65 agents) | prices only | Trend followers and scalpers chase the move once it shows; contrarians fade it |

Put together, a confirmed headline is priced in fast: in a test with an anchored market maker and the same kinds of trader on 150 symbols, a +10 % earnings surprise moved the price +6.5 % within 5 seconds and +10 % within 15 seconds. Rumours leave much less trace than you might expect. The market maker sells to every rumour buyer at its value-anchored price, so in the same test a credible good rumour (sentiment 0.7, credibility 0.5) lifted the price under 3 %, and a sector rumour moved its sector by less than 1 %. To show students how a rumour moves a market, run without the market maker — see Tuning the Traders.

## 5.6 Steering the news

### 5.6.1 Fewer, more, or none

Change the rates in `market_sim.yaml` (how to deploy your copy: Tuning the Traders):

| You want | Set |
|---|---|
| A quiet market where you alone publish news | every `rate_*` to `0` |
| A news-heavy market, a headline every few minutes | `rate_symbol: 60` (and perhaps `rate_sector: 3`) |
| More rumours, harder to read | `rumour_share: 0.6`, `rumour_confirm: 0.5` |

| You want | Set |
|---|---|
| Rumours that linger for a lesson | `rumour_delay_min: 0.1`, `rumour_delay_max: 0.3` |
| No market-wide shocks | `rate_market: 0` |

Each rate is a Poisson rate: the gaps between headlines are random, so `rate_symbol: 60` gives ten an hour *on average*, sometimes three in a minute, sometimes none for twenty.

### 5.6.2 Publishing news yourself

`pm-news` sends a headline to the running market model as an ADMIN participant. In the container:

```
# From the host
docker exec edumatcher-sim pm-news --id OPS01 inject --symbol NVDA \
    --kind EARNINGS --sentiment 0.8 --impact 0.08 \
    --headline "NVDA smashes estimates on data-centre demand"

# From within the container
pm-news --id OPS01 inject --symbol NVDA \
    --kind EARNINGS --sentiment 0.8 --impact 0.08 \
    --headline "NVDA smashes estimates on data-centre demand"
```

The value of NVDA moves +8.3 % at once; the price follows within seconds. A rumour you inject waits for you:

```
# From the host
docker exec edumatcher-sim pm-news --id OPS01 inject --sector ENERGY \
    --kind REGULATORY --sentiment -0.7 --impact -0.06 --rumour --credibility 0.6 N12
docker exec edumatcher-sim pm-news --id OPS01 confirm N12     # or: retract N12

# From within the container
pm-news --id OPS01 inject --sector ENERGY \
    --kind REGULATORY --sentiment -0.7 --impact -0.06 --rumour --credibility 0.6 N12
pm-news --id OPS01 confirm N12     # or: retract N12
```

`inject` prints the headline's id (`N12` here). A rumour you inject never resolves on its own. Your headlines and generated ones mix freely; set the rates to 0 if yours should be the only ones.

### 5.6.3 Scenarios

For a lesson, script the news in a file and play it:

```
# From the host
docker exec edumatcher-sim pm-news --id OPS01 play \
    /opt/edumatcher/.venv/lib/python3.14/site-packages/docs/examples/news/earnings-surprise.yaml

# From within the container
pm-news --id OPS01 play \
    /opt/edumatcher/.venv/lib/python3.14/site-packages/docs/examples/news/earnings-surprise.yaml
```

Four scenarios ship with EduMatcher — `earnings-surprise`, `rumour-confirmed`, `rumour-retracted` and `chip-race` (a chip breakthrough, then a rumoured fire at a rival's fab) — and fit `s300-sim` as they fit the `s150-*` examples. Their format, and how to write your own, is in The Market Model; the AI Traders & News exercises

use them in class.

### 5.6.4 Watching the news

| Where | What |
|---|---|
| terminal-gui, news view | Every headline as it is published, with status and sentiment |
| `docker exec edumatcher-sim pm-news list` | Headlines printed as they arrive, until Ctrl+C |
| `docker exec edumatcher-sim pm-news past --limit 20 --date 2026-10-14` | Recent headlines from that session date, newest first |
| `GET /api/v1/news` (any API key) | Today's headlines, and the sector of every symbol |

## 5.7 Try it

1. Pick a quiet symbol in book-gui and note its price and the value (`pm-market-sim --status --id OPS01` lists it if it is far off; the admin API, `GET /api/v1/admin/sim` with the `OPS01` key, gives all of them).
2. Inject a confirmed headline with impact `0.05` and watch the price for a minute. Who traded first? (The tape and the order book show `AI` ids.)
3. Inject a rumour with sentiment `0.8`, credibility `0.9`, and no impact worth speaking of (`--impact 0.001`). Does the price move? Why not much?
4. Retract it. Then try again without the market maker (Tuning the Traders shows how to remove it).

## 5.8 See also

- The Market Model — the reference for `pm-news` and scenarios
- How the AI Traders Decide — the `news` and `value` strategies in full
- AI Traders & News — classroom exercises with news

CHAPTER 6

# How the AI Traders Decide

**Learning objectives**

After reading this page you will be able to:

- Follow one AI trader from a decision to the orders it sends
- Explain each strategy, execution style, tempo and risk setting
- Predict how a preset will behave, and write your own

**Prerequisites**: AI Traders.

## 6.1 The loop

An AI trader does nothing on a fixed timer. Decisions arrive at random — a Poisson process at the tempo's `decisions_per_min` — so a swarm's orders do not come in bursts. In between, it keeps its house in order. Each time it wakes, in this order:

1. **Housekeeping.** Cancel resting orders older than `max_order_age_sec`, or more than `stale_price_ticks` behind the touch on their own side (a buy order behind the best bid, a sell order behind the best ask). Not in pre-open and the auction calls: there an order waits for the uncross, and whatever is left after it goes at the first sweep of continuous trading.
2. **Slices due** of a `twap` parent order.
3. **Protection.** Place, resize or remove the protective orders for each open position (continuous trading only).
4. **A decision**, if one is due: the strategy names a symbol, a side, an urgency and a size multiplier — an *intent* — or nothing. The execution style turns the intent into orders, within the risk limits.

During continuous trading the decision rate follows the day's U shape: about twice the average after the open and before the close, half of it at midday. A swarm's `budget` scales every trader's rate so the swarm as a whole sends the actions per second it was given.

The trader reads the market from the engine's public feed — book snapshots, trades, the session, halts — and from `pm-market-sim` the true values and the news. It sends nothing for a halted symbol, nothing while the market is closed, and only what each phase allows: resting LIMIT and ICEBERG orders in the call phases, ATO in the opening auction, ATC in the closing auction, everything during continuous trading.

## 6.2 Strategies: what to trade

A strategy says which way it leans with a *score* from −1 to 1 and buys with probability `(1 + score × strength) / 2`, so it leans without becoming predictable.

| Strategy | Looks at | Buys when | Parameters |
|---|---|---|---|
| `noise` | one random symbol | at random (score 0) | `urgency` |
| `trend` | `sample` random symbols, acts on the strongest signal | its fast average price (30 s) is above its slow one (300 s) | `strength`, `sample`, `urgency` |
| `reversion` | the same way | the price is below its slow average | `strength`, `sample`, `urgency` |
| `value` | `sample` random symbols, acts on the widest gap | the price is below its view of the true value by more than `threshold` | `threshold`, `bias_std`, `redraw`, `sample`, `urgency`, `rumour_shift` |
| `news` | the news touching its symbols | the headline is good news | `lag_sec`, `lag_sigma`, `half_life_sec`, `threshold`, `urgency`, `move` |

**Trend and reversion** need about 30 seconds and 20 prices of a symbol before they have an opinion on it. Both scale the difference by the symbol's own recent volatility, so a 1% move means more in a quiet stock than in a wild one. The stronger the signal, the more urgent the intent.

**Value** sees each symbol's true value through a bias of its own — a persistent error, drawn per symbol with standard deviation `bias_std` and re-drawn now and then (`redraw`). That is why two value traders disagree, and trade with each other. It acts once the log gap between its view and the price exceeds `threshold`; the wider the gap, the more urgent and the larger the order, up to three times its usual size. When it crosses the spread it pays up to its view of the value less `threshold` — not one tick past the touch — so a jump in the value is priced in by a few orders rather than a thousand one-tick steps. An open rumour shifts its view by `rumour_shift × sentiment × credibility`.

**News** hears each headline after a delay of its own (lognormal, median `lag_sec`), so news traders do not all hit the book in the same millisecond. From then on its conviction is the headline's |sentiment| — times the credibility for a rumour — halving every `half_life_sec`. While conviction is above `threshold` it trades the headline's symbols in the headline's direction, urgently and in up to three times its usual size, paying up to `move × conviction` (a log move) beyond the price it saw when it first acted on the headline. A confirmation turns a rumour into news known to be true; a retraction leaves an impulse the other way, as large as the rumour was believable.

## 6.3 Execution styles: how to trade it

| Style | Sends | Parameters |
|---|---|---|
| `passive` | A LIMIT `offset_ticks` behind the touch (0 joins it), or a random depth up to `offset_max`, most of them near the touch | `offset_ticks`, `offset_max`, `tif` |
| `marketable` | A LIMIT `cross_ticks` through the touch; what does not fill rests | `cross_ticks`, `tif` |
| `sweep` | A MARKET, IOC or FOK order against the other side | `sweep_type`, `cross_ticks` |
| `iceberg` | An ICEBERG behind the touch showing `visible_fraction` of its size | `offset_ticks`, `visible_fraction` |
| `twap` | A parent order cut into `slices` child orders over `horizon_sec` | `slices`, `horizon_sec`, `child_style` |

Any style becomes marketable when the intent's urgency reaches `urgency_cross`. A crossing order (marketable, IOC, FOK) is priced at the strategy's limit when it gives one (`value` and `news` do), otherwise `cross_ticks` past the touch. In the call phases — pre-open and the auctions — a trader takes part in a decision with probability `auction_participation`, and then sends a limit order straddling the reference price (with TIF ATO or ATC in the auctions), so that buyers and sellers overlap at the uncross. Prices are kept inside the symbol's price collar and quantities inside its order limits: an AI trader does not send what the engine would refuse.

Three more execution settings, off unless a preset sets them:

| Setting | Effect |
|---|---|
| `smp` | The self-match prevention mode sent on every order (`NONE`, `CANCEL_AGGRESSOR`, `CANCEL_RESTING`, `CANCEL_BOTH`); unset, the gateway's default applies |
| `amend_fraction` | That share of the orders the risk rules would cancel is amended instead: one left behind is re-priced to its own side's touch (losing priority), one still at the touch is trimmed to half its unfilled quantity (keeping priority) |
| `combo_fraction` | That share of continuous-trading decisions is sent as a small all-or-none COMBO: the intended order and the opposite side of another of the trader's symbols, both at the touch, a quarter of the size |

## 6.4 Tempo: how often and how big

| Tempo | Decisions per minute | Order size | Sizes |
|---|---|---|---|
| `many-small` | 30 | 1–25 | mostly small |
| `aggressive` | 20 | 20–120 | even |
| `cautious` | 6 | 10–60 | even |
| `few-large` | 2 | 150–700 | mostly large |

A preset can give its own instead: `tempo: {decisions_per_min: 12, size_min: 5, size_max: 80, size_distribut small-heavy}` (`balanced`, `small-heavy` or `block-heavy`). The strategy's size multiplier applies on top.

## 6.5 Risk: what it will not do

| Setting | Effect |
|---|---|
| `max_position` | No order that could take the position past ± this many shares, counting open orders as if filled |
| `max_live_orders_per_symbol` | At most this many working orders per symbol |
| `max_order_age_sec` | Cancel resting orders older than this |
| `stale_price_ticks` | Cancel resting orders this far behind their own side's touch |
| `protection`, `protection_pct` | Protective orders for open positions, `protection_pct` of the price away |

`protection` is one of `none`, `stop` (a STOP order), `stop_limit`, `trailing` (a TRAILING_STOP) or `bracket` (an OCO of a take-profit LIMIT and a STOP). It follows the position: a new fill resizes it, a flat position removes it.

A trader that gets 25 orders refused within 10 seconds pauses for 5 seconds — the *reject breaker* — so a misconfigured preset cannot flood the exchange with refused orders. A cancel that lost the race with a fill, or an order in flight when the market closed, does not count.

## 6.6 Writing a preset

Start from the closest built-in preset in `src/edumatcher/ai_trader/presets/` and change one part at a time:

| You want | Change |
|---|---|
| More flow | `tempo` |
| Takes liquidity instead of providing it | `execution.style: sweep` or `marketable` |
| Holds bigger positions | `risk.max_position` |
| Reacts to news faster | `strategy.lag_sec` (news), `execution.urgency_cross` |
| Disagrees more with other value traders | `strategy.bias_std` (value) |

Every parameter has a range; a value outside it is refused with the file and the key named. AI Traders shows how to run your preset.

## 6.7 The other bot: `pm-mm-bot`

`pm-ai-trader` trades like a directional participant. The market-maker bot has a different job: it posts two-sided quotes on many symbols from one participant ID. Run one with the swarm — AI traders alone make a thin, jumpy book.

CHAPTER 7

# Tuning the Traders

**Learning objectives**

After reading this chapter you will be able to:

- Find every setting that changes how the simulated market behaves
- Predict what a change does — to volume, spreads, volatility and how closely prices follow the values — before you make it
- Make the market calmer, wilder, better or worse informed, more or less news-driven, and check that the change worked

**Prerequisites**: The True Value, News and Rumours, How the AI Traders Decide.

## 7.1 Where the settings live

The market you see is the sum of four things, each with its own file:

| Layer | Decides | File | Read by |
|---|---|---|---|
| **The model** | How the true values move; how much news there is | `market_sim.yaml` | `pm-market-sim` |
| **The swarm** | How many traders, which kinds, how busy, which symbols | `swarm.yaml` | `pm-ai-swarm` |
| **The presets** | How each kind of trader trades | preset YAML files | `pm-ai-swarm` |
| **The market maker** | How tight and deep the quotes are, how fast they follow the value | its command-line flags in the process profile | `pm-mm-bot` |

and the exchange underneath — symbols, tick sizes, trading hours, the market maker's obligations — in `engine_config.yaml`.

How to deploy your own copy of each:

| Running | `market_sim.yaml`, `engine_config.yaml` | `swarm.yaml`, presets, market-maker flags |
|---|---|---|
| In the container | put them in `./data-sim/custom/`, start with `EM_CONFIG_FILE=/data/custom/engine_config.yaml` | `pm-opctl-cli init`, then edit `./data-sim/emo-config.yaml` |
| By hand | `pm-config-deploy my-sim/engine_config.yaml` | `pm-opctl-cli init`, then edit `$EDUMATCHER_DATA_DIR/emo-config.yam` |

The details are in The Simulation Container and A Complete Set-up by Hand. Every change takes effect when the process that reads it restarts; the simplest is to restart the whole stack.

## 7.2 What to watch

Before changing anything, decide what you want to move, and how you will see it move:

| Property | What it means | How to see it |
|---|---|---|
| **Activity** | Orders and trades a second | the growth of `actions=…` between two of the swarm's once-a-minute status lines (the rate in brackets is the average since the start; see Watching the Market): `pm-log-cli query --process pm-ai-swarm --grep actions= --limit 3`; trades in terminal-gui |
| **Spread and depth** | How tight and thick the books are | book-gui |
| **Volatility** | How far prices move in a minute, a day | terminal-gui charts; `pm-stats-cli daily` (high − low) |
| **Efficiency** | How close prices stay to the values | `pm-market-sim --status --id OPS01`; the admin API |
| **Reaction to news** | How fast and far prices follow a headline | inject one and watch (News and Rumours) |
| **Who wins** | Profit and loss per kind of trader | the daily summaries in `/data/ai_swarm/` (Watching the Market) |

## 7.3 The swarm

`swarm.yaml` of the `s300-sim` example:

```
version: 1
agents: {prefix: AI, start: 1, count: 100}
workers: 0
budget: 120           # order actions per second, whole swarm
seed: 1000
symbols:
  per_agent: 30       # each agent trades 30 of the 300 symbols
composition:
  noise-retail: 15
  value-investor: 22
  news-trader: 5
  institutional: 8
  scalper: 8
  trend-follower: 10
  contrarian: 10
  iceberg-seller: 4
  market-taker: 4
  block-taker: 2
  auction-player: 2
  patient-liquidity: 10
```

| Setting | What it does | Raise it | Lower it |
|---|---|---|---|
| `budget` | Order actions (new orders and cancels) a second for the whole swarm. The swarm scales every trader's decision rate to meet it, correcting every 10 seconds | Busier books, more trades, more CPU and disk | A quieter market; too low, and some symbols go a long time without a trade |
| `count` | How many traders. Each needs a participant `AI001`… in the engine configuration (`s300-sim` has 100) | More, smaller voices: the same budget spread thinner, more disagreement | Fewer: each trader's quirks show more |
| `symbols.per_agent` | Symbols each trader trades. Each symbol is traded by about `count × per_agent / 300` traders | More traders per symbol: every symbol gets every kind of trader, smoother prices | Fewer per symbol: some symbols get only noise traders and never find their value |
| `composition` | The mix of presets (weights; any sum) | — | — |
| `seed` | The traders' random choices; the same seed and market give the same decisions | — | — |

| Setting | What it does | Raise it | Lower it |
|---|---|---|---|
| `workers` | Processes the traders run in; `0` = one per hundred traders, at most the CPUs minus one | Uses more cores | — |

**The budget needs a few minutes**

After the open the traders fire off their opening orders and the controller has nothing to learn from yet: in the first minutes of a day the swarm sends several times its budget, and settles to it within about four minutes. A budget is an average, not a cap.

### 7.3.1 The composition: who the market is made of

The mix is the most powerful setting there is. In the `s300-sim` mix, 35 of the 100 traders know something about the values or the news (22 value, 8 institutional, 5 news); 65 trade on prices alone.

| Give more weight to | And the market gets |
|---|---|
| `value-investor`, `institutional` | Prices closer to the values; quicker recovery after noise; less volatility than the values themselves |
| `news-trader` | Faster, sharper reactions to headlines — including rumours that turn out false |
| `noise-retail` | More volume and depth near the touch; prices that wander further from the values |
| `patient-liquidity` | Deeper books: more price levels behind the best prices, and moves that run into resting size |
| `trend-follower`, `scalper` | Momentum: moves run further than the value justifies, then snap back; fatter tails |
| `contrarian` | Mean reversion: moves get faded, prices stickier, short-term volatility lower |
| `market-taker`, `block-taker` | The spread crossed more often; more trades, more volatility, thinner books |
| `iceberg-seller` | Hidden size: books look thinner than they trade |
| `auction-player` | Busier opening and closing auctions |

A market of only informed traders hardly trades — they agree with each other about the value, give or take their 1 % errors. A market of only noise traders trades a lot and finds nothing. Real markets, and good teaching markets, are in between.

## 7.4 The presets

Each preset is a strategy, an execution style, a tempo and risk limits (How the AI Traders Decide defines every parameter). To change one kind of trader, copy its preset file from the package (`pm-ai-trader --list-presets`

names them; they are in `edumatcher/ai_trader/presets/`), change it, give it a new `name`, and put the file's path in the composition instead of the built-in name:

```
composition:
  noise-retail: 25
  my-value-investor.yaml: 22     # relative to this swarm.yaml
  # ...
```

What each lever does:

### 7.4.1 Strategy

| Parameter | Strategies | Raise it and the trader… |
|---|---|---|
| `urgency` | all | crosses the spread more often (and sooner reaches `urgency_cross`) |
| `strength` | trend, reversion | follows its signal more consistently instead of half at random |
| `sample` | trend, reversion, value | looks at more symbols per decision and picks the strongest signal — a sharper eye, more concentrated trading |
| `threshold` | value | waits for a bigger gap before trading: fewer, larger corrections, prices further from the values |
| `bias_std` | value | is more wrong about the value: more disagreement, more trading among value traders, a noisier consensus |
| `redraw` | value | changes its mind more often |
| `rumour_shift` | value | moves its view more on a rumour |
| `lag_sec`, `lag_sigma` | news | hears news later, and more spread out from the other news traders |
| `half_life_sec` | news | keeps trading a headline longer |
| `threshold` | news | ignores weaker news |
| `move` | news | pays further beyond the pre-news price: news is priced in faster, and overshoots more |

### 7.4.2 Execution

| Parameter | Raise it and the trader… |
|---|---|
| `offset_ticks` | rests further from the touch: more depth behind the best price, fewer fills |
| `offset_max` | spreads its resting orders over more price levels, from `offset_ticks` out to this, most of them near the touch |
| `cross_ticks` | pays further through the touch when it crosses |
| `urgency_cross` | crosses less often (it crosses when urgency reaches this) |
| `visible_fraction` (iceberg) | shows more of its order |
| `slices`, `horizon_sec` (twap) | works its order in more pieces over a longer time: less impact, slower to reach the value |
| `auction_participation` | takes part in more auction decisions |
| `tif: GTC` | leaves orders on the book overnight |
| `gtc_fraction` | sends that share of its resting orders as GTC, whatever `tif` says: a few long orders that outlive the day (a trader keeps at most one per symbol, and never cancels it as stale) |
| `amend_fraction` | amends more of its stale orders instead of cancelling them: re-priced to the touch, or trimmed when still at it |
| `combo_fraction` | sends more of its continuous decisions as two-symbol all-or-none combos |
| `smp` | (not a number) picks the self-match prevention mode on its orders |

### 7.4.3 Tempo

`decisions_per_min` sets how often a trader decides *relative to the others*: the swarm's budget then scales everyone's rate up or down together. So with a budget, doubling one preset's `decisions_per_min` doubles its share of the activity, not the total. Order size (`size_min`, `size_max`, `size_distribution`) is not scaled: bigger orders mean more volume, more impact per trade and more inventory.

### 7.4.4 Risk

| Parameter | Raise it and the trader… |
|---|---|
| `max_position` | holds larger positions: informed traders can push prices further toward the value before they are full |
| `max_live_orders_per_symbol` | keeps more orders working: more depth |
| `max_order_age_sec` | leaves orders longer: more depth, staler prices |

| Parameter | Raise it and the trader… |
|---|---|
| `stale_price_ticks` | tolerates its order further behind the touch before cancelling |
| `protection_pct` | puts its stops further away: fewer stop-outs, bigger losses when they hit |

## 7.5 The market maker

`MM01` quotes every symbol: a bid and an ask 20 cents apart, 200 shares each, centred on a price it moves toward the true value every second. Its flags are in the `ai-sim` profile's `mm-bot` entry:

```
pm-mm-bot --gateway-id MM01 --all-symbols --strategy symmetric --gap 0.20 --qty 200
          --reissue-delay-ms 5000 --drift-ticks 15 --anchor-sim 0.3
```

It uses the `symmetric` strategy: it quotes its own price ± half the gap and re-quotes only when that price has moved more than `--drift-ticks`, or a few seconds after a fill. The `passive` strategy of the `mm-demo` profile steps behind other traders' orders instead, and re-quotes whenever those orders change; with a hundred AI traders on 300 symbols that was 83 % of all the order messages on the exchange and half its CPU, for the same price quality.

| Flag | Now | Raise it | Lower it |
|---|---|---|---|
| `--anchor-sim` | 0.3 | Quotes follow the value faster: prices stick to the values, rumours barely move them | Slower to follow; `0` = ignores the values (see the warning below) |
| `--gap` | 0.20 | A wider spread to cross: noise traders pay more, value traders wait for bigger gaps. The engine's obligation for the market maker caps it at 20 ticks (0.20) | Tighter books |
| `--qty` | 200 | Deeper quotes: harder for any trader to move a price | Thinner: prices move on smaller orders |
| `--reissue-delay-ms` | 5000 | Waits longer to re-quote after a fill: gaps in the book after a fill | Refills at once: a wall at its price |
| `--drift-ticks` | 15 | Re-quotes less often as its price moves: quotes lag the value by up to this many ticks | Re-quotes more often (more messages) |

**Anchored, or not at all**

A market maker that ignores the values (`--anchor-sim 0`) keeps refilling its quotes around its own last price. Informed traders then cannot move the price, and the market drifts away from the values: in a test, 2–4 % within minutes, against under 1 % anchored, and 1.4 % with no market maker at all. If you want the value traders to do the work, remove the market maker rather than un-anchoring it.

**Removing it.** In the container a dead process comes back within a minute, so stopping `mm-bot` is not enough: write the profiles out (`pm-opctl-cli init`), delete the `mm-bot` entry from the `ai-sim` profile in `emo-config.yaml`, and restart. Without it the books are the AI traders' orders alone: thinner, and with a narrower median spread (0.03 % of the price against 0.10 %; the traders compete for the touch instead of queueing behind the market maker). Prices sit three times further from the values (1.3 % against 0.44 %, measured over a day at `EM_SPEED=24`), and rumours move prices visibly — the right market for a lesson on rumours.

## 7.6 The model

The values and the news have their own chapters: The True Value and News and Rumours. In short: volatility and jumps decide how hard the traders have to work to keep up; news rates decide how often the informed traders get something to trade.

## 7.7 Recipes

Start from the `s300-sim` settings and change only what the recipe says. Each was run against the default on the same machine, with the same seeds: 16 minutes of continuous trading at real speed, or, where the effect needs a whole day (news, rumours), a day at `EM_SPEED=24`. The default for comparison: prices 0.29 % from the values (RMS), a typical one-minute price move of 0.42 %, 1,400 trades a minute, 290 of the 300 symbols trading in any five minutes; at `EM_SPEED=24`, 0.44 % and 0.73 %.

| Recipe | Change | Measured, against the default |
|---|---|---|
| **Calm first lesson** | `market_sim.yaml`: every `vol` halved, `jump_rate: 0`, news rates 0; `swarm.yaml`: `budget: 30` | Prices 0.16 % from the values, one-minute moves 0.22 %: half of both. 345 trades a minute, 263 symbols in five minutes, a quarter of a core |
| **Wild market** | every `vol` × 2, `jump_rate: 0.2`; composition: `trend-follower: 20`, `scalper: 15`, `market-taker: 8` | One-minute moves 0.57 % (+35 %), 1,550 trades a minute, every symbol trading; prices still close to the values (0.35 %) |
| **Inefficient market** | composition: `value-investor: 5`, `institutional: 2`; remove the market maker | Prices 2.1 % from the values, seven times the default; 36 symbols more than 2 % off at any moment. A market for students to beat |

| Recipe | Change | Measured, against the default |
|---|---|---|
| **Efficient market** | composition: `value-investor: 40`, `institutional: 15`; `--anchor-sim 0.5` | No measurable change (0.32 %): the default is already about as close to the values as this market gets. Use it to show that more value traders stop helping |
| **News day** | `rate_symbol: 60`, `rate_sector: 3`; composition: `news-trader: 15` | 66 headlines in a trading day instead of 6; one-minute moves 0.85 % (+17 %); prices follow the values as closely as before (0.45 %) |
| **Rumour mill** | `rumour_share: 0.7`, `rumour_confirm: 0.5`; remove the market maker | Without the market maker alone prices sit 1.3 % from the values; the rumours add a little (1.4 %, 35 symbols more than 2 % off instead of 27). At 6 headlines a day that is four or five rumours: raise `rate_symbol` for more |
| **Auction lesson** | composition: `auction-player: 20` | Opening auction volume +60 %, and 298 of 300 symbols open with a cross instead of 243; continuous trading unchanged |
| **Thin market** | `per_agent: 5`; `--qty 50` | 38 symbols trade in a minute instead of 240, 60 in five minutes; prices 0.9 % from the values: symbols sit still, then jump |

After a change, give the market a full trading day — the budget needs a few minutes, and each kind of trader needs its own symbols to warm up — before you judge it.

## 7.8 Measuring a change

1. Run a day with the old settings; note the RMS gap (`pm-market-sim --status --id OPS01` shows the extreme ones; the admin API all of them), typical spreads in book-gui, actions a second, and the day's ranges (`pm-stats-cli daily`).
2. Change one thing. Restart.
3. Run a day; compare the same numbers.

Accelerated time (Accelerated Time) shortens step 3, with one caveat: at high speed the traders have fewer orders per trading day to follow the values, so compare settings at the same speed.

CHAPTER 8

# AI Traders

**Learning objectives**

After reading this page you will understand:

- What `pm-ai-trader` and `pm-ai-swarm` are and how they reach the exchange
- What a *preset* is, and which built-in presets there are
- How to run one AI trader, a swarm of them, and your own preset
- Which traders need the market model (`pm-market-sim`) and why

**Prerequisites**: Running the Exchange, ALF Gateway — AI traders log on through `pm-alf-gwy` like any ALF client, so their IDs must be participants in the deployed configuration.

## 8.1 What are the AI trader processes?

| Process | Purpose |
|---|---|
| `pm-ai-trader` | One AI trader, logged on as one participant |
| `pm-ai-swarm` | Many AI traders — tens to hundreds — in a few worker processes, each trader logged on as its own participant |

An AI trader is a participant like a student desk. It logs on to `pm-alf-gwy` with its own ID (`AI001`, `AI002`, …, role `TRADER`), sends its orders over that ALF session, and reads market data from the engine's public feed. Everything it does is visible to everyone else exactly as a person's orders are: in the books, the trades, the audit trail. When its session ends, the participant's `disconnect_behaviour` applies (`CANCEL_ALL` for the example AI participants).

The `s150-*` examples register `AI001`–`AI020`, the `s300-load` example `AI001`–`AI500` and the `s300-sim` example `AI001`–`AI100`; all ship a `swarm.yaml` and a `market_sim.yaml`. `s300-sim` is the long-running simulated market of the AI Traders and Market Simulation book.

## 8.2 Presets

What a trader does is set by its *preset*, four parts:

| Part | Decides | Choices |
|---|---|---|
| **Strategy** | *What* it wants to trade, which way, how urgently | `noise`, `trend`, `reversion`, `value`, `news` |
| **Execution** | *How* a wish becomes orders | `passive`, `marketable`, `sweep`, `iceberg`, `twap` |
| **Tempo** | How often it decides, and how big its orders are | `many-small`, `aggressive`, `cautious`, `few-large`, or your own |
| **Risk** | Position limit, live orders per symbol, order age, protective orders | per preset |

How the AI Traders Decide explains each in detail. The built-in presets (`pm-ai-trader --list-presets`):

| Preset | Strategy | Execution | Tempo | Protection | Character |
|---|---|---|---|---|---|
| `noise-retail` | noise | passive | many-small | — | Uninformed retail flow: small random orders resting near the touch |
| `market-taker` | noise | sweep (MARKET) | cautious | — | Impatient noise trader that takes liquidity |
| `iceberg-seller` | noise | iceberg | few-large | — | Large orders showing a fifth of their size |
| `patient-liquidity` | noise | passive (3-15 ticks deep) | own | — | Small orders waiting deep in the book for up to half an hour: book depth |
| `scalper` | trend | sweep (IOC) | aggressive | trailing stop | Fast momentum trader |
| `trend-follower` | trend | marketable | cautious | stop | Follows momentum with marketable limits |

| Preset | Strategy | Execution | Tempo | Protection | Character |
|---|---|---|---|---|---|
| `block-taker` | trend | sweep (FOK) | few-large | stop-limit | Takes whole blocks or nothing |
| `contrarian` | reversion | passive (GTC) | cautious | bracket (OCO) | Fades moves patiently |
| `auction-player` | reversion | passive | cautious | — | Trades mostly in the opening and closing auctions (ATO/ATC) |
| `value-investor` | value | passive | cautious | — | Buys below its view of the true value, sells above |
| `institutional` | value | twap | few-large | — | Large parent orders toward the value, worked in slices over minutes |
| `news-trader` | news | marketable | aggressive | — | Trades headlines within seconds, in their direction |

Between them the presets send every order type and time in force the exchange has: LIMIT, MARKET, IOC, FOK, ICEBERG, STOP, STOP_LIMIT, TRAILING_STOP, OCO; DAY, GTC, ATO, ATC.

### 8.2.1 Value and news need the market model

The `value` and `news` strategies trade on what `pm-market-sim` publishes: a hidden *true value* per symbol, and news. Without it running, `value-investor`, `institutional` and `news-trader` stand idle — they log on but send nothing. The true values never reach students: only the AI traders, `pm-mm-bot --anchor-sim` and ADMIN keys see them. The headlines are public.

---

## 8.3 Running one AI trader

```
pm-ai-trader --id AI001 --preset noise-retail --symbols AAPL,MSFT -v
```

| Flag | Default | Meaning |
|---|---|---|
| `--id` | — | Participant ID to log on as |
| `--preset` / `--preset-file` | `noise-retail` | A built-in preset, or your own preset YAML |
| `--list-presets` | — | Print the built-in presets and exit |
| `--symbols` | every symbol | Comma-separated symbols to trade |
| `--seed` | `1` | Same seed, same decisions given the same market |
| `--alf-host` / `--alf-port` | engine host / deployed `alf_gateway.port` | Where `pm-alf-gwy` listens |
| `--duration` | `0` | Seconds to run; `0` = until Ctrl+C |
| `-v` | off | Log a status line a minute and every session change |

Stop it with Ctrl+C: it closes its session, and its orders are cancelled.

---

## 8.4 Running a swarm

```
pm-ai-swarm --swarm docs/examples/ref_data/s150-nominal-setup/swarm.yaml -v
```

`swarm.yaml` says how many traders, which presets in what mix, which symbols, and how many order actions a second the whole swarm may send:

```
version: 1
agents: {prefix: AI, start: 1, count: 20}
budget: 40            # order actions per second, whole swarm
seed: 1000            # agent i gets seed + i
symbols:
  per_agent: 150      # every agent trades every symbol
composition:          # preset: weight
  noise-retail: 25
  value-investor: 22
  news-trader: 5
  institutional: 8
  scalper: 8
  trend-follower: 10
  contrarian: 10
  iceberg-seller: 4
  market-taker: 4
  block-taker: 2
  auction-player: 2
```

Command-line flags override the file (`--count`, `--budget`, `--workers`, `--presets noise-retail:3,scalper:1`, `--symbols-per-agent`, `--duration`). The operator's swarm runbook has every key, how to size a swarm, the

logs, the daily summaries and what happens when a process goes down.

---

## 8.5 Your own preset

Copy a built-in preset and change it. A preset file names its four parts:

```
name: nervous-scalper
description: A scalper with a tight trailing stop and small positions.
strategy: {name: trend, strength: 0.9, sample: 4, urgency: 0.9}
execution: {style: sweep, sweep_type: IOC, cross_ticks: 1}
tempo: {decisions_per_min: 30, size_min: 10, size_max: 50}
risk: {max_position: 200, max_live_orders_per_symbol: 2, max_order_age_sec: 15,
       protection: trailing, protection_pct: 0.005}
```

```
pm-ai-trader --id AI001 --preset-file nervous-scalper.yaml -v
```

In a `swarm.yaml`, a composition key that ends in `.yaml` is a preset file, relative to the `swarm.yaml`. Unknown keys and out-of-range values are refused before anything starts, naming the file and the key.

---

## 8.6 Reading its output

With `-v`, each trader (or each swarm worker, for its traders together) logs one line a minute:

```
[AI001] t=120s session=CONTINUOUS actions=96 (0.8/s) live_orders=2 fills=31 rejects=0
```

`actions` counts new orders and cancels. A trader whose orders keep being refused pauses itself for a few seconds (the *reject breaker*); a cancel that lost the race with a fill, or an order in flight when the market closed, does not count. After each close the swarm writes a summary per trader — orders, fills, position, P&L — see the runbook.

---

## 8.7 See also

- How the AI Traders Decide — strategies, execution styles, tempo and risk in detail
- Running the AI-Trader Swarm — the operator's runbook
- The Market Model (`pm-market-sim`) — true values and news
- The Market-Maker Bot (`pm-mm-bot`) — liquidity for the AI traders to trade against
- Order Types
- AI Traders & News — classroom exercises

CHAPTER 9

# 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.

## 9.1 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.




## 9.2 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).

## 9.3 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-agen 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`.

### 9.3.1 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 |

| Flag | Default | `swarm.yaml` key | Meaning |
|---|---|---|---|
| `--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 |

### 9.3.2 `swarm.yaml`

| Key | Default | Meaning |
|---|---|---|
| `version` | — | Must be `1` |
| `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 |

| Key | Default | Meaning |
|---|---|---|
| `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.

### 9.3.3 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%.

## 9.4 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.

## 9.5 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 |

## 9.6 Stopping

`SIGTERM` or `Ctrl-C` stops every worker; each closes its ALF sessions, which applies the participants' `disconnect_beha` (`CANCEL_ALL` for the example AI participants). Workers get 10 seconds before they are killed.

## 9.7 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 10

# 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.

## 10.1 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.

## 10.2 The model

For each symbol, one step of trading time $\Delta$ (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.

## 10.3 `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
```

## 10.4 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.json` 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%
...
```

## 10.5 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.

## 10.6 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 |

### 10.6.1 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
```

### 10.6.2 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.

### 10.6.3 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-surpr` `rumour-confirmed.yaml`, `rumour-retracted.yaml` and `chip-race.yaml`. The AI Traders & News training chapter uses them.

CHAPTER 11

# A Complete Set-up by Hand

**Learning objectives**

After reading this chapter you will be able to:

- Run the whole simulation straight on your machine, without containers
- Name every process it runs and what each one is for
- Find every file it reads and writes
- Make your own variant of the simulation — other symbols, other traders, another market model — and run it

**Prerequisites**: Python 3.14 and `pipx` (Quick Start: Install and Start covers installing them).
The container in Ten-Minute Start is this chapter packaged. Doing it by hand takes a few more commands, but you see every part, and you can change anything.

## 11.1 Install and start

```
pipx install edumatcher
export EDUMATCHER_DATA_DIR=~/edumatcher-sim     # where everything is kept
pm-setup --config s300-sim                      # deploy the example
pm-opctl-cli start ai-sim                       # start the 17 processes
```

`pm-setup --config s300-sim` creates the data directory and deploys the bundled `s300-sim` example into it: the engine configuration and the market model's `market_sim.yaml`. `pm-opctl-cli start ai-sim` starts the process profile. Every `pm-*` command finds the data directory through `EDUMATCHER_DATA_DIR`, so set it in every terminal you use (or in your shell profile).

**Checkpoint.**

```
pm-opctl-cli list
```

prints one row per process; all should be `running` within half a minute. If `mm-bot` shows `dead`, it started before the engine was ready — answer `y` when `list` offers to restart it (the container does this by itself).

**Time zone**

The scheduler follows your computer's local clock. `s300-sim` keeps 09:00–16:05 on the Swedish calendar; outside Sweden, start the profile with `TZ=Europe/Stockholm pm-opctl-cli start ai-sim` to keep Stockholm hours, or leave it and trade 09:00–16:05 *your* time.
To stop everything:

```
pm-opctl-cli stop
```

**Web applications for the hand-built simulation**

The browser applications come as ready-built containers that you can run beside this simulation. Start the simulation with `EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0` exported (the dashboard needs it), then in a shell with `EDUMATCHER_DATA_DIR` set as above:

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

See Web applications with a pipx exchange. The trading and read-only keys are the ones in your `s300-sim` configuration. Do not run the container simulation at the same time: both use the same ports.

## 11.2 Accelerated time by hand

The scheduler reads two environment variables (the container sets them from `EM_SPEED` and `EM_START`):

```
EDUMATCHER_SCHEDULER_SPEED=24 EDUMATCHER_SCHEDULER_START=2026-11-02T08:55 \
    pm-opctl-cli start ai-sim
```

Accelerated Time explains what this does.

## 11.3 What runs

The `ai-sim` profile is the full exchange plus three simulation processes. `pm-opctl-cli show` prints where the profiles are defined; this is what it starts:

| Process | Command | Role |
|---|---|---|
| `log` | `pm-log-srv` | Collects every process's log into `log.db` (kept 30 days) |
| `audit` | `pm-audit --verbose` | The audit trail of every message, `audit.log` |
| `stats` | `pm-stats --verbose --keep-days 7 --keep-order-events-days 0.5 --human-gateways 'TRADER*' --keep-human-order-events-days 7` | Daily statistics (kept for good), every trade (kept 7 days) and every order event (kept 12 hours) in `stats.db` |

| Process | Command | Role |
|---|---|---|
| `clearing` | `pm-clearing --verbose --print-every 0 --retention-days 7` | Positions and settlement, `clearing.db`: a week of individual trades (pruned at every close), daily summaries for good; without its console P&L table, which it would redraw every 100 trades |
| `engine` | `pm-engine` | Matching: the 300 order books |
| `scheduler` | `pm-scheduler --daily --verbose` | The trading calendar |
| `market-data-gwy (md-gwy)` | `pm-md-gwy --verbose` | Market data for terminal-gui and other clients |
| `post-trade-gwy (ralf-gwy)` | `pm-ralf-gwy --verbose` | Post-trade feed |
| `drop-copy-gwy (dc-gwy)` | `pm-dc-gwy --verbose` | Drop copies |
| `api-desk-gwy (api-gwy)` | `pm-api-gwy --verbose --instance desk` | REST/WebSocket API for trading (port 8080) |
| `api-dashboards-gwy (api-gwy)` | `pm-api-gwy --verbose --instance dashboards` | Read-only API for the GUIs (port 8081) |
| `alf-gwy` | `pm-alf-gwy --verbose` | The order gateway the AI traders and trader consoles use |
| `balf-gwy` | `pm-balf-gwy --verbose` | The binary order gateway |
| `index-srv` | `pm-index --verbose` | Index calculation |
| `mm-bot` | `pm-mm-bot --gateway-id MM01 --all-symbols --strategy symmetric --gap 0.20 --qty 200 --reissue-delay-ms 5000 --drift-ticks 15 --anchor-sim 0.3` | The market maker, anchored to the true values |
| `market-sim` | `pm-market-sim --verbose` | The market model: true values and news |
| `ai-swarm` | `pm-ai-swarm --swarm <examples>/s300-sim-setup/swarm.yaml --verbose` | The hundred AI traders |

The engine runs without `--verbose` here: with it, it logs every order it accepts, which on this market is tens of millions of log lines a month.

## 11.4 Where everything is

Everything lives under `$EDUMATCHER_DATA_DIR` (`/data` in the container, `./data-sim` on its host):

| Path | What | Grows? |
|---|---|---|
| `ref_data/engine_config.yaml`, `engine_config.json` | The deployed configuration (source and compiled) | no |
| `ref_data/market_sim.yaml` | The deployed market model | no |
| `market_sim_state.json` | The model's current values, saved every 10 s | no |
| `market_sim_news.db` | Published news and rumour events queried by `pm-news past`, and the model's value jumps | grows with each event |
| `emo/` | `pm-opctl` PID files and each process's console output (`<name>.log`) | slowly |
| `log.db` | Every process's log | up to 30 days, then flat |
| `audit.log`, `audit/` | The audit trail: today in `audit.log`, closed day parts and one zstd archive per past day in `audit/` | parts never deleted; archives forever unless `pm-audit --keep-days N` |
| `stats.db` | Daily statistics; trades, order events, snapshots | rollups forever, trades and snapshots 7 days, order events about an hour |
| `clearing.db` | Clearing events and positions | trades 7 days, daily summaries and positions for good |
| `ai_swarm/<date>-<worker>.jsonl` | Each AI trader's results, one line per trader per day | about 50 KB a day |
| `gtc_orders.json`, `gtc_combos.json` | Orders that survive the close | no |

Running It for Weeks has the measured sizes.

## 11.5 Your own variant

The bundled example is read-only; make a copy and change that. Find it with the Python of the environment `pipx` installed EduMatcher into:

```
EM_PY="$(pipx environment --value PIPX_LOCAL_VENVS)/edumatcher/bin/python"
"$EM_PY" -c "from edumatcher.config_deploy import examples_root; print(examples_root())"
```

and copy its `s300-sim-setup` directory:

```
cp -r "$("$EM_PY" -c 'from edumatcher.config_deploy import examples_root; print(examples_root())')/s300-sim-setup
```

| File | Change it to change | Deploy with |
|---|---|---|
| `engine_config.yaml` | Symbols, participants, the trading hours, the market maker's obligations | `pm-config-deploy ~/my-sim/engine_config.yaml` |
| `market_sim.yaml` | Volatility, correlation, jumps, news rates (The True Value, News and Rumours) | the same command: it deploys the `market_sim.yaml` beside the engine configuration too |
| `swarm.yaml` | How many traders, which kinds, how busy (Tuning the Traders) | point the profile at it (below) |

The profile is the one piece that is not a file yet. Write the built-in profiles out, so you can edit them:

```
pm-opctl-cli init          # writes $EDUMATCHER_DATA_DIR/emo-config.yaml
```

In `emo-config.yaml`, find the `ai-sim` profile's `ai-swarm` entry and change its `--swarm` path to `~/my-sim/swarm.yaml` (spelled out in full — `/home/you/my-sim/swarm.yaml`). From now on `pm-opctl-cli` uses that file's profiles instead of the built-in ones; the market maker's flags, the retention of `pm-stats` and everything else are yours to edit there too.

Then restart:

```
pm-opctl-cli stop
pm-opctl-cli start ai-sim
```

**Change one thing at a time**

Restart, watch a trading day, compare. Tuning the Traders says what each setting is expected to do and how to see it.

## 11.6 Running the pieces one at a time

To see how the parts fit, start the trading core yourself, each command in its own terminal (with `EDUMATCHER_DATA_D` set):

```
pm-engine
pm-alf-gwy -v
pm-scheduler --daily -v
pm-market-sim -v
pm-mm-bot --gateway-id MM01 --all-symbols --strategy symmetric --gap 0.20 \
    --qty 200 --reissue-delay-ms 5000 --drift-ticks 15 --anchor-sim 0.3
pm-ai-swarm --swarm ~/my-sim/swarm.yaml -v
```

That is a trading market without statistics, clearing, audit or the GUIs' APIs. Stop the swarm and the market keeps its quotes; stop the market maker and watch the books thin out and the prices drift from the values; stop `pm-market-sim` and watch the value and news traders fall silent.

CHAPTER 12

# The Simulation Container

**Learning objectives**

After reading this chapter you will be able to:

- Explain what is in the `edumatcher-ai-sim` image and how it differs from the plain `edumatcher` image
- Read `compose.ai-sim.yaml` and change what it runs through its environment variables
- Run the simulation on a server for a class, safely
- Upgrade, reset, back up and build the image yourself

**Prerequisites**: Ten-Minute Start.

## 12.1 Two images, one exchange

| Image | What it is |
|---|---|
| `ghcr.io/johan162/edumatcher` | The exchange. Starts the `s10-basic` example with the `default` profile unless told otherwise |
| `ghcr.io/johan162/edumatcher-ai-sim` | The same image with different defaults: the `s300-sim` example, the `ai-sim` profile, Stockholm time, dead processes restarted |

Both are built from the same `Dockerfile` (`deployment/docker/`) — the simulation is its last stage, a handful of environment variables on top of the exchange — and published together with every release, for `amd64` and `arm64`, tagged with the release version and `latest`. Nothing in the simulation image is a fork: every program in it is the ordinary `edumatcher` package.

```
FROM runtime AS ai-sim

ENV EM_CONFIG=s300-sim \
    EM_PROFILE=ai-sim \
    EM_RESTART_DEAD=1 \
    EM_SPEED=1 \
    TZ=Europe/Stockholm
```

## 12.2 What happens when it starts

The image's entry point (`entrypoint.sh`) does five things:

1. **Deploys the configuration** into `/data` — the `EM_CONFIG` example, or your own file if `EM_CONFIG_FILE` is set. On later starts it keeps what is deployed, unless `EM_CONFIG` names a different example than last time.
2. **Sets the clock**: with `EM_SPEED` other than 1, or `EM_START`, it passes them on to the scheduler (as `EDUMATCHER_SCHEDULER_SPEED` and `EDUMATCHER_SCHEDULER_START`).
3. **Starts the profile** with `pm-opctl-cli start ai-sim`. The profile's scheduler runs `pm-eod run --pending` half a minute after every close, so each simulated day is settled and written up on its own — see End of day below.
4. **Watches it**: once a minute it restarts any process of the profile that has died, and says so in the container log (`docker logs edumatcher-sim`):

   ```
   ==> 2026-11-03 09:00:41 Restarting: mm-bot
   ==> 2026-11-03 09:00:41   started mm-bot (pid 812): pm-mm-bot --gateway-id MM01 ...
   ```

   The market maker usually appears here once, a minute after the first start: it gives up if it starts before the engine is ready.
5. **Stops cleanly** on `docker stop` (or `compose down`): it stops every process of the profile, which takes up to a minute.

The image's health check asks whether the engine accepts connections, so `docker ps` shows `healthy` once the exchange is up — whether or not the market is open.

## 12.3 `compose.ai-sim.yaml`

The compose file runs six containers in one compose project, `edumatcher-sim`:

| Container | Image | Host port | |
|---|---|---|---|
| `edumatcher-sim` | `edumatcher-ai-sim` | 5560, 5565, 5570, 5580, 5590, 5600, 8080, 8081 | the exchange and the simulation |
| `edumatcher-sim-terminal-gui` | `edumatcher-terminal-gui` | 8090 | market-data terminal |
| `edumatcher-sim-log-gui` | `edumatcher-log-gui` | 8091 | log viewer |
| `edumatcher-sim-trader-gui` | `edumatcher-trader-gui` | 8093 | trading screen |
| `edumatcher-sim-book-gui` | `edumatcher-book-gui` | 8094 | order-book viewer |
| `edumatcher-sim-dash-gui` | `edumatcher-dash-gui` | 8095 | operator dashboard |

The GUIs reach the exchange over the compose network, at the host name `edumatcher`; the host ports are for you and your students' browsers and programs. The GUIs that read market history use the example's read-only API key, which the compose file already knows.

There are two copies of this file, and they describe the same six containers:

| | |
|---|---|
| `deployment/curl/compose.ai-sim.yaml` | What `install.sh --sim` fetches into `~/.edumatcher-sim`. Build-free: every image comes from GHCR. Its data directory is `./data` |
| `deployment/docker/compose.ai-sim.yaml` | The repository's own, which can also *build* the image (`make build-sim`). Its data directory is `./data-sim` |

The rest of this chapter uses the installed one and `./edumatcher.sh`.

### 12.3.1 Settings

Everything is set with environment variables, in the `.env` file next to the compose file (the installer writes one) or on the command line, as in `EM_SPEED=24 docker compose -f compose.ai-sim.yaml up -d`:

| Variable | Default | Meaning |
|---|---|---|
| `EM_SPEED` | `1` | How many times faster than real time the trading calendar runs (Accelerated Time) |
| `EM_START` | — | Where an accelerated calendar starts, local time, e.g. `2026-11-02T08:55` |
| `TZ` | `Europe/Stockholm` | The time zone of the trading hours |
| `EM_CONFIG_FILE` | — | Your own engine configuration, as a path inside the container (below) |
| `EM_VERSION` | `latest` | Which release of all six images to run, e.g. `0.51.0` |
| `GHCR_OWNER` | `johan162` | Whose published images to pull |
| `BIND_ADDR` | `127.0.0.1` | Which host interface the ports are published on |
| `TERMINAL_GUI_PORT`, `LOG_GUI_PORT`, `TRADER_GUI_PORT`, `BOOK_GUI_PORT`, `DASH_GUI_PORT` | 8090, 8091, 8093, 8094, 8095 | Host ports of the GUIs |
| `PM_READONLY_API_KEY` | the `s300-sim` key | The read-only key the GUIs use; change it only with your own configuration |
| `AISIM_IMAGE` | the GHCR image | A different backend image, e.g. one you built |

**Pin a version for a course**

`latest` moves with every release. For a course that runs for weeks, put `EM_VERSION=<version>` in `.env` so a restart never brings a different exchange. `./edumatcher.sh update 0.51.0` does it for you.

### 12.3.2 Data

Everything the exchange writes is in the install's `data` directory, mounted at `/data`: `~/.edumatcher-sim/data` for an installed simulation, `./data-sim` next to the compose file in a checkout. It survives `stop`, `start`, reboots and upgrades.

| To | Do |
|---|---|
| Start over, with a fresh market | `./edumatcher.sh stop`, delete `data`, `./edumatcher.sh start` |
| Back it up | Stop the stack, copy `data`, start it again |
| Look at a file | It is on your disk: `data/ai_swarm/`, `data/stats.db`, … |

A backup taken while the exchange runs can catch a database mid-write; stop it first.

The first start also builds `data/companies.db`: made-up company profiles for the 300 tickers (name, industry, CEO, founder story, financials, P/E, …). Look them up with `./edumatcher.sh shell pm-compdb-cli show AAPL`, or list and filter them with `pm-compdb-cli list`. It is built only when missing, so companies you add survive restarts; delete it to get the bundled set back. The reference manual's pm-compdb-cli section explains where the source files are and when the entrypoint builds the database.

### 12.3.3 End of day

After each close the scheduler starts `pm-eod`, which turns the day's gains and losses into a **statement for every trader** (AI001–AI100, MM01 and the desks), keeps a running **balance** for each, and backs up the databases. Nothing needs configuring. The fictional currency means the balances are simply a scoreboard: who has made money in the simulated market so far.

| Where | What |
|---|---|
| `data/settlement/<date>/leaderboard.csv` | Everyone ranked by balance, best first |
| `data/settlement/<date>/statements/AI042.md` | One trader's statement (also `.json`, `.csv`) |
| `data/settlement/<date>/summary.md` | The day in one page, including the proof that the amounts sum to zero |
| `data/backup/<date>/` | Compressed copies of `clearing.db` and `eod.db`, kept 2 days. `stats.db` is left out: it is large and mostly disposable order events |

Ask for the current standings at any time, from inside the container:

```
./edumatcher.sh shell pm-eod balances
./edumatcher.sh shell pm-eod status
```

There is no `roster.yaml`, so statements carry the gateway names (`AI042`) rather than people's names. To label the desks `TRADER01`–`TRADER10` for a class, add a `ref_data/roster.yaml` in the data directory (see Closing the Day). A day settles only when the market has actually closed, so a weekend or a holiday produces no statements.

### 12.3.4 The commands you will use

From the install directory (`cd ~/.edumatcher-sim`):

| Task | Command |
|---|---|
| Start | `./edumatcher.sh start` |
| Stop | `./edumatcher.sh stop` |
| Process table, container status | `./edumatcher.sh status` |
| A shell in the exchange | `./edumatcher.sh shell` |
| One command in it | `./edumatcher.sh shell pm-market-sim --status --id OPS01` |
| The container log (starts, restarts) | `./edumatcher.sh logs edumatcher` |
| The web addresses | `./edumatcher.sh urls` |
| Which directory is behind each container path | `./edumatcher.sh mounts` |
| Accelerate the calendar | `./edumatcher.sh speed 24 2026-11-02T08:55`, then `restart` |
| Upgrade | `./edumatcher.sh update` (or `update 0.51.0`) |
| Remove it | `./edumatcher.sh uninstall` (`--data` to delete the data too) |

The script is a wrapper around `docker compose` and `docker exec`; anything it does you can do by hand — `docker compose -f compose.ai-sim.yaml up -d`, `docker exec edumatcher-sim pm-opctl-cli list --no-restart`, `docker logs -f edumatcher-sim`.

From a clone of the repository, `deployment/docker/Makefile` wraps the same things: `make up-sim`, `make down-sim`, `make sim-status` (process table and disk), `make sim-shell`; `make up-sim SPEED=24 START=2026-11-02T08:` for an accelerated run.

## 12.4 Changing what it runs

### 12.4.1 Your own configuration

The container deploys a configuration of your own when `EM_CONFIG_FILE` names one. Start from the deployed one:

```
mkdir -p my-sim
./edumatcher.sh shell cat /data/ref_data/engine_config.yaml > my-sim/engine_config.yaml
./edumatcher.sh shell cat /data/ref_data/market_sim.yaml    > my-sim/market_sim.yaml
# edit my-sim/market_sim.yaml (or engine_config.yaml), then:
./edumatcher.sh config ./my-sim/engine_config.yaml
./edumatcher.sh restart
```

`config` copies both files into the install's `./config`, which is mounted read-only at `/config`, and sets `EM_CONFIG_FILE` in `.env` so it stays set. If the configuration is refused, the container log says why (`./edumatcher.sh logs edumatcher`) — check it first with `./edumatcher.sh shell pm-cverifier /config/engine_config.yaml`.

A deployed configuration stays deployed: clearing `EM_CONFIG_FILE` alone does not bring `s300-sim` back. Clear it, then redeploy the bundled one and restart:

```
./edumatcher.sh shell pm-setup --config s300-sim --force
./edumatcher.sh restart
```

### 12.4.2 Your own traders or process flags

The traders' mix and the processes' flags are in the process profile. Write the built-in profiles out once and edit the copy:

```
./edumatcher.sh shell pm-opctl-cli init
```

`init` writes `data/emo-config.yaml`, which from then on replaces the built-in profiles. To use your own `swarm.yaml`, put it in `./config/` and change the `ai-sim` profile's `ai-swarm` command to `--swarm /config/swarm.yaml` restart the stack. Delete `emo-config.yaml` to go back to the built-in profiles (and get the profiles of a newer release after an upgrade).

Tuning the Traders says what to change.

## 12.5 Opening it to a classroom

By default every port is published on `127.0.0.1`: only the machine itself can reach the exchange. To let students in from the local network, set `BIND_ADDR=0.0.0.0` in `.env` and restart, and first:

- **Change the API keys.** The example's keys are printed in this book. Use your own configuration (above) with new `api_key` values for every participant, and give each student only their own.
- **Keep it on a network you trust.** Nothing here is encrypted: API keys and orders travel in clear text. Put a TLS reverse proxy in front of the web ports if the network is not yours.
- **Mind the order gateways.** The API (8080) checks a key; the order gateways (5560, 5565) identify a participant by its ID alone, so anyone who reaches them can log on as any participant — an AI trader included, which then loses its session. Publish them only on a network you trust, or take their lines out of the compose file's `ports` and let students trade through the API and trader-gui.

## 12.6 Resources

Measured on a 2-vCPU Linux machine, at real speed during continuous trading (see Running It for Weeks for the details and for disk):

| | |
|---|---|
| CPU, whole exchange | about half of one core while the market is open; almost nothing while it is closed |
| Memory | about 1 GB for the exchange container (the sum of the processes' resident memory), flat over the day |
| Busiest processes | `pm-engine` and the two API gateways together, a tenth of a core each; then `pm-stats`, `pm-audit`, the swarm worker, `pm-alf-gwy` and `pm-mm-bot`, about a twentieth each |

A laptop that runs a browser comfortably runs the simulation.

## 12.7 Upgrading

```
./edumatcher.sh update          # the newest release
./edumatcher.sh update 0.51.0   # ...or an exact one
```

The data directory is kept, and with it the deployed configuration and the market's history. The release notes say when a release changes the `s300-sim` example; to take the new one, start over with a fresh data directory, or redeploy with the new example while keeping your history: `./edumatcher.sh shell pm-setup --config s300-sim --force`, then restart.

## 12.8 Building the image yourself

From a clone of the repository:

```
cd deployment/docker
make build-sim          # builds edumatcher-ai-sim:local from this checkout
make up-sim LOCAL=1     # runs it with the GUI images from GHCR
```

`make build-sim` builds a wheel from the checkout (it needs `poetry`) and the image's `ai-sim` stage. Without `make`:

```
docker build --target ai-sim -t edumatcher-ai-sim:local deployment/docker
AISIM_IMAGE=edumatcher-ai-sim:local docker compose -f deployment/docker/compose.ai-sim.yaml up -d
```

(Without a wheel in `deployment/docker/.wheel/`, the build installs the latest `edumatcher` from PyPI.)

CHAPTER 13

# Accelerated Time

**Learning objectives**

After reading this chapter you will be able to:

- Run the simulation's trading calendar faster than real time, and pick a speed for what you want to see
- Say exactly what speeds up and what does not
- Predict how a faster calendar changes the market you watch
- Avoid the traps: restarts, dates in the statistics, auctions at high speed

## 13.1 One setting

```
./edumatcher.sh speed 24 && ./edumatcher.sh restart
```

`EM_SPEED=N` runs the trading calendar N times faster than the clock on the wall. At 24 a calendar day passes in an hour of wall time: the market opens, trades through the day, closes, sleeps through the night and opens again an hour later. Weekends and Swedish holidays pass too, at the same speed. `EM_SPEED=1` (the default) is real time.

`EM_START` says where the faster calendar starts, as local time — the second argument:

```
./edumatcher.sh speed 24 2026-11-02T08:55 && ./edumatcher.sh restart
```

Without it the calendar starts at the real present moment. A start a few minutes before 09:00 on a weekday is the usual choice: the market opens almost at once.

By hand (without the container) the same two settings are the scheduler's `--speed` and `--start` flags, or the environment variables `EDUMATCHER_SCHEDULER_SPEED` and `EDUMATCHER_SCHEDULER_START`; see A Complete Set-up by Hand.

## 13.2 Choosing a speed

The trading day is 09:00–16:05: a 25-minute pre-open, a 5-minute opening auction, 6.5 hours of continuous trading, a 5-minute closing auction.

| EM_SPEED | A trading day (09:00–16:05) | Continuous trading | An auction | A week | 30 trading days | Good for |
|---|---|---|---|---|---|---|
| 1 | 7 h 5 min | 6 h 30 min | 5 min | 7 days | 6 weeks | A live market alongside a course |
| 10 | 42.5 min | 39 min | 30 s | 16.8 h | 4.2 days | A whole day inside one lesson |
| 24 | 17.7 min | 16.3 min | 12.5 s | 7 h | 42 h | A day an hour; a week over a working day |
| 60 | 7.1 min | 6.5 min | 5 s | 2.8 h | 17 h | Several days in a lesson; overnight runs |
| 600 | 42.5 s | 39 s | 0.5 s | 17 min | 1.7 h | Soak tests: a month of closes in an afternoon |

(Thirty trading days are six calendar weeks, weekends included.)

## 13.3 What speeds up, and what does not

Only the **calendar** is faster. Everything that happens *during* trading runs on the wall clock.

| Faster (calendar time) | Not faster (wall clock) |
|---|---|
| The open, the auctions, the close, nights, weekends, holidays | How often each AI trader decides, how many orders a second the swarm sends |
| How far the true values move in a day: a compressed day moves as much as a real one | How long an order rests before its trader cancels it |
| How many headlines a trading day brings, and how soon a rumour resolves | The market maker's quoting rhythm |
| | Heartbeats, time-outs, every log and file timestamp |

So a compressed day is a **short day, not a fast day**. At `EM_SPEED=24` the traders send their usual orders a second, but continuous trading lasts 16 minutes, so a day has a twenty-fourth of the orders of a real one. Meanwhile the values move a whole day's worth in those 16 minutes. The consequences you can see:

- **Prices lag the value more.** Value traders and the market maker have fewer orders to pull the price after a value that moves 24 times faster per minute. Measured with the `s300-sim` set-up: the typical gap between price and value (the RMS of ln(price / value)) was about 0.3 % at real speed, about 0.45 % at `EM_SPEED=24` and about 3.5 % at 600.
- **News comes thick and fast.** Six symbol headlines a trading day is one every three minutes of wall time at 24, one every 7 seconds at 600. Rumours that resolve after 8–60 trading minutes resolve after 20 s–2.5 min at 24.
- **Auctions get thin.** A 5-minute auction lasts 12.5 s at 24 and half a second at 600 — barely time for the auction players to send their orders. Opening and closing prices at high speed come mostly from the market maker's quotes.
- **Order ages feel longer.** A trader that cancels after 60 s keeps an order for a sixteenth of the continuous session at 24, and for longer than the whole session at 600.

For teaching, 10 to 60 keeps a market that looks right; above that, use it to test that the exchange keeps running, not to look at prices.

## 13.4 Traps

**Every restart starts the calendar again.** The accelerated clock begins at `EM_START` (or now) whenever the scheduler starts — after `compose down/up`, a reboot, or the container restarting a dead scheduler. With `EM_START` set, the market goes back to that morning. For a long accelerated run, set `EM_START` for the first start only, then remove it (from `.env`, or leave `START=` off `make up-sim`) and run the same `up` command again, so Compose recreates the container without it; the data stays. A container keeps the settings it was created with, so until then every restart still goes back to `EM_START`. Later restarts begin at the real present, which never goes back.

**Dates are wall-clock dates.** Statistics, clearing and the AI traders' daily results are filed by the date the wall clock shows, because every message is time-stamped with real time. At `EM_SPEED=24` a whole compressed week lands on one real date: `pm-stats-cli daily` shows one row per symbol for that date, with the week's high and low; the AI swarm's daily summaries (`ai_swarm/<date>-<worker>.jsonl`, by UTC date) hold one line per trader per *compressed* day, numbered `day 1`, `day 2`, … in the file of the real date. If you need true daily statistics, run at real speed.

**Load is per wall second.** A faster calendar does not make the exchange work harder: the orders a second are the same. The disk grows by the same amount per *wall-clock* day at any speed (see Running It for Weeks), so a compressed month costs the disk of the real days it took to run.

## 13.5 A lesson plan at speed 10

1. Before class: start with `EM_SPEED=10` and `EM_START` set to a weekday at 08:55, about five minutes before you want the open.
2. Pre-open and opening auction (3 minutes): watch the indicative prices form in book-gui; open the books at the open.
3. Continuous trading (39 minutes): inject a headline at minute 5 (News and Rumours), a rumour at minute 15, confirm or retract it at minute 20.
4. Closing auction (30 seconds) and close: compare the day's moves with the values (`pm-market-sim --status --id OPS01` during trading).
5. After class: `down`, remove `EM_START` and `EM_SPEED` from `.env`, `up -d` — back to real hours.

CHAPTER 14

# Watching the Market

**Learning objectives**

After reading this chapter you will be able to:

- Use each browser application to watch the simulated market
- Use the command-line tools inside the container for what the browser does not show: the values, the traders' activity, the logs
- Read the AI traders' daily results and say which kind of trader is winning

**Prerequisites**: Ten-Minute Start.

## 14.1 The five windows

| Application | Address | Watch it for |
|---|---|---|
| **TapeDeck** (terminal-gui) | http://localhost:8090 | The whole market at a glance: movers, the trade tape, sessions and halts, the news |
| **Order Book Viewer** (book-gui) | http://localhost:8094 | One symbol's book, price level by price level, with its tape |
| **Trading GUI** (trader-gui) | http://localhost:8093 | Trading as a student desk; the instructor's screens with the `OPS01` key |
| **Log Operator Console** (log-gui) | http://localhost:8091 | What every process is doing; warnings and errors |
| **Operator Dashboard** (dash-gui) | http://localhost:8095 | Whether the simulation keeps up: every process's health, orders and trades per minute, and the AI traders as participants |

Each has its own chapter in the Participant or Operator's Guide (see See also); here is what to look at in a

simulated market.

### 14.1.1 TapeDeck: the market at a glance

Put it on the classroom screen. The **Market Overview** and **Movers** show which of the 300 shares are moving; a share that moved a lot with no headline in the **News** view probably jumped (an unannounced change in its value), or is being pushed by momentum traders. The **Trade Tape** shows who traded: `AI` ids are the swarm, `MM01` the market maker, `TRADER` ids your students. During the auctions the **Session & Halt Status Board** counts down to the next phase.

### 14.1.2 Order Book Viewer: one book up close

Pick a symbol and watch for a minute:

- The market maker's quotes are the 200-share levels 20 cents apart, around the middle of the book. When one is filled, that side stays empty for about 5 seconds, until the bot posts a fresh pair.
- The AI traders' orders are everything else: small passive orders from the noise traders near the touch, larger ones from value investors and institutions, an iceberg showing a fifth of its size.
- After a headline the book empties on one side within seconds as news traders sweep it, then refills at the new price.

### 14.1.3 Trading GUI: trade in it

Students log in with their desk keys (Ten-Minute Start) and trade like any participant: their orders meet the AI traders' orders in the same books. Logged in as `OPS01` you get the operator's screens — session control, halts, the kill switch — the same as on any exchange. Use them carefully on a market that is meant to keep running: a halted symbol stays halted until you resume it.

### 14.1.4 Log Operator Console: what the processes say

Filter by process. `pm-ai-swarm` writes one status line a minute and a summary after each close; `pm-market-sim` logs every headline it publishes; `pm-scheduler` every phase change. Warnings are worth a look; a steady trickle of `cancel … never answered` from the swarm means the exchange is overloaded (see Troubleshooting).

## 14.2 The command line

Everything below runs inside the exchange container. Either prefix each command with `docker exec edumatcher-s` or open a shell there once — `docker exec -it edumatcher-sim bash` — and type them as shown. (By hand, without the container, type them in any terminal with `EDUMATCHER_DATA_DIR` set.)

### 14.2.1 Is everything running?

```
pm-opctl-cli list --no-restart
```

One row per process, with its uptime and memory. `dead` rows are restarted by the container within a minute.

### 14.2.2 What is the market doing?

```
pm-log-cli query --process pm-ai-swarm --grep "actions=" --limit 3
```

```
[w1-AI001-AI100] t=480s session=CONTINUOUS actions=58029 (120.9/s) live_orders=2325 fills=27616 rejects=25
[w1-AI001-AI100] t=900s session=CONTINUOUS actions=91317 (101.4/s) live_orders=2368 fills=38669 rejects=25
```

`actions` counts new orders and cancels since the swarm started, and the rate in brackets is the average over that whole time, closed hours included. For the current rate, take two lines: (91317 − 58029) / (900 − 480) is 79 a second, the swarm's budget of 80. `live_orders` is how many orders the traders have working; `rejects` should stay a tiny share of the actions.

```
pm-board                # a rotating board of symbols: last, change, volume
pm-viewer -s AAPL       # one order book in the terminal
pm-ticker               # a scrolling ticker of the day's prices
```

### 14.2.3 How far from the values?

```
pm-market-sim --status --id OPS01
```

lists the shares whose prices are furthest from their true values (see The True Value).

### 14.2.4 The news

```
pm-news list            # headlines as they are published (Ctrl+C to stop)
```

### 14.2.5 The day's statistics

```
pm-stats-cli daily --symbol AAPL                    # open, high, low, close, volume per day
pm-stats-cli trades --symbol AAPL --limit 10        # the last trades
pm-stats-cli dates                                  # the trading dates recorded
```

`pm-stats` keeps the daily rows for good and individual trades for the last seven days (Running It for Weeks).

## 14.3 Who made money

After each close (five seconds after, so the closing auction's fills are in) the swarm writes one line per trader to `/data/ai_swarm/<date>-<worker>.jsonl` — on the host, `./data-sim/ai_swarm/`:

```
{"closed_at": "2026-11-03T15:05:05+00:00", "day": 2, "agent": "AI007",
 "preset": "value-investor", "submitted": 412, "fills": 233, "filled_qty": 9120,
 "cancels": 160, "rejected": 0, "killed": 0, "unseen_expiries": 0,
 "gtc_orders": 0, "gross_position": 1840, "pnl_realized": 412.10, "pnl_mtm": 87.40}
```

The counters are for that day; the profit and loss (`pnl_realized`, made on closed positions, and `pnl_mtm`, on the open position at the last price) since the trader last started. To total the latest P&L by kind of trader:

```
python3 - <<'EOF'
import glob, json
latest = {}
for path in sorted(glob.glob("/data/ai_swarm/*.jsonl")):
    for line in open(path):
        row = json.loads(line)
        latest[row["agent"]] = row
by_preset = {}
for row in latest.values():
    pnl = row["pnl_realized"] + row["pnl_mtm"]
    total, n = by_preset.get(row["preset"], (0.0, 0))
    by_preset[row["preset"]] = (total + pnl, n + 1)
for preset, (total, n) in sorted(by_preset.items(), key=lambda kv: -kv[1][0]):
    print(f"{preset:16} {n:3} traders  P&L {total:12.2f}  per trader {total / n:10.2f}")
EOF
```

In a market like `s300-sim` the informed traders — value investors, institutions, news traders — usually come out ahead, and the noise and momentum traders behind: that is what being informed is worth. If the order flips, the market is giving the uninformed an edge — worth a look at the settings (Tuning the Traders). It makes a good class discussion either way: who paid whom, and why.

## 14.4 For the instructor: the values through the API

The values are available to ADMIN keys through the API gateway, for your own dashboards and scripts:

```
curl -s -H "Authorization: Bearer key-ops01-3e34rs" \
    http://localhost:8080/api/v1/admin/sim
```

returns the model's state and every share's latest value; the WebSocket `ws://localhost:8080/api/v1/ws/admin/sim` streams them. Students' keys are refused.

## 14.5 See also

- TapeDeck, the Trading GUI, the Order Book Viewer, the Log Operator Console, the Operator Dashboard
- AI Traders & News — classroom exercises
- Running the AI-Trader Swarm — the swarm's logs and summaries in full

CHAPTER 15

# AI Traders & News

## 15.1 Objective

Run a swarm of AI traders over a market whose prices have a hidden *true value* behind them, then move that market with news: trade an earnings surprise, decide what to do with a rumour, watch a retraction unwind, and see how a breakthrough and a factory fire move two rival chipmakers.

**Background reading**

- AI Traders - Running the AI-Trader Swarm - The Market Model (`pm-market-sim`)

## 15.2 Prerequisites

- Chapters 01–13 completed.
- The `s150-nominal` example deployed (Exercise 1). It registers the AI traders `AI001`–`AI020`, the student desks `TRADER01`–`TRADER10`, the instructor `OPS01` (ADMIN) and the market maker `MM01`, and it ships the market model's `market_sim.yaml` and the swarm's `swarm.yaml`.

## 15.3 Background

Three processes make the AI market:

- **`pm-market-sim`** — the market model. Every symbol gets a *true value* that moves like a company's value does: with the market, with its sector, on its own, and now and then in a jump. Students never see it. It also publishes **news**: headlines that move the values.
- **`pm-ai-swarm`** — runs many AI traders, each logged on through `pm-alf-gwy` as its own participant (`AI001`, `AI002`, …) exactly like a student desk. `pm-ai-trader` runs a single one.
- **`pm-news`** — the instructor's news desk: publish a headline now, or play a timed scenario.

Each AI trader is a *preset*: what it trades on (its strategy), how it places orders, how fast, and its risk limits. `pm-ai-trader --list-presets` prints them all:

| Preset | Trades on | Character |
|---|---|---|
| `noise-retail` | nothing in particular | Small random orders resting near the touch |
| `patient-liquidity` | nothing in particular | Small orders waiting deep in the book, giving it depth |
| `market-taker`, `iceberg-seller` | nothing in particular | Takes liquidity with MARKET orders; hides size behind icebergs |
| `scalper`, `trend-follower`, `block-taker` | momentum | Buys what has been rising, with IOC, marketable or fill-or-kill orders and protective stops |
| `contrarian`, `auction-player` | reversion | Fades moves; trades the opening and closing auctions |
| `value-investor`, `institutional` | the true value | Buys below its own (imperfect) estimate of the value, sells above it |
| `news-trader` | headlines | Trades a headline within seconds, hard, in its direction |

The value and news traders are why prices end up near the values and react to news; the noise and momentum traders are why they never sit exactly on them. A headline has a public *sentiment* (−1 bad to 1 good) and a secret *impact* on the value. A **rumour** carries a *credibility* and moves no value until it is **confirmed**; a **retracted** rumour never moves anything.

## 15.4 Exercise 1: Deploy an Example with AI Traders

```
pm-config-deploy --example s150-nominal
```

This **replaces** the configuration you have built since chapter 01. From here on the participants are the ones the example defines — `TRADER01`–`TRADER10`, the instructor `OPS01` (ADMIN), `MM01`, and `AI001`–`AI020` — so `GW_ADMIN`, `TRADER03`, `MM_AAPL_01` and the other gateways of your own configuration no longer exist. Stop the consoles and bots that use them. To return to your own configuration later, deploy `engine_config.yaml` again.

The scenario and swarm files this chapter uses ship with EduMatcher, but not in your current directory: on a container or `pipx` install they live inside the installed package, and only in a repository checkout are they under `docs/examples`. Ask Python where they are, once per terminal, and use `$EX` in the commands below:

```
EX=$(python -c "from edumatcher.config_deploy import examples_root; print(examples_root().parent)")
ls "$EX/news"          # the four scenarios: chip-race, earnings-surprise, rumour-confirmed, rumour-retracted
```

(In the container shell `python` is the right interpreter; in a Poetry checkout use `poetry run python`; with `pipx` use `~/.local/pipx/venvs/edumatcher/bin/python`.)

Restart `pm-engine` (and reopen the market in a moment), then start the gateway the AI traders log on through, the market model, and a market maker for all 150 symbols that quotes around the model's true values, each in its own terminal:

```
pm-alf-gwy -v
pm-market-sim -v
pm-mm-bot --gateway-id MM01 --all-symbols --strategy passive --gap 0.20 --qty 200 --anchor-sim 0.3
```

Without `--anchor-sim` the market maker keeps refilling its quotes around its own mid, and no amount of informed trading moves the price.

Open the market from the operator console:

```
pm-admin --id OPS01
[OPS01|ADMIN]> SESSION|STATE=PRE_OPEN
[OPS01|ADMIN]> SESSION|STATE=OPENING_AUCTION
[OPS01|ADMIN]> SESSION|STATE=CONTINUOUS
```

`pm-market-sim` starts moving the values when continuous trading starts. Check it:

```
pm-market-sim --status --id OPS01
```

:material-checkbox-blank-outline: **Checkpoint:** `pm-market-sim: RUNNING … symbols=150`.

## 15.5 Exercise 2: One AI Trader

```
pm-ai-trader --id AI001 --preset noise-retail --symbols AAPL -v
```

The trader logs on, reads the AAPL book and starts resting small orders near the touch. Watch them arrive with `BOOK|SYM=AAPL` in the operator console. Stop it with Ctrl+C: its session closes and the participant's `disconnect_behaviour` (`CANCEL_ALL`) removes its orders.

:material-checkbox-blank-outline: **Checkpoint:** AI001's orders appear in the AAPL book and disappear when you stop it.

## 15.6 Exercise 3: The Swarm

```
pm-ai-swarm --swarm "$EX/ref_data/s150-nominal-setup/swarm.yaml" -v
```

`swarm.yaml` gives the twenty traders `AI001`–`AI020` a mix of every preset (`per_agent: 150` — every agent trades every symbol, not a slice of them), and sets a budget of about 40 order actions a second for the whole swarm. Once a minute the swarm logs a status line:

```
[w1-AI001-AI020] t=120s session=CONTINUOUS actions=4801 (40.0/s) live_orders=164 fills=1290 rejects=0
```

:material-checkbox-blank-outline: **Checkpoint:** a status line every minute; books changing on symbols nobody in the room is trading.

## 15.7 Exercise 4: Price and Value

The instructor can see what the students cannot:

```
pm-market-sim --status --id OPS01 --top 5
```

```
symbol        value        mid  mid/value
TSLA         212.40     219.95     +3.55%
...
```

The value traders lean against these gaps; the noise and momentum traders keep opening new ones. Run the command again a few minutes later: the symbols at the top change, and large gaps rarely last.

:material-checkbox-blank-outline: **Checkpoint:** mid/value gaps of a few percent at most, changing from one look to the next.

## 15.8 Exercise 5: An Earnings Surprise

Every student opens a news display: the **News** screen of the trading GUI, the News view of TapeDeck, `pm-ticker`, or simply

```
pm-news list
```

Then the instructor plays the first scenario:

```
pm-news --id OPS01 play "$EX/news/earnings-surprise.yaml"
```

Sixty seconds into the play (it counts from its start when the market is already in continuous trading) AAPL reports far above expectations. The news is confirmed from the start: AAPL's value jumps about 8% at once. The news traders hit the book within seconds; the value investors follow.

Students try to buy AAPL as soon as they see the headline:

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=MARKET|QTY=100
```

Compare your fill with the AAPL price a minute later.

:material-checkbox-blank-outline: **Checkpoint:** AAPL trades several percent higher within a minute of the headline.

## 15.9 Exercise 6: A Rumour That Turns Out True

First stop the market maker (Ctrl+C on `pm-mm-bot`; its quotes are withdrawn). It quotes around the true values, and a rumour changes no value: it would sell to every rumour buyer at the old price, and the rumour would leave no trace. Without it, prices are what the traders believe — the AI traders' resting orders keep the books two-sided.

```
pm-news --id OPS01 play "$EX/news/rumour-confirmed.yaml"
```

At 60 seconds a rumour says JPM is a takeover target, credibility 50%. Nothing has happened to JPM's value — any move now is traders betting on the rumour. Discuss before the next step: buy now, or wait? At 240 seconds the bid is confirmed and the value jumps about 12%.

:material-checkbox-blank-outline: **Checkpoint:** JPM moves a little on the rumour and much more on the confirmation.

## 15.10 Exercise 7: A Rumour That Is Retracted

```
pm-news --id OPS01 play "$EX/news/rumour-retracted.yaml"
```

At 60 seconds a rumour says energy producers face a new windfall tax: a *sector* headline, so every ENERGY symbol (`XOM`, `CVX`, …) is concerned. The values do not move; prices dip only because traders believe it. At 240 seconds it is retracted — in the GUIs the rumour is now struck through — and prices drift back to where

the values are.

:material-checkbox-blank-outline: **Checkpoint:** ENERGY symbols dip after the rumour and are back near their earlier prices a few minutes after the retraction.

## 15.11 Exercise 8: A Breakthrough and a Fire

Not all news is earnings. *Events* (a fire at a factory, a research breakthrough, a big new contract, a product recall) go one way only, and they are big. Their kind says what happened: `OPERATIONS`, `RESEARCH`, `CONTRACT`, `ESG` or `PRODUCT`. Keep the market maker stopped, and have students watch NVDA and AMD side by side.

```
pm-news --id OPS01 play "$EX/news/chip-race.yaml"
```

At 60 seconds NVDA announces a `RESEARCH` breakthrough: its next chips are 50% faster. The news is confirmed, and NVDA's value jumps about 10%. It is a *symbol* headline, so AMD's value does not move at all. Discuss: should AMD's price fall because its rival got ahead?

At 240 seconds a rumour says AMD's main chip fab is on fire, credibility 70%. Discuss before the next step: sell AMD now, or wait for confirmation? A credibility of 0.7 is high, and a fire never turns out to be good news. At 360 seconds the fire is confirmed and AMD's value drops about 7%.

:material-checkbox-blank-outline: **Checkpoint:** NVDA jumps on the first headline while AMD barely moves; AMD dips on the rumour and falls further on the confirmation.

Restart the market maker afterwards (the command from Exercise 1).

## 15.12 Exercise 9: Stop the Swarm

Ctrl+C on `pm-ai-swarm` stops every worker; each closes its traders' sessions, which cancels their orders. Stop `pm-market-sim` with Ctrl+C too: it saves its values and continues from them next time.

:material-checkbox-blank-outline: **Checkpoint:** no `AI0xx` orders left in the books; the market maker's quotes remain.

## 15.13 Classroom Tips

- Your own headlines need no scenario: `pm-news --id OPS01 inject --symbol TSLA --kind PRODUCT --sentim -0.6 --impact -0.05`. A rumour you inject (`--rumour --credibility 0.4`) waits for your `pm-news --id OPS01 confirm N7` or `retract N7` — the id is printed when you inject it.
- `pm-market-sim` also generates news on its own, a handful a day. Set the rates in `market_sim.yaml` to `0` for a class where only your headlines should appear.
- Same `swarm.yaml` and `market_sim.yaml` seeds, same traders and same values: change `seed` in either file for a different market.
- A larger `--anchor-sim` (up to 1) makes the market maker follow the true value faster; then the AI traders, and the students, have less of a gap to trade.
- Running the AI-Trader Swarm covers sizing, logs and daily summaries for longer runs.

## 15.14 Reflection

Why does a confirmed headline move the price further than a rumour with the same sentiment? Who in the swarm was on the other side of the news traders' orders right after the earnings headline? The market maker quotes around the true value: what would have happened to the price, and to the students' chances of buying AAPL cheaply, if it had not been anchored? An earnings headline can go either way, but a fire can only be bad: how did that change what you did on the AMD rumour?

## 15.15 Further Reading

- AI Traders
- Running the AI-Trader Swarm
- The Market Model (`pm-market-sim`)
- The Market-Maker Bot (`pm-mm-bot`)
- AI Traders and Market Simulation — a simulated market that runs for weeks, with the AI traders and news on all the time

**Next:** 15 — Statistics & Reporting

CHAPTER 16

# Running It for Weeks

**Learning objectives**

After reading this chapter you will be able to:

- Prepare the simulation to run unattended for a month or more
- Say what grows on disk, by how much, and where it stops growing
- Check its health in a minute a day, and know what the container repairs by itself
- Back it up, upgrade it and restart it without losing the market's history

## 16.1 Before you leave it

| Check | Why |
|---|---|
| **Pin the version**: `EM_VERSION=<release>` in `.env` | `latest` changes with every release; a restart would bring a different exchange |
| **Free disk**: at least 4 GB for 30 days, plus room for the nightly backup | See What grows and The nightly backup |
| **The machine stays awake** | A laptop that sleeps stops the market; nothing trades while it sleeps, and the day's schedule resumes when it wakes |
| **Docker starts at boot** (Docker Desktop: *Start when you sign in*; Linux: `systemctl enable docker`) | The containers restart after a reboot only if Docker itself does |
| **Time zone** is what you want (`TZ`, default `Europe/Stockholm`) | The trading hours follow it |
| **`EM_START` is not set** | It would send the calendar back at every restart |

## 16.2 What the container repairs by itself

| Event | What happens |
|---|---|
| A process dies | Restarted within a minute; the container log notes it (`docker logs edumatcher-sim`). The market maker can take up to four: the engine keeps its old session until it misses three heartbeats, a minute apart, and refuses the new one until then |
| The container dies or the machine reboots | Docker restarts it (`restart: unless-stopped`); the exchange redeploys nothing, keeps its data and starts the profile again |
| The engine restarts during trading | The AI traders and the market maker log on again; the engine restarts flat (positions from zero, GTC orders restored) and `CLOSED`. The scheduler does not notice: during the trading day the market stays closed until you restart it with `docker exec edumatcher-sim pkill -f pm-scheduler` (the watchdog starts it again within a minute, and it catches up to the current phase) |
| The market model restarts | It continues from the values it saved at most 10 seconds earlier |
| Daylight saving time changes | The scheduler re-reads the wall clock every few minutes; the next phase happens at the right local time |
| A holiday | No trading that day; nothing to do |

## 16.3 What grows

Measured with the `s300-sim` set-up at real speed on a 2-vCPU machine: the growth over continuous trading at the swarm's steady 80 actions a second (about 1,400 trades a minute), scaled to a 6.5-hour trading day. Your numbers will be close if you keep the example's budget; they scale with it. "7 days" are calendar days, so about five trading days are kept.

| File | Per trading day | Kept | Stops growing at |
|---|---|---|---|
| `stats.db` — trades and price snapshots | about 85 MB | 7 days | about 0.45 GB |
| `stats.db` — order events | about 250 MB an hour | the last 12 hours (`--keep-order-events-days 0.5`) | about 3 GB, within the first trading day |
| `stats.db` — your own order events | a few KB a day | 7 days (`--human-gateways 'TRADER*'`) | — |
| `stats.db` — daily statistics | about 30 KB | for good | — (about 1 MB a month) |

| File | Per trading day | Kept | Stops growing at |
|---|---|---|---|
| `clearing.db` — trades | about 155 MB | 7 days, pruned at every close | about 0.8 GB |
| `clearing.db` — positions and daily summaries | a few hundred KB | for good | — |
| `log.db` | about 5 MB | 30 days | about 150 MB |
| `audit.log*` | far more than it keeps | the last 60 MB | 60 MB |
| `ai_swarm/*.jsonl` | about 30 KB | for good | — |
| `emo/*.log` | a few KB | for good | — |

**Projected for 30 calendar days (21–22 trading days): about 2 GB in `./data-sim`.** Most of it is there after the first week; after that only the log (until day 30) and the small daily files grow.

**Why the audit trail is so short**

The audit trail records every message on the exchange's bus. In this market that is tens of megabytes a minute, so the 60 MB it keeps (six 10 MB files) cover only the last few minutes of trading. It is there to investigate what just happened, not to keep history; the statistics database and the swarm's daily summaries keep the history.

**Disk at a faster calendar.** The exchange's work — and so the growth — is per wall-clock second, not per trading day. At `EM_SPEED=24` a trading day writes a twenty-fourth of the table above, but 24 of them pass in a day — and the share of the wall-clock time spent trading is the same at any speed (6.5 hours of every 24, less weekends). So the disk fills at the same rate per real day at any speed.

## 16.4 A minute a day

From the repository's `deployment/docker` (or with the commands it wraps):

```
make sim-status
```

```
pm-opctl profile: ai-sim
...
data: 1.2G   disk free: 41G
```

Look for:

1. every process `running`;
2. the disk free comfortably above what the next weeks need;
3. once a week, `./edumatcher.sh logs edumatcher | grep Restarting` — the same process restarting every day is worth a look in its log;
4. one daily summary file per trading day in `data/ai_swarm/`.

## 16.5 The nightly backup

After each close `pm-eod` backs up the money record into `data/backup/<date>/` (compressed; see End of day). In the `ai-sim` profile this is deliberately light, because the simulation's databases are big:

- **Two days are kept**, not the default seven (`--backup-keep-days 2`).
- **`stats.db` is not backed up** (`--backup-skip stats.db`). It is the largest file here — about 3 GB at the close by the table above — and most of it is order events that are thrown away after 12 hours anyway. What cannot be recreated is `clearing.db` (the trades) and `eod.db` (the balances), and those are backed up.

The table above was measured before the backup existed, so **its totals do not include it**. What can be said without measuring:

- To take a copy that is safe while a database is open, SQLite first writes an uncompressed copy next to the backup. The biggest one is `clearing.db`, about **0.8 GB** at the close by the table above, so the disk needs that much free *at the close*, on top of the steady state. The copy is deleted as soon as it is compressed.
- Two compressed backups of `clearing.db` and `eod.db` stay in `data/backup/`. How small they compress has not been measured for this set-up. **After the first week run `du -sh data/backup`** and add it to your disk estimate.
- `data/settlement/` grows by a few hundred kilobytes a trading day (a statement for every trader) and is never pruned.

To change this, edit the scheduler's `--on-close` in `emo-config.yaml` (`./edumatcher.sh shell pm-opctl-cli init` writes the file; see Your own traders or process flags). For example, add `--backup-skip clearing.db` to skip that backup as well, or remove `--backup-skip stats.db` to back up everything (and keep another 3 GB free).

## 16.6 Backups

The whole state is the install's `data` directory. To copy it consistently, stop first:

```
./edumatcher.sh stop
tar czf sim-backup-$(date +%F).tgz data
./edumatcher.sh start
```

The market is closed at night and at weekends — a good time. To restore, stop, replace `data` with the backup's, start.

## 16.7 Upgrading during a run

```
./edumatcher.sh update          # the newest release
./edumatcher.sh update 0.51.0   # ...or an exact one
```

Do it while the market is closed. The data directory is kept; the deployed configuration stays as it was (a new release's version of `s300-sim` is deployed only on a fresh data directory). If you wrote the profiles out with `pm-opctl-cli init`, your `emo-config.yaml` keeps the old profiles too: compare it with the new release's built-in ones (`docker exec -e EDUMATCHER_DATA_DIR=/tmp/scratch edumatcher-sim pm-opctl-cli init` writes them to `/tmp/scratch/emo-config.yaml` in the container, away from yours).

## 16.8 The evidence

The numbers in this chapter come from runs of the `edumatcher-ai-sim` set-up recorded in the repository (`docs-design/reviews/ai-sim/`): a soak covering 34 trading days at accelerated speed (`EM_SPEED=600`, under

two hours), with every daily summary written, memory within 5 % of where it started, and the market maker killed once on purpose and back in under three minutes; and real-time runs for the CPU and disk figures.

CHAPTER 17

# Troubleshooting and FAQ

## 17.1 When something does not work

| You see | Likely cause | Do |
|---|---|---|
| `pull access denied` / `manifest unknown` on `up` | The release you asked for (`EM_VERSION`, or `latest`) has no `edumatcher-ai-sim` image | Pin a release that has one, or build it yourself |
| `port is already allocated` | Another EduMatcher stack, or another program, holds 5560–5600, 8080/8081 or 8090–8095 | Stop the other stack (`make down-all`, `./edumatcher.sh stop`); only one can run |
| Nothing trades | The market is closed: weekend, a Swedish holiday, or outside 09:00–16:05 Stockholm time | `docker exec edumatcher-sim date`; for trading now, accelerated time |
| The market closed in the middle of the day and stays closed | The engine restarted (it starts `CLOSED`) and the running scheduler did not notice | `docker exec edumatcher-sim pkill -f pm-scheduler`: the container restarts it within a minute and it catches up to the current phase |
| Trading starts at the wrong hour | The container's time zone is not the one you think | `TZ` in `.env` (default `Europe/Stockholm`) |
| `mm-bot` is `dead` right after a start | It started before the engine was ready and gave up | Nothing: the container restarts it within a minute |
| `mm-bot` restarted several times in a row after it died | The engine still holds the dead bot's session and refuses a second one for `MM01` | Nothing: the engine drops that session after three missed heartbeats, about three minutes, and the next restart logs on |

| You see | Likely cause | Do |
|---|---|---|
| A process keeps coming back as `dead` | It fails at start | Its console output: `docker exec edumatcher-sim tail -50 /data/emo/<name>.log`; and the Log Operator Console |
| `ai-swarm` dead, `ai-swarm.log` says a participant is missing | The deployed configuration does not register `AI001`–`AI100` (you deployed another one) | Deploy `s300-sim` or a configuration with those participants |
| `market-sim` dead, log says `no …/market_sim.yaml` | The deployed configuration came without a market model | Put a `market_sim.yaml` next to your `engine_config.yaml` and deploy again |
| Value and news traders send nothing | `pm-market-sim` is not running | `pm-opctl-cli list`; see the row above |
| Unknown profile `ai-sim` | `emo-config.yaml` in the data directory was written by an older release and replaces the built-in profiles | Delete `./data-sim/emo-config.yaml` (or add an `ai-sim` profile to it) and restart |
| Swarm logs `cancel … never answered` steadily, `actions/s` far below budget | The machine is overloaded: the exchange cannot keep up | Lower `budget` (Tuning the Traders); check `docker stats` |
| The GUIs show no history, Order Book Viewer empty | The read-only API key does not match the deployed configuration | With your own configuration, set `PM_READONLY_API_KEY` to its read-only key (`pm-config-show -a`) |
| `./data-sim` keeps growing | Expected: trades are kept 7 days and the log 30 days, so it grows quickly for the first week, slowly until day 30, then levels off | Running It for Weeks |
| After a restart the accelerated calendar went back in time | `EM_START` applies at every start | Remove `EM_START` after the first start (Accelerated Time) |
| Prices drift far from the values | The market maker is not anchored, or there are too few value traders per symbol | `--anchor-sim`, the composition, `per_agent` (Tuning the Traders) |

## 17.2 FAQ

**Can students see the true values?** No. The values travel on the market model's own sockets, bound inside the container, and the API serves them to ADMIN keys only. Headlines are public; their impact on the values is not.

**Can students lose money to the bots?** Yes — that is the point. The AI traders are participants like the students, and the informed ones trade on better information. The P&L is play money.

**Can I trade against the AI traders with my own program?** Yes. Connect as one of `TRADER01`–

`TRADER10` through the API (port 8080) or the order gateway (5565), exactly as on any EduMatcher exchange; the Protocols and Clients book describes both.

**Do the AI traders learn?** No. Each follows its preset's fixed rules with its own random choices. What changes from day to day is the market — the values, the news, the other traders — not the rules.

**Is a run repeatable?** The model is seeded (`seed` in `market_sim.yaml`) and so are the traders (`seed` in `swarm.yaml`), but the order in which messages from a hundred traders reach the engine depends on timing, so two runs diverge within seconds. The *statistics* of a run — how far prices stray, how much each kind of trader makes — are repeatable; the exact prices are not.

**How do I start over?** Stop the stack, delete `./data-sim`, start it.

**How do I give the market a fresh set of values without losing the history?** Stop the stack, delete `./data-sim/market_sim_state.json`, start it: the model starts again from the engine's previous closes.

**Why is there no trading at 09:00 on a Monday in January?** Swedish public holidays — Epiphany, 6 January, for one — are closed days on the `s300-sim` calendar. Use your own configuration's `locale.country` and `holidays` settings to change the calendar.

**Can I run it in the cloud?** Yes; any machine with Docker and two cores will do. Mind the classroom security notes before you publish the ports.

**Can I run two simulations side by side?** Not with the bundled compose file: both would want the same ports. Copy it, change every host port and the project and container names, and use a different data directory.

CHAPTER 18

# Simulation Glossary

The terms this book uses for the simulation. Exchange terms in general — order book, spread, auction, tick — are in the Reference Manual's glossary and in How an Exchange Works.

**Accelerated time** — Running the trading calendar faster than the wall clock (`EM_SPEED`). Only the calendar is faster; the traders act at their usual pace. See Accelerated Time.

**`ai-sim`** — The process profile that runs the simulation: the full exchange, the market maker, the market model and the AI swarm.

**AI trader** — A program that trades as a participant (`AI001…`), following a preset. Many of them run inside one `pm-ai-swarm`.

**Anchor** (`--anchor-sim`) — How strongly the market maker moves its quotes toward the true value each second: the share of the remaining gap it closes.

**Bias** — A value trader's own, persistent error about a share's value. Different biases are why value traders disagree and trade with each other.

**Budget** — The order actions per second (new orders and cancels) the whole swarm aims for.

**Composition** — The mix of presets in the swarm: how many traders of each kind.

**Credibility** — How believable a rumour looks, 0 to 1. Informative, not reliable.

**Efficiency** — How closely prices follow the true values; measured as the gap.

**`EM_SPEED`, `EM_START`** — The container's settings for accelerated time: how many times faster, and from when.

**Gap** — `ln(price / value)`: how far a price is from its true value, about the same as the percentage difference. Summarised over all shares as its RMS (root mean square).

**Headline** — A piece of news: scope, kind, sentiment, and the secret impact.

**Impact** — The log move a confirmed headline causes in the true value; secret. 0.05 is about +5 %.

**Informed trader** — A trader that knows something about the true values or the news: the `value-investor`, `institutional` and `news-trader` presets.

**Jump** — A sudden move in a value without any headline: an unannounced event.

**Market model** — `pm-market-sim`: keeps the true values, moves them, publishes news.

**Noise trader** — A trader that trades in random directions, without information: `noise-retail`, `market-taker`, `iceberg-seller`.

**Preset** — A kind of AI trader: a strategy, an execution style, a tempo and risk limits, in one YAML file.

**Price discovery** — The process by which trading moves a price toward the value.

**Retention** — How long a record is kept before it is deleted to bound the disk: 7 days for individual trades in the statistics and in clearing, about an hour for order events, 30 days for logs, the last 60 MB of the audit trail.

**Rumour** — A headline that may not be true. It moves no value until it is confirmed; if it is retracted, it never does.

**Scope** — What a headline is about: one symbol, a sector, or the whole market.

**Sector** — An industry group of shares that move together: eleven in `s300-sim`.

**Sentiment** — How good a headline sounds, −1 to +1; public, and a blurred version of the impact.

**`s300-sim`** — The bundled example the simulation runs: 300 shares, `AI001`–`AI100`, `MM01`, `TRADER01`–`TRADER10`, `OPS01`.

**Swarm** — `pm-ai-swarm`: the process that runs many AI traders, in one or more worker processes.

**True value** — A share's hidden value in the market model; what price discovery aims at.

**Volatility** — How much a value (or a price) moves: the standard deviation of its log changes, quoted per year (30 % a year is about 1.9 % a day).

**Watchdog** — The container's once-a-minute check that restarts any process of the profile that has died (`EM_RESTART_DEAD=1`).