

MEELMO

User Manual

The Desktop engineering client · The browser operator HMI ·
Authoring screens in the HMI Studio · Automation rules · When
something is wrong

Contents

Part I — The Desktop engineering client

1. First run
2. The main window
3. I/O configuration
4. Acquisition diagnostics
5. Trend graphs
6. Measurement, annotations and statistics
7. Virtual and computed signals
8. Recording

9. Playback
10. Alarms
11. The historian and the history browser
12. Datasets, reports and run comparison
13. Configuration revisions
14. Users, roles and audit
15. Service accounts and integration
16. Modules and licensing

Part II — The browser operator HMI

1. Signing in and the site switcher
2. Live values, gauges, states and mimics
3. Live trends: scope, logger, multiplot and stacked
4. Alarms

5. History replay
6. What an operator may and may not do
7. Models and analysis, without the Desktop
8. Administration in the browser

Part III — Authoring screens in the HMI Studio

1. Projects, pages and the canvas
2. Binding signals and choosing an update rate
3. Composites, templates and preview

4. Validation, publishing and assignments
5. Commands and acknowledge
6. Asking the Assistant

Part IV — Automation rules

1. Starting a rule
2. Settings are chosen, not typed

3. The graph view
4. Testing and activating

Part V — When something is wrong

How to use the product, from first launch to a commissioned plant. This manual ships inside the application and is what **Help ► User Manual** opens.

It is in five parts. **Part I** is the Desktop engineering client, where a plant is built and analysed. **Part II** is the browser operator HMI, where a plant is watched and run. **Part III** is authoring the screens Part II shows. **Part IV** is automation rules — what the plant does on its own when something happens. **Part V** is troubleshooting.

Companion documents: *INSTALLATION_GUIDE.md* to get it installed, and the *Software Architecture Document* for what is underneath.

Who does what. The Desktop is the **engineering and commissioning** client — it has no "run the plant" operator runtime and no operator faceplates. The browser is the **exclusive operator surface**. Both are clients of the same node. If you are looking for the screen an operator stares at all shift, it is in Part II.

Part I — The Desktop engineering client

1. First run



The Desktop workspace

A node with no administrator shows a **first-run page** rather than a login: name the node, name its site, and create the first administrator. Every account after that is made in **Users & permissions** by somebody who has signed in.

Then open something that moves:

1. **File ► Open Project...** → `examples/simulator-demo.meelmoproj`.
2. **Acquisition ► Start.** The signal tree shows values changing.
3. Double-click a tag, or drag it onto the chart, to plot it.

In a hurry, open `examples/packaging-line-demo.meelmoproj` **instead** — a simulated bottling and packaging line of about 27 signals across the line, filler, quality, downstream machines, utilities and shift counters. It opens with three tiled trend graphs already arranged, virtual signals, live alarms and a recording trigger. Press **Start** and it comes alive.

`examples/galvanizing-line-demo.meelmoproj` is the same thing for continuous process, if that is the plant you are judging it against.

2. The main window

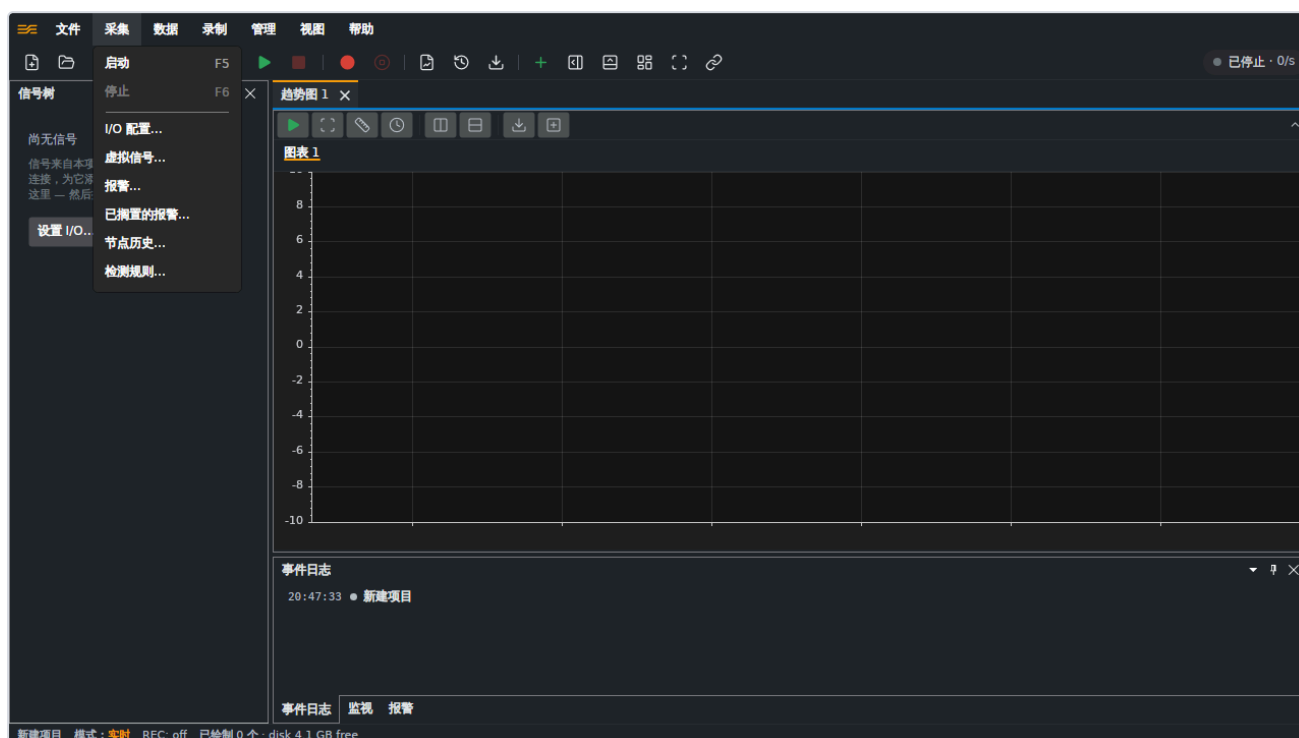
Four regions, all of them movable.

- **The signal tree** — Connection → Group → Tag, with live values and a quality indicator on each.

A tag's own name is a path in every symbolic controller — `"MyDB".Axis.Speed` , `MAIN.fbMotor.rSpeed` , `Objects/Machine/Axis1` , `Tank[3].Level` — so the tree opens it into branches on `.` , `/` and the array index rather than listing one long row per element. A data block, a structure and an array each become a branch you can close, so an array of two hundred elements is one line until you open it. A dot inside a quoted block name (`"My.DB".Speed`) is part of the name and is not split on. Branches with only a few children start open, so a project of ordinary names looks as it always did.

- **The chart area** — one or more trend graphs sharing a time axis.
- **Panels** — Signals, Event Log and Watch. Drag, tile, tab, float, collapse or close them; re-open from the **View** menu.
- **The status bar** — acquisition state, the recording file and its sample count.

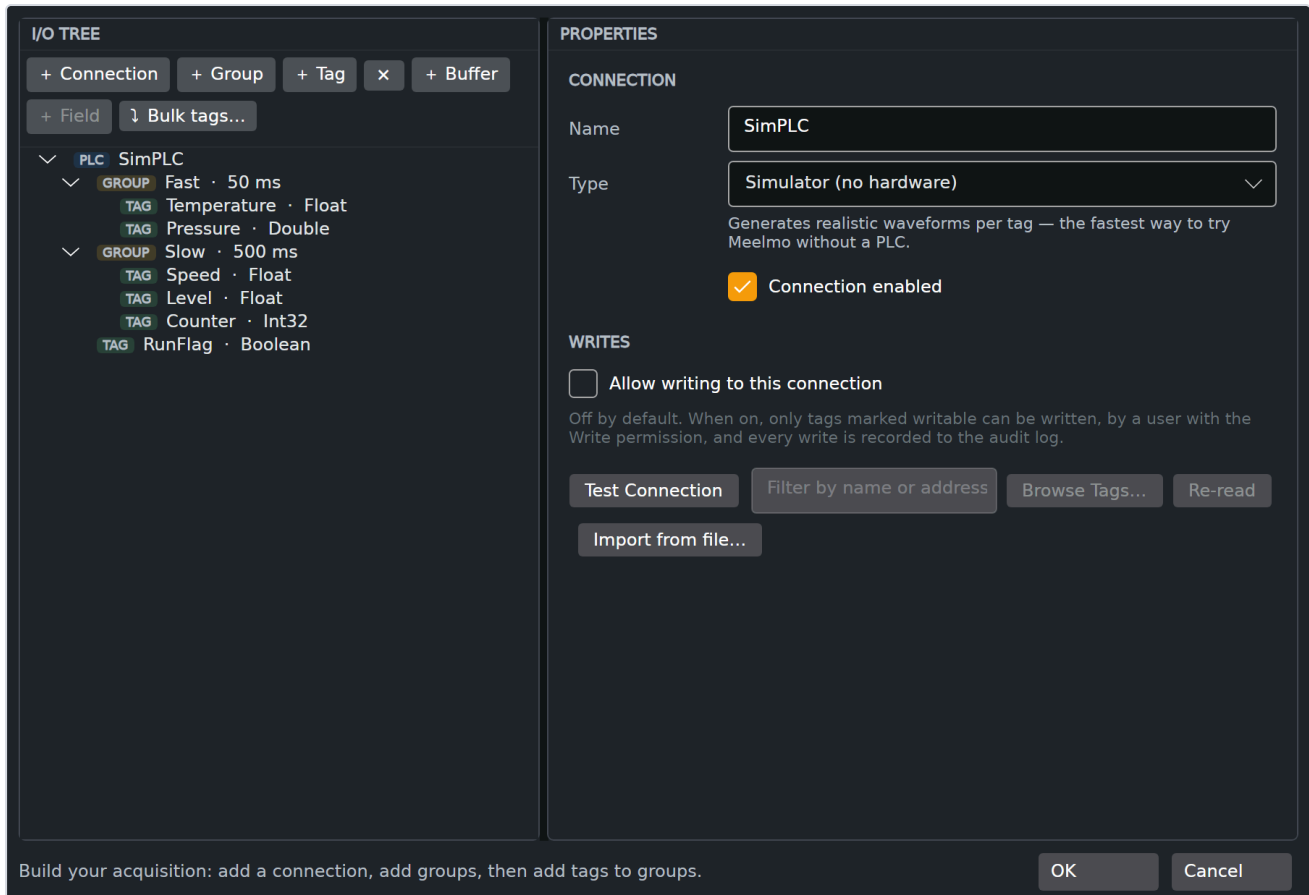
The layout is saved **with the project**, so which signals sit on which chart strip comes back when you open it.



The Acquisition menu

Four menus carry the work. **Acquisition** starts and stops the plant and holds I/O configuration, virtual signals, alarms and diagnostics. **Recording** holds capture, continuous mode, triggers and the recordings browser. **Data** holds the historian, the history browser, datasets and reports. **Administration** holds users, modules, licensing, certificates, deployment and the audit trail.

3. I/O configuration



I/O configuration

Acquisition ► I/O Configuration... is where a plant is described. Three levels:

Connections — one per controller or endpoint. Pick the driver, fill its settings (each driver ships sensible defaults), and press **Test connection** before going further. Which drivers the list offers depends on your edition; the Product Guide describes every one.

Acquisition groups — tags in a group are read in **one driver call and share one timestamp**. That is data correlation, and it is why grouping is a modelling decision rather than a UI convenience: two values you intend to compare should be in the same group. Each group acquires at its own rate; ungrouped tags are bucketed by rate.

Tags — a grid, with validation. On Siemens S7CommPlus, Allen-Bradley, Beckhoff and OPC UA you do not have to type addresses: **Browse** reads the controller's own symbol table, filters as you type, maps the types for you, and walks into structures and function-block instances so you can pick individual members. A browse of a large program reports progress and can be cancelled.

Arrays expand into one entry per element, named by the index the controller's own engineering tool uses — an `Array[1..10]` lists as `[1] ... [10]`. That is because a tag holds one value: an array reads back in a single request, but there is nothing to trend in a symbol that names the whole of it, so browse offers you the elements instead.

Choosing an acquisition mode

Each connection — and optionally each group — runs in one of three modes, and the UI offers only the modes the driver actually supports.

Mode	Who decides when a sample happens	Use it for
Polling	Meelmo, on a timer	Most fieldbus protocols
Subscription	The server (OPC UA, MQTT)	Change-driven data, preserved source time
Cyclic streaming	The controller, every cycle	Genuine sub-10 ms, down to ~1 ms

Timing you can defend comes from the bottom two rows. A poll loop on a general-purpose operating system is best effort; asking for 5 ms does not mean getting 5 ms, and the product will tell you which you got rather than repeat back what you asked for. Projects saved before modes existed load as **Polling**.

Applying an I/O configuration restarts only the connections it changed. A

connection whose driver settings, groups, tags and mode are the same in both revisions keeps acquiring without a gap; a changed one is stopped and started again, an added one starts and a removed one stops. This is not a hot edit: a changed connection is restarted, never altered while it runs. A change to the temporal-coherence settings every connection shares restarts all of them, and `applyMode: "full"` in the acquisition section restarts everything on every activation, as before. The activation's answer and the audit trail say which connections were restarted, added, removed and left running.

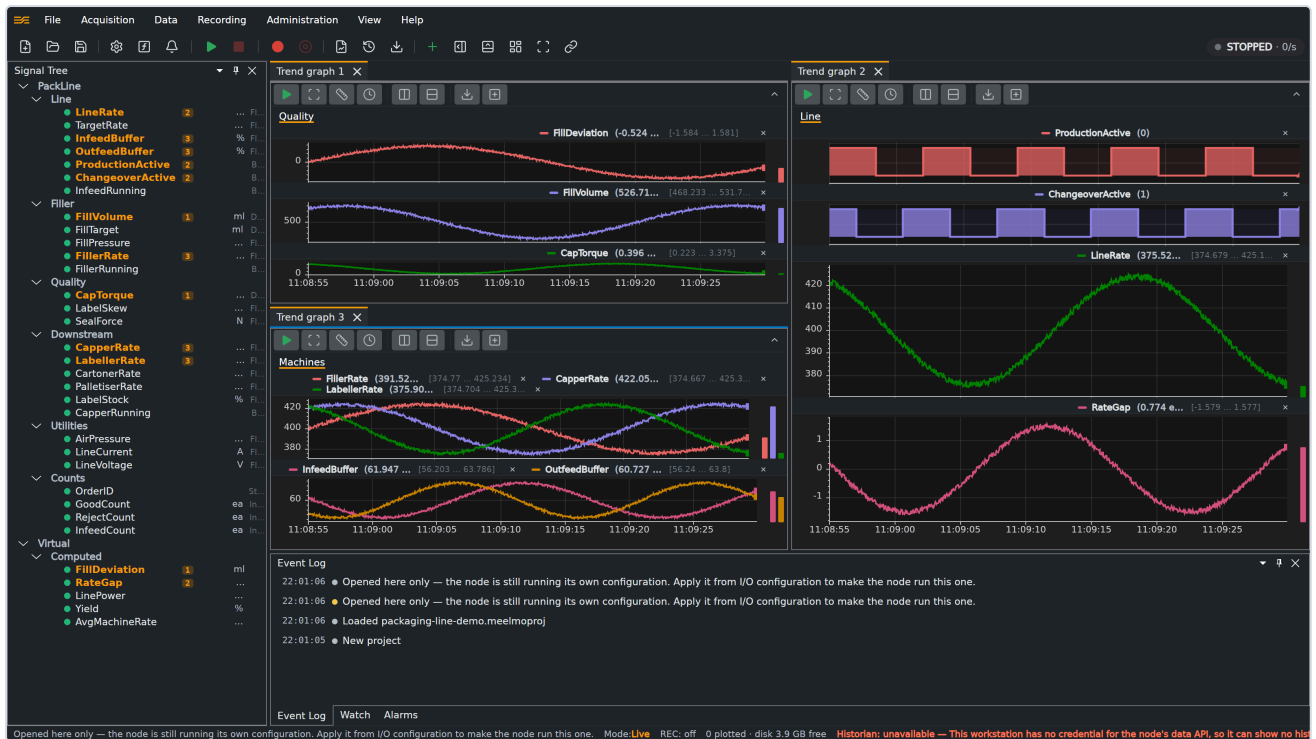
In the Desktop the choice is **When applied** at the foot of **I/O configuration** — *Restart only what changed* or *Restart every connection*, each with a sentence saying what it does — and after OK the status bar shows the node's account, e.g. *The node applied revision 42: 1 connection restarted, 5 untouched.*

4. Acquisition diagnostics

Administration ► Diagnostics... is the screen that proves the commissioning. Per connection and per group it shows the mode; the **requested versus actual and revised** period; min and max period; jitter at p50, p95 and p99; read round-trip; source-to-receive latency; sequence gaps, duplicates and out-of-order packets; driver queue depth; poll overruns; and a recorder queue/recorded/dropped summary.

Open it before you believe a trend. A line that is not sampling the way it was configured is found here, at commissioning, or it is found months later during an investigation — and by then the recording is already wrong.

5. Trend graphs



A trend graph

Double-click or drag a tag to plot it. + **Trend graph** above the chart adds another strip; strips share the time axis and keep their own Y axes.

- **Axis layouts** — stack strips vertically or horizontally ([vertical](#), [horizontal](#)).
- **Digital and text signals** — booleans draw as thin digital state tracks and text as categorical lanes, rather than being forced onto a numeric axis.
- **Auto-scroll** keeps the right edge pinned to now. A **zoom on a live chart keeps following the plant**; a pan leaves live for the time it asked about.
- **Zoom history** — back and forward through a sequence of zooms, not only *zoom to fit*.
- **Colours are position-based**, so the same strip position gets the same colour across graphs and a screenshot stays readable.

Rendering cost follows **screen resolution, not sample count**: a million-point trace repaints in about 85 ms, roughly four times the cost of ten thousand points. Charts are capped at 500,000 points per trace by default.

6. Measurement, annotations and statistics

The cursor snaps to samples and reads each channel's value at that instant. The **A-B grid** measures between two points. The **statistics panel** gives per-channel min, max, mean, RMS, σ and Δ over the visible window, plus FFT.

Annotations survive the analysis. Text pinned to a moment on a chart is drawn as a labelled line and **saved with the project**, so a finding is something a colleague can open rather than a screenshot you have to email. Every other mark a chart carries is transient — that difference is deliberate.

7. Virtual and computed signals

SIGNALS

- **Temperature_F**
[SimPLC/Fast/Temperature] * 1.8 + 32
- **AvgLevel**
avg([SimPLC/Slow/Level], [SimPLC/SI...]

+ Add Duplicate x

DEFINITION

Name: Temperature_F Unit: °F ☒ Enabled

DEFINITION Kind: Expression

[SimPLC/Fast/Temperature] * 1.8 + 32

✓ Valid — references: SimPLC/Fast/Temperature

INSERT

Tags Functions Operators

Temperature · SimPLC
[SimPLC/Fast/Temperature]

Pressure · SimPLC
[SimPLC/Fast/Pressure]

Speed · SimPLC
[SimPLC/Slow/Speed]

Level · SimPLC
[SimPLC/Slow/Level]

Double-click a tag to insert its reference.

Build computed signals from acquired tags. Insert tags and functions from the palette below; the expression is validated as you type.

OK Cancel

Virtual signals

Acquisition ► Virtual Signals defines new channels from expressions over existing ones — `Temperature * 1.8 + 32`, `avg(a, b)` — evaluated live and treated like any other signal: plottable, recordable, alarmable.

Expressions can also read a signal's own past:

Function	What it does
<code>delay(x, t)</code>	A transport delay — so two machines a conveyor apart can be compared on the same case
<code>movavg(x, t)</code>	Moving average over a window
<code>tmax(x, t) / tmin(x, t)</code>	Rolling extremes
<code>deriv(x) / integ(x)</code>	Rate of change, accumulation
<code>edge(x)</code>	Transition detection

Each call site keeps its own bounded history, so two `delay` calls with different windows do not interfere.

8. Recording

Trigger-based recording

Automatically start recording when a condition becomes true and stop it after a hold time - no need to press Record.

☒ Enable trigger recording

Condition (record while true):

Signal:

SimPLC/Fast/Pressure

Condition:

is greater than 8.5

Compiles to: [SimPLC/Fast/Pressure] > 8.5

☐ Advanced (expression)

Condition is valid.

Keep recording after clear (seconds): 30

Pre-trigger buffer (seconds): 10

Prepend this many seconds captured just before the trigger fired, so the recording includes the run-up to the event. 0 = start exactly at the trigger.

OK Cancel

A recording trigger

Recording ► Start Recording writes an `.opslog` file; the status bar shows the file and the sample count. Record while viewing live — the two are independent.

What is captured: every value, including strings, with its quality code and **both** timestamps — when it was true at the controller and when this process saw it — plus a sequence number where the driver supplies one. About 16 bytes per sample. The file is crash-safe, self-describing (the signal tree is rebuilt from it), and **readable while it is still being written**.

Two layouts. *Single file* is one growing file per Start/Stop. *Continuous (Recording ► Continuous)* rotates by size or time inside a retention window and gzips what has rolled, leaving the active file plain so it stays readable live. Continuous capture and the headless recorder service are Professional features.

Lossless or best-effort. Lossless applies real back-pressure and drops nothing, reporting if it still cannot keep up. Best-effort counts every dropped sample rather than quietly losing it. The counters are Enqueued, Persisted, Evicted and Rejected, with `Dropped = Evicted + Rejected`.

Triggered recording starts on a signal condition and prepends a **pre-trigger buffer**, so the run-up to the event is in the file rather than the event being the first sample. Identical in the application and in the recorder service.

9. Playback

File ► Open Recording..., or **Recording ► Browse Recordings...** The tree is rebuilt from the recording's own signals, a fast overview renders immediately, and zooming re-queries the file at higher resolution. Cursors, statistics and FFT work exactly as they do live.

You can open a recording that is still being written, which is what you want during a long capture.

10. Alarms

The screenshot shows a configuration window titled "Alarm monitoring" with two tabs: "Rules" (selected) and "Notifications". Below the title is a descriptive paragraph: "Rules are evaluated live at acquisition rate. An alarm raises when its condition holds for the raise delay, and clears once the condition has been false for the clear delay. Raised and cleared events appear in the Alarms panel and can be sent by email or webhook."

On the left is a list of existing rules: "Boiler overpressure [Critical]" (highlighted), "Coolant temperature high [Alarm]", and "Tank level low [Warning]".

The main configuration area for the selected rule includes:

- An "Enabled" checkbox, which is checked.
- A "Name:" field containing "Boiler overpressure".
- A "Condition (non-zero = in alarm):" section containing:
 - A "Signal:" field with "SimPLC/Fast/Pressure".
 - A "Condition:" dropdown set to "is greater than" and a value field set to "8.5".
 - A compile preview: "Compiles to: [SimPLC/Fast/Pressure] > 8.5".
 - An "Advanced (expression)" checkbox, which is unchecked.
 - A status message: "Condition is valid."
- A "Severity:" dropdown set to "Critical".
- "Raise delay (s):" and "Clear delay (s):" fields set to "2" and "5" respectively.
- A "Message (optional):" text area.

At the bottom left are "Add" and "Remove" buttons. At the bottom right are "OK" and "Cancel" buttons.

Alarms

Acquisition ► Alarms authors condition rules with severities and raise/clear delays. They are evaluated **at acquisition rate**, identically in the Desktop and in the headless recorder — the rule does not behave differently because nobody is watching.

The alarm panel raises, shows and acknowledges. Notifications go by email and webhook, and to Slack, Teams, PagerDuty and Opsgenie, with raise and clear correlated by a dedup key so a flapping condition does not become a pager storm.

The webhook can post in the receiving system's own shape: a **body template** with `{{alarm}}`, `{{message}}`, `{{raisedUtc}}` and the other fields (each escaped for the **content type**), extra **headers** one per line, and an **Authorization** the node seals. **Preview** renders the body against

a sample alarm with the notifier's own code, showing any credential as (set) ; OK refuses a content type, header line or cleartext Authorization the node would reject.

Rule When it speaks What it says

An advisory reads what the plant recorded over a window and says what would be worth doing about it. It never acts: what it produces is a sentence on an operator's screen.

Kind

Id

Name

Asset path

Priority

Inputs

Each input is a signal, a name the condition uses for it, and what to do with its window. Signals are chosen from the project so a rule cannot watch a topic nobody publishes.

Packaging/Filler/FillVolume	Mean	Add input	Remove
--	-------------------------------------	--	---------------------------------------

measured = Mean(Packaging/Filler/FillVolume)
target = Mean(Packaging/Filler/FillTarget)
producing = Mean(Packaging/Filler/Running)

Condition

The same expression language as a virtual signal or an alarm condition — applied - target > 3 && speed > 60. It is true when it evaluates to anything but zero.

The condition compiles and every name it uses is supplied.

Authoring an advisory rule

A rule names its condition, its severity, and the delays before it raises and before it clears. **Use the delays.** A raise delay is what stops a signal that crosses a threshold for one scan from waking somebody; a clear delay is what stops a value sitting on the threshold from flapping. A rule with neither is the usual cause of an alarm list nobody reads.

Rule When it speaks What it says

An advisory reads what the plant recorded over a window and says what would be worth doing about it. It never acts: what it produces is a sentence on an operator's screen.

Kind: Constraint

Id:

Name:

Asset path:

Priority: Action

Machines

In line order, upstream first. The constraint is the machine that spends least of the window blocked or starved — everything else is waiting on it. Leave a state empty where the plant does not report it.

Machine name	Running	Blocked	Starved	Faulted
	—	—	—	—

Add machine Remove

Filler · 2 states
Capper · 2 states

Minimum margin:

How much clearer than the next machine the constraint has to be, as a fraction of the window. Below it the line is balanced, the leader changes hourly, and naming one would be noise with a machine's name on it. 0.05 is a reasonable start.

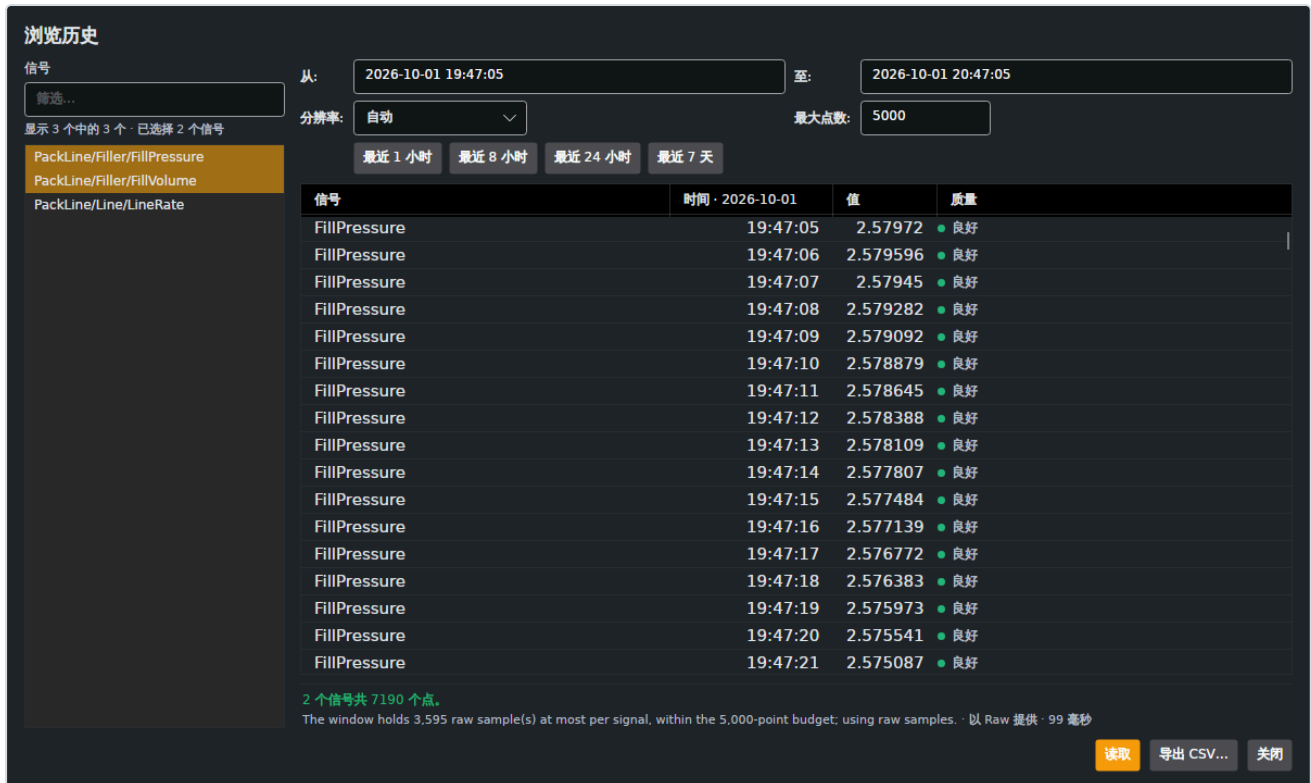
OK Cancel

A rule constrained to a state

A rule can also be **constrained** — evaluated only while the plant is in a particular state, or while an execution is open. A high-temperature rule that fires during a planned soak is a false alarm; the same rule constrained to *not soaking* is a real one.

A rule can fire on absence. Choose the comparison **has had no new sample for** or **has not changed for** and give a number of seconds: the first catches a source that has stopped sending, the second a value frozen while the reads go on. They compile to `nosample([signal]) >= seconds` and `nochange([signal]) >= seconds`, which can be combined with other conditions in **Advanced**. The node asks these rules once a second as well as when samples arrive, so a source that sends nothing at all still raises the alarm; and it measures silence from when its rules started, so a restart does not raise them all at once. A bad reading every poll is not a sample.

11. The historian and the history browser



The history browser

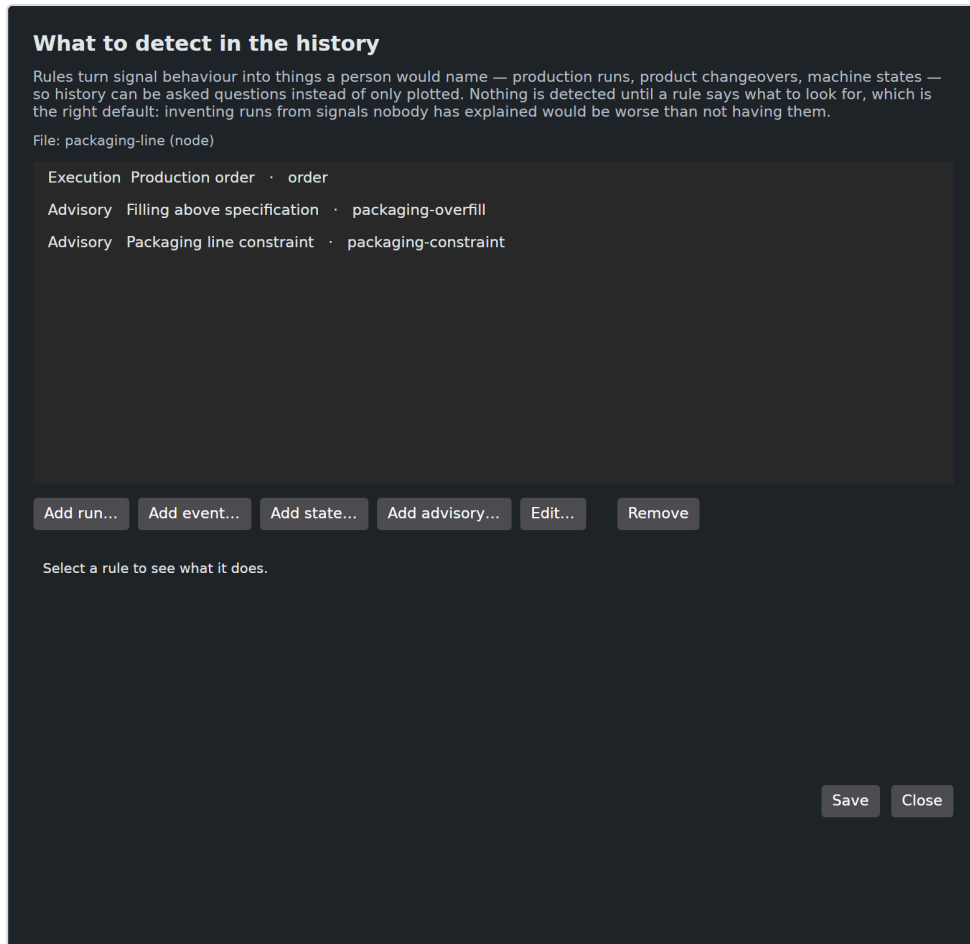
Data ► Historian configures long-term storage. The embedded store needs no configuration; SQLite is the same shape in one file; SQL Server, Oracle and TimescaleDB each need their module and a connection string.

The historian is **separate from .opsLog recording** and both can run at once. Signals are identified by a stable hierarchical id (Site/Area/Asset/Group/Signal) that never depends on a database key, so a rename becomes an alias and history stays reachable.

The **history browser** queries it: pick signals, pick a window, and the resolution selector serves the finest tier the point budget buys. Booleans and text return value-changes, never an analog envelope.

Publication is configured in the same window, and it has three independent wires: the MQTT or Sparkplug B **broker** (with an optional client certificate for a broker that authenticates by certificate, the **Publish alarms** and **Publish detected events** switches, and a Sparkplug **primary host id**), the node's read-only **OPC UA server** tab (port, host name, trusted client certificates, root path, and four commissioning-only switches — the one that accepts any client certificate is shown with a warning), and the **Kafka** tab (bootstrap servers, security protocol, SASL, topics, compression and certificates; Azure Event Hubs through its Kafka endpoint). Beside them, the **MQTT broker** tab runs a governed broker on the node itself: listen address, ports, anonymous clients, the namespace's levels, and access rules and payload contracts one per line. Passwords open as empty boxes that say whether one is stored, are sealed when saved, and are removed only with **Clear**. Save refuses what the node would refuse at activation, in the node's own words.

Detection rules — turning signal behaviour into things you can name



Detection rules

The historian stores signals. **Detection rules** turn what those signals do into entities a person names: an execution (a batch, an order, a wash cycle), a state (running, changeover, down), an event, a context.

The rules are **project data, authored per site** — the domain vocabulary lives in your configuration and never in the product. That is why the primitives are called Asset, Signal, Event, Execution, State, Context and Relationship rather than anything from your industry: the engine has no opinion about what a *batch* is, and you do.

Once a rule is running, those entities are queryable through the API and MCP, they drive the HMI's timeline and production components, and they are what run comparison aligns on. Detection queues its writes and **never blocks the acquisition thread**.

12. Datasets, reports and run comparison

数据集工作室

数据集定义描述导出的形态：哪些信号成为列、如何采样行、缺值时如何处理。它不包含时间范围——运行时由您指定。保存在服务器上，可重复使用。

已保存的定义

filler-1s (Wide, 2 column(s))

line-rate-1m (Wide, 2 column(s))

加载

新建

刷新

名称

filler-1s

描述

Fill volume and filler pressure, one second

形态

Wide

采样

FixedInterval

间隔 (500ms, 1s, 5m, 1h, 1d) —— 固定间隔采样时必须填

1s

值可以延续跨越的最大间隔 (留空表示不限)

缺失值

Null

最大行数

10000000

列

1. PackLine/Filler/FillVolume ← PackLine/Filler/FillVolume

2. PackLine/Filler/FillPressure ← PackLine/Filler/FillPressure

上移

下移

移除

信号

Dataset Studio

Datasets are reusable, server-side shapes exported to CSV, Excel, Arrow or Parquet. Define once, run repeatedly; report elements bind to them **by name**, never to a historian or a credential.

Reports are server-side definitions rendered to HTML, Excel and PDF, with schedules and delivery by directory, email or webhook. Every report states its **coverage** — what it could not see — and there is **no switch to hide it**. A report is mailed through the node's mail relay: the SMTP server set in the Desktop under **Acquisition ► Alarms... ► Notifications**, which alarm e-mail uses too.



The Report Studio

A report is laid out in **millimetres**, not screen pixels — A4, A3, Letter, Legal or custom, either orientation, with per-page margins and headers and footers stamped on every printed page. Elements bind to **saved datasets by name**, never to a historian or a credential, so a report is portable between installations that have the same dataset defined.

The browser preview, a scheduled delivery and a downloaded PDF go through the **same** composer, layout engine and renderer. What you see in the preview is the document that arrives in somebody's inbox.

Run comparison / golden-run overlay aligns reference executions on **phase** — normalized or shared-elapsed, never wall-clock — with mean $\pm \sigma$ deviation bands, a signed deviation summary and coverage. It is available in the Desktop, over the API, through the MCP `run_compare` tool, on the browser page `/analysis/runs`, and as the `run-overlay` HMI component.

Export from **File ► Export**: the plotted signals over the current range as CSV or an `.xlsx` workbook with a statistics sheet, or a self-contained HTML report of the current graphs and statistics.

13. Configuration revisions

Revisions

32026-10-01 22:01admin — running
Slowed the fast group to 250 ms

22026-10-01 22:01admin
Commissioned from the project file

12026-10-01 22:01admin
First run

1 setting differs between the running revision 3 and revision 2.

acquisition/endpoints/0/groups/0/pollRateMs
250 → 100

Roll back to this

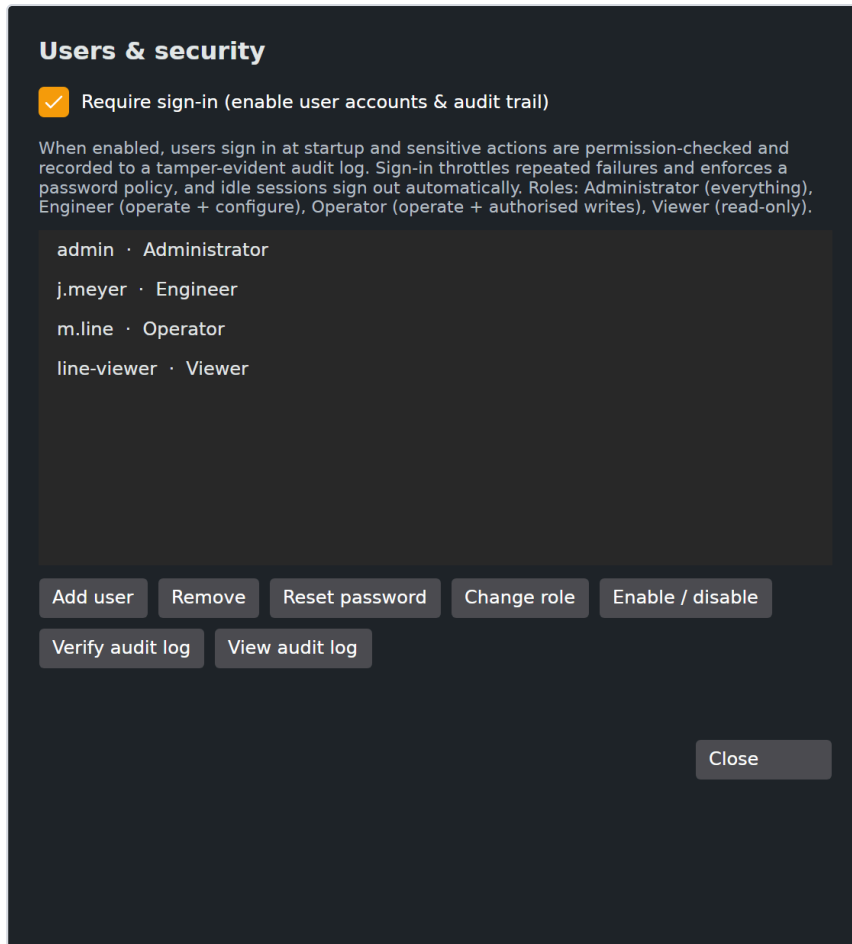
Close

The node's configuration history

A node keeps its configuration as **numbered revisions**, not as a file somebody last saved. Every activated change is a revision with an author, a timestamp and a reason, and you can read what changed between any two of them.

This matters more than it sounds. "The line started scrapping at 06:00 and nobody touched anything" is a claim; the revision list is the answer, and it either shows a change at 05:40 or it does not. A configuration you can diff is also one you can **roll back** — and a rollback is itself a revision, with your name on it, rather than a silent return to a previous state.

14. Users, roles and audit



Users

Authentication is **opt-in and off by default** — a single-user installation needs none. When enabled, sign-in is per user with a role, across the Desktop, the web host and the data-access server.

The Desktop has exactly **three** roles, and all three are engineering authority:

Role	May
Viewer	Open projects, play back recordings, export
Engineer	Viewer, plus commission and configure — acquisition, recording, alarms, I/O, virtual signals, the historian, reporting views, certificates
Administrator	Everything, including users, modules, licensing, service accounts and audit

There is no Desktop "Operator" role. Operator is a *browser* identity with HMI permissions only. Desktop users, browser users and service accounts are three distinct populations, all held by the node.

The **audit trail** is append-only and hash-chained. It is verified at start-up and on demand, and **if an entry cannot be written the action is blocked** rather than performed unrecorded.

15. Service accounts and integration

服务账户

每个集成都有自己的账户，因此可以撤回某个凭据而不影响其他集成。权限、范围和限制由服务器执行；此窗口向服务器查询。

- Commissioning — Reader, Export, ManageDatasets, Administer [svc-44c4c295c9a6]
- Packaging line dashboard — Reader, Export [svc-f3a84b8e2d3c]
- Shift report — Reader [svc-8cffb5cbe7ff]

刷新

新凭据

停用

启用

删除

新建账户

名称

1 号线报表

负责人

向谁询问

角色 (预设下方的权限)

Reporting — read only

Reads the catalog, history, real-time values and events. The account a dashboard, a report or a read-only integration needs — and nothing it does not.

权限

☒ ReadCatalog
☒ ReadHistory
☒ ReadRealtime
☒ ReadEvents
☐ Export
☐ ManageDatasets

☐ WriteAnnotations
☐ Administer
☐ Mcp

可读取的信号前缀 (留空表示所有信号)

PlantA/Line1

Service accounts

An integration authenticates as a **service account**, not as a person. **Administration ► Service accounts** creates, rotates and revokes them at the node. The token is `<accountId>.<secret>`, is **shown once**, and is stored as a salted SHA-256 digest — never the token itself.

Scope each account to the permissions it needs — `ReadCatalog`, `ReadHistory`, `ReadRealtime`, `ReadEvents`, `Export`, `ManageDatasets`, `WriteAnnotations`, `Commission`, `Administer`, or the `Reader` bundle — and to allowed and denied signal prefixes. Prefix scoping **narrows results rather than failing**, so an integration reading beyond its scope gets less data, not an error it will retry forever.

数据访问服务器

数据集定义和服务账户保存在服务器上，而不是本机。请连接到保存它们的服务器。

服务器地址

`https://historian-host:7443`

API 令牌

令牌仅用于本次会话，不会保存在任何地方。请存放在密码管理器中。

连接 取消

Connecting to data access

All integration is a **read** path. There is no write back to a controller through the API, the SDKs, MCP, Grafana or Sparkplug. See the *Software Architecture Document*, §8.

16. Modules and licensing

License

Trial — 30 of 30 days remaining.

This machine's ID (send this to get a license):

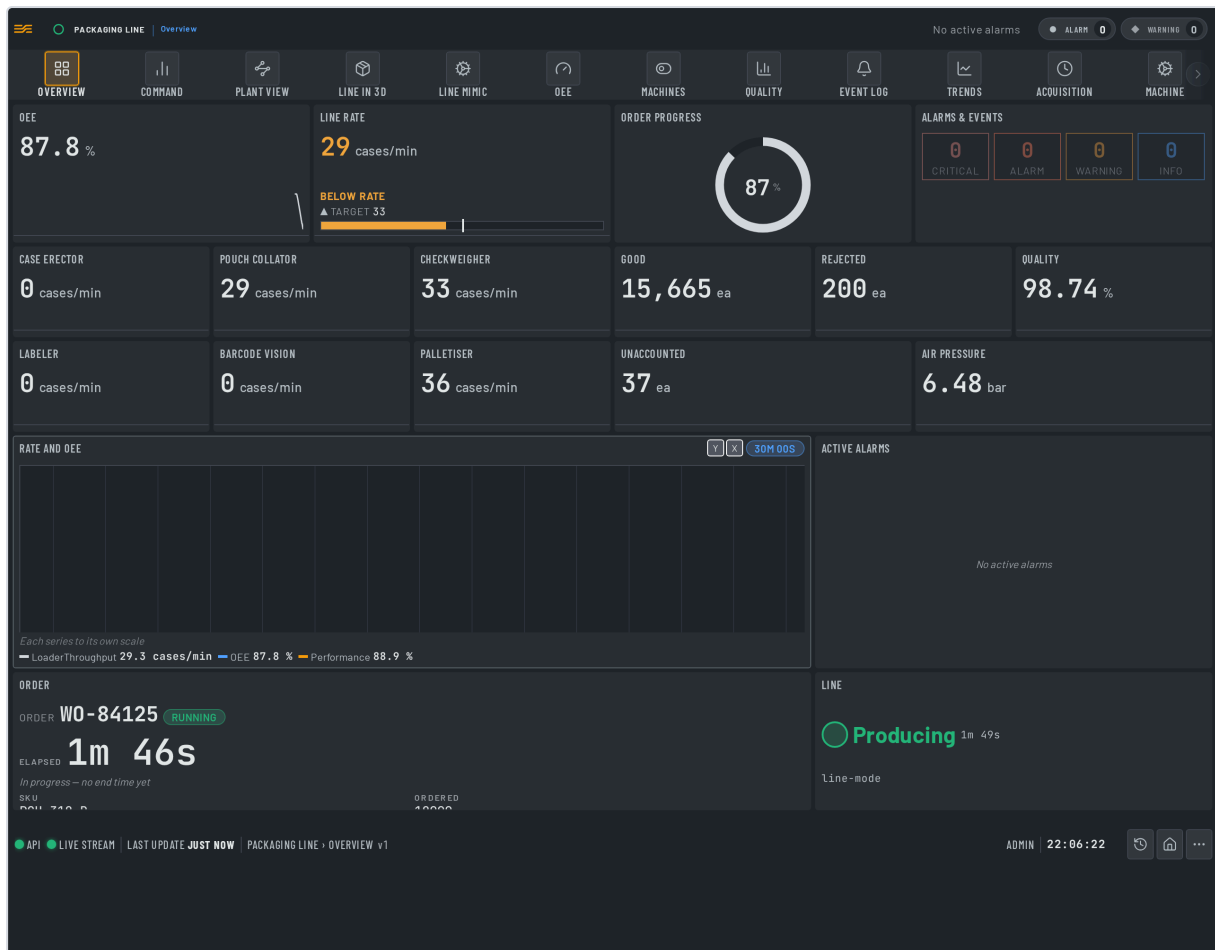
`83BF-1F35-E742` Copy

Install License... Close

The licence screen

Help ► Licence says what this installation is permitted, and is the first screen to open when something is unexpectedly unavailable. **Administration ► Modules** enables the six modules; each is gated on six axes and **the screen names the first one that failed**, so "not licensed" and "not enabled" are never confused with each other.

Part II — The browser operator HMI



The operator HMI

The exclusive operator surface. It installs as a PWA, and it attaches to a node.

1. Signing in and the site switcher

Sign in over **https**. The session is a cookie — HttpOnly, Secure, SameSite=Strict — and idle sessions are signed out.

The first time a browser opens the HMI, it has to trust the site's certificate. A plant server makes its own certificate authority (Installation Guide §7), and a browser that has never met it says *Not secure*. Clicking through the warning is not enough: the page loads, but its live connection is refused, so every value stays stale behind **Live stream down**.

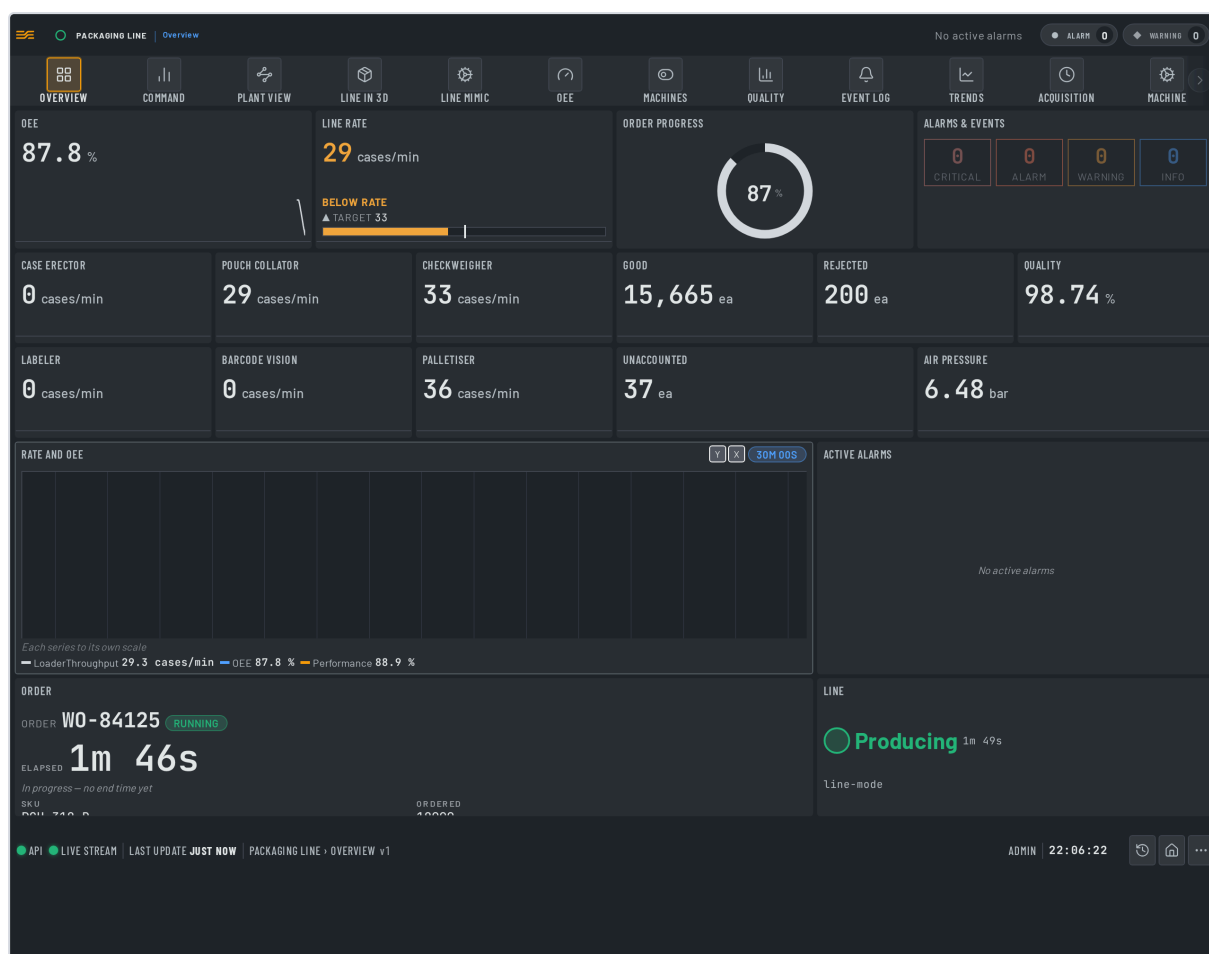
- **On a machine Meelmo Setup installed** (Windows), there is nothing to do: setup adds the site's root certificate to the machine's trusted roots.
- **On any other machine** — a tablet, a phone, a PC with only a browser — follow **Browser warning about this server? Trust its certificate** on the sign-in page, or open `https://<server>:8443/trust`. That page shows the certificate's fingerprints, to compare with the ones on the plant server before you trust anything, offers the certificate to

download, and gives the steps for Windows, macOS, iOS, Android and Firefox. On Windows, **trust-meelmo-site.cmd** does it in one click: it asks for administrator rights once and adds exactly that certificate. Close and reopen the browser afterwards.

The runtime shell resolves **one project per viewer** from the site's assignments, routed by everyone / role / station. A viewer entitled to more than one sees a **site switcher**; an administrator sees them all.

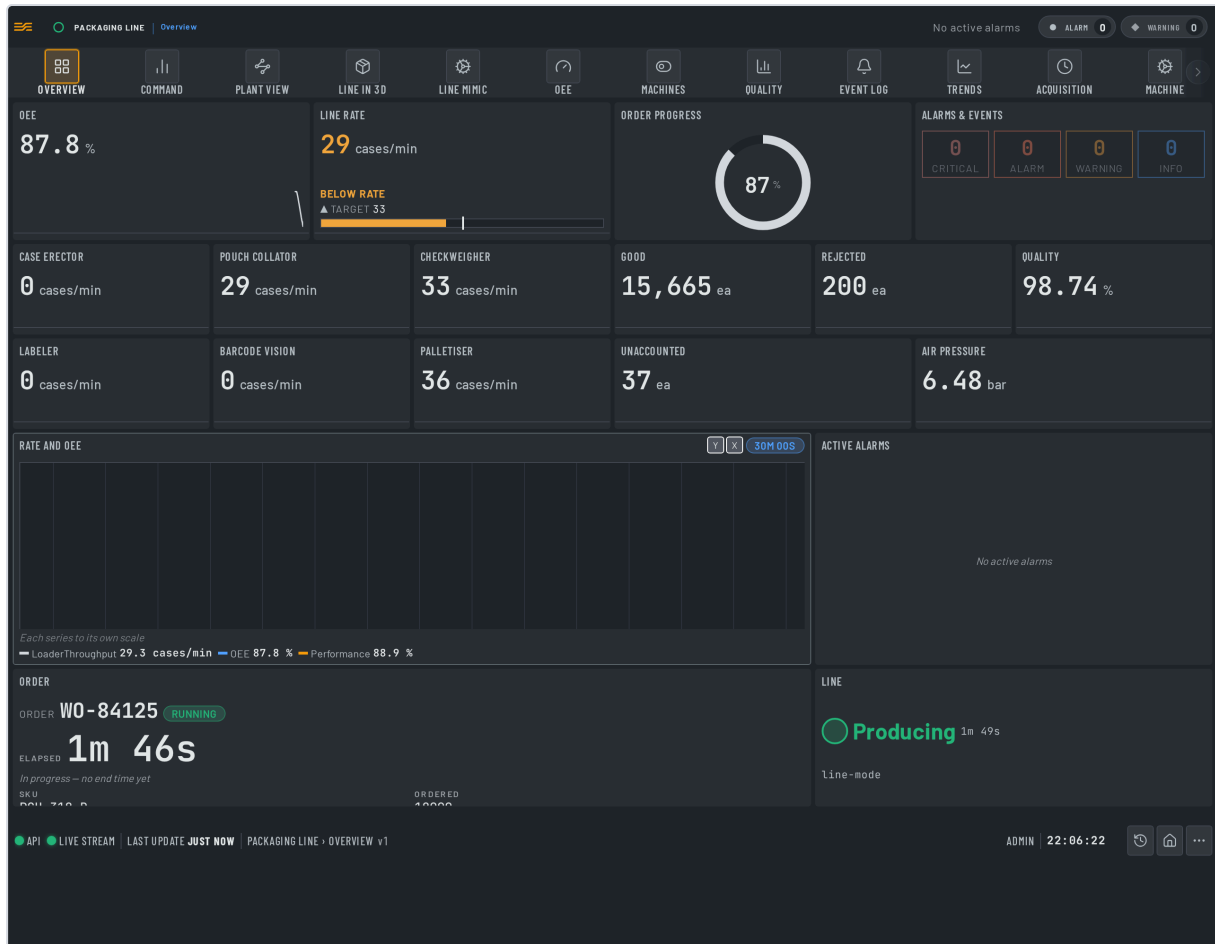
2. Live values, gauges, states and mimics

Pages are built from the component types of one catalog, shared by the server that validates and the browser that renders; the HMI Studio manual describes every one. Freeform mimics draw from a library of equipment symbols. Any shape can be animated from a signal.



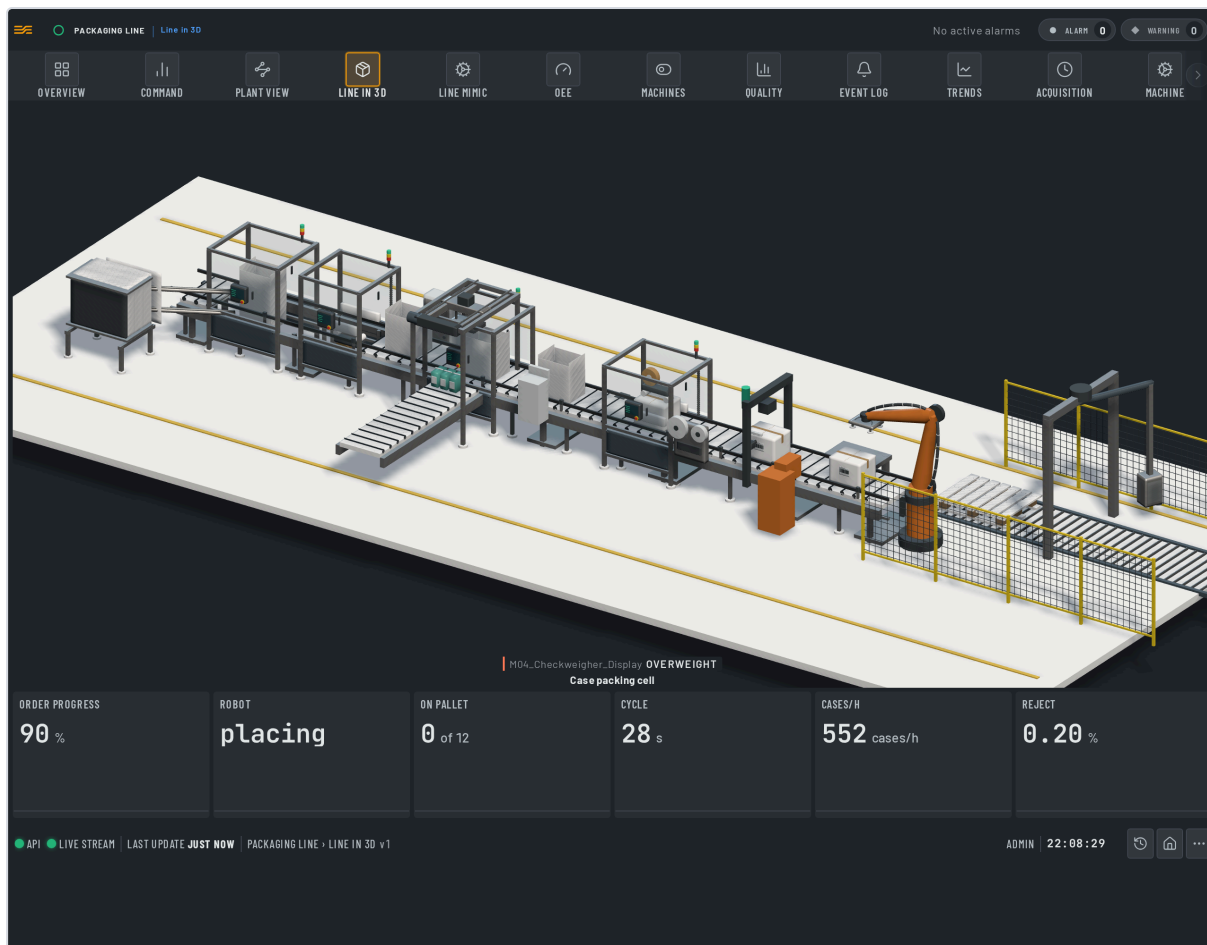
An operator overview page

The shell is the same on every page: the site and page name at top left, the alarm and warning counts at top right, the page rail underneath, and a status bar along the bottom carrying **API**, **Live stream**, the age of the last update and who is signed in. When a page stops updating, that bar is where it says so — a value that has gone stale never pretends to be current.



A machine chain with OEE, rejects by cause and the shift's counts

A command screen like this one is a mimic, a set of state and rate readings, three OEE gauges, a reject breakdown and the shift counters — all bound to live signals and all **observing**. Everything on it updates from the node; nothing on it writes to the plant.



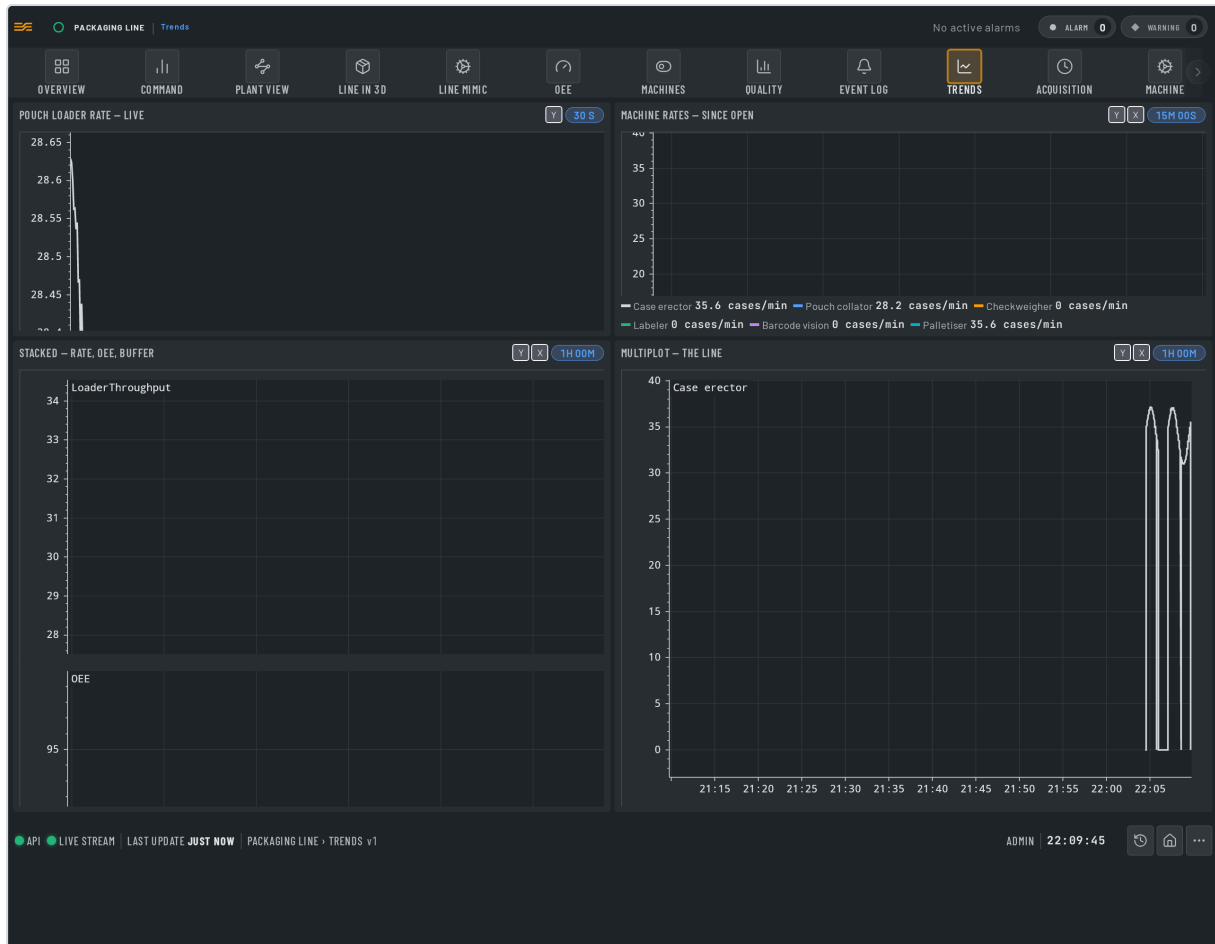
A 3D line view — press a machine to open what it carries

Where a site has a model of its line, a 3D view is an ordinary page component: pressing a machine opens its detail, headed by the machine's name rather than a signal id, listing exactly the signals that part carries.

3. Live trends: scope, logger, multiplot and stacked

Four kinds, and the difference matters:

Widget	What it is for
Trend	The plant's recent history, updating
Scope	A live waveform — every acquired sample, not a decimated snapshot
Logger	An accumulating record that grows across the shift
Multiplot	Several signals stacked on one time axis

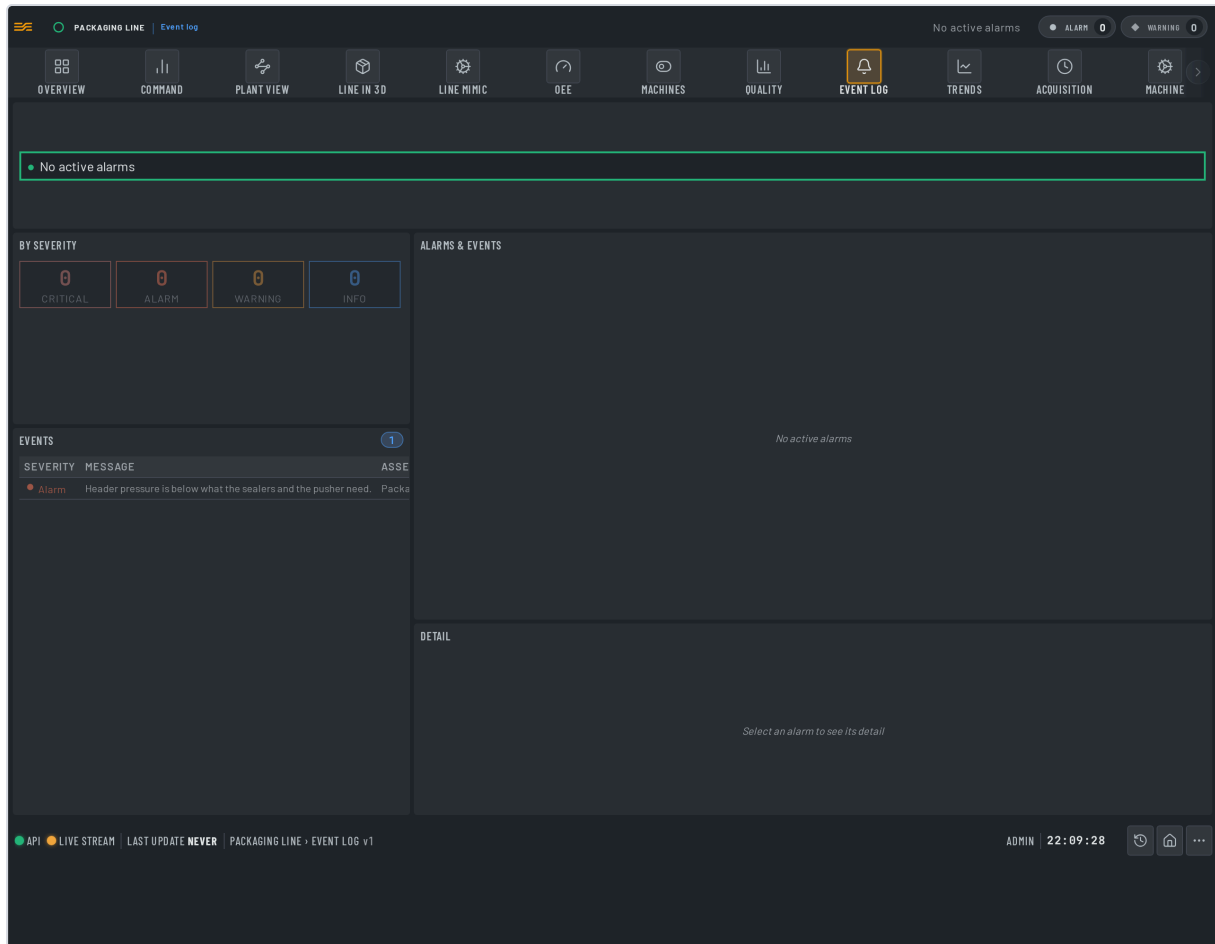


Live trends on an operator page

The transport underneath the scope and the logger is **lossless**: every acquired sample, flushed on a fixed cadence rather than one frame per group read. That is what lets them draw the real waveform. When two components ask for the same signal at different rates, **the fastest wins**.

That distinction is worth seeing rather than reading. A trend samples what the server sends at an interval; a **scope** receives every acquired sample. On a signal that moves faster than the interval, the trend is a correct picture of a decimated series and the scope is a correct picture of the waveform — and only one of those answers "what did the process actually do".

4. Alarms



The alarm page

Raise, view, **acknowledge** and **shelve**. Acknowledgement stamps the occurrence durably, is server-authoritative, is audited, and requires the `AcknowledgeAlarms` permission. It writes nothing to the process.

Both happen at the node, as you, whichever host drew the button. The browser host sends the request to the node that raised the alarm with your own session: the node decides whether you may, answers the alarm in its engine — so every workstation's alarm panel shows it answered — and in its store, and records your name in its own trail. That includes a **plant server**, where the recorder service holds the node and the browser host is its client: an operator there acknowledges and shelves exactly as on a host that holds the node, instead of being told to go to a node with no screen.

Shelving sets an alarm's rule aside for a while, with a reason the next shift will read. The node caps how long (eight hours unless a site says otherwise), and a shelved alarm is still recorded — it simply tells nobody until its time runs out or somebody brings it back. Below the active list, the alarm drawer shows the **shelving review**: what is set aside, by whom, why, and how long is left, with **Bring back** beside each. Shelving needs the same permission as acknowledging.

An alarm stays on the banner until somebody acknowledges it, even after its condition has cleared. So an alarm that raised and cleared in a few seconds, while nobody was looking, is still there to be seen and answered; acknowledging it is what takes it off the banner.

Where a site has turned on `signatures.requiredForAlarms`, an acknowledgement, a shelf and bringing one back each ask for an electronic signature first — your password again and what signing means — and the signature is bound into the host's hash-chained trail before the node is asked.

The counts in the shell header are live on every page, so an operator on the OEE screen sees an alarm raised without having to be on the alarm screen to notice it.

5. History replay

A time-slider on any published page: drag back, re-animate a mimic from history, jump to an alarm, play at up to 60x, and resume live. **Replay is read-only** — commands and acknowledgements are refused while replaying, because an acknowledgement of a two-hour-old alarm made from a replayed screen is an acknowledgement of the wrong thing.

6. What an operator may and may not do

Monitoring is the default. Where command-and-control is enabled, a tag write passes four independent layers: the host must be started with writes allowed and must be acquiring, or the route does not exist; the operator must hold `CommandProcess`; the configuration must allow the write and enforce its range; and every attempt passes the audited write authorizer.

This command path is shipped and tested, and is pending a formal adversarial security review before it is recommended for production command use. Until that review clears, treat the operator HMI as a monitoring surface.

7. Models and analysis, without the Desktop

A site whose only client is a browser has the Desktop's **Analysis Studio** and **Models Studio** as pages of the web client, under the same person's session and through the same routes. Before they existed, the only interactive way into either module from a browser was an AI client over MCP.

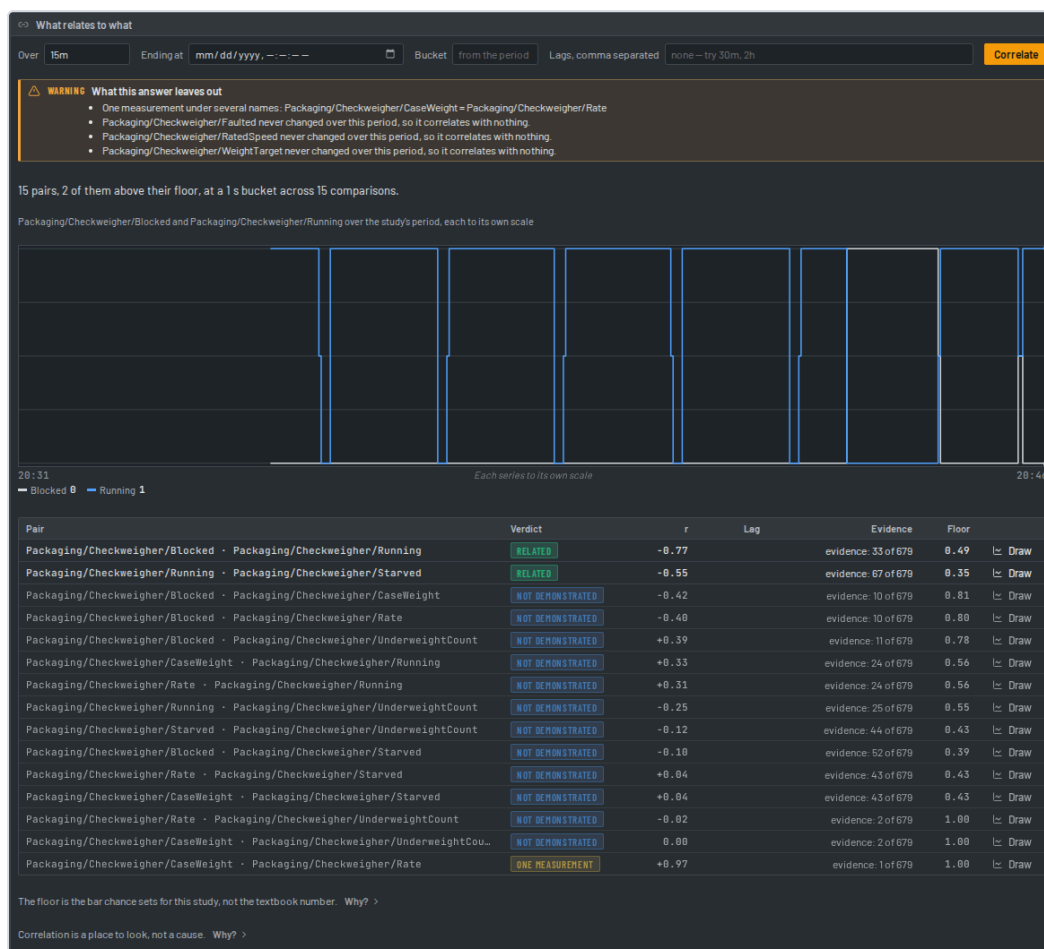
Both appear in the top navigation for anybody whose account may **read history** — the permission every route behind them checks first. The licence is the second thing those routes check, and the pages ask it each time they open: where the licence does not include what a page is made of, the page says which module and who can see the licence, instead of offering a form that could only be refused.

Analysis (`/investigate`) is two pages under one tab:

- **What changed.** Choose the signals to look at — a search and one button take every match, so "everything under Line 2" is two keystrokes. Where the node's own rules detected something in the last week, **Something that happened** lists it — when, what the site calls it, and where — and picking one sets the instant; that is the half somebody anticipated. If nobody knows when it started, **Find moments** sweeps a period for instants where several signals stepped together; pick one and **Investigate** ranks every chosen

signal against its own recent behaviour. Each row says what the signal did, how far out it went, when it moved — *before* is a candidate, *after* is a consequence — and what it was judged from: a signal whose record was too long to read raw is judged from buckets, its departure is rounded to one and says so, and one whose buckets did not cover the whole period is marked *part of the period*. The signals that moved are drawn, with the investigated window shaded. **Find windows like this** searches the sweep's period for other stretches shaped like this one, made of the signals that moved — closest first, each a moment to investigate in turn, with the distance shown as a ranking rather than a measure. **Make a report** writes the investigation out through the same builder the Desktop's Analysis Studio uses and saves it among the node's reports (offered to whoever may design reports), with a link to open it. Name the fault and **Describe it** to turn the investigation into a rule; **Has this happened before?** replays that rule over history. The rule can then be kept in the project — see *Keeping a model or a rule* below — or copied and downloaded.

- **What relates.** The correlation study: choose the signals and a period, and every pair comes back in the study's own order, led by its verdict — *related*, *not demonstrated*, or *one measurement* under two names — then the coefficient, the lag and the evidence the pair was really worth. **Draw** puts a pair on a chart.



The correlation study in the browser: every pair led by its verdict, one pair drawn over the study's period

Models (/models) is the Models Studio's four steps in its order: what moves with the target, choosing the inputs, fitting, and only then estimating. The signals offered are the project's own — what the node's configuration acquires and computes, as the Desktop offers the open project's — and where the host cannot say, the whole catalogue, with a note saying that is what

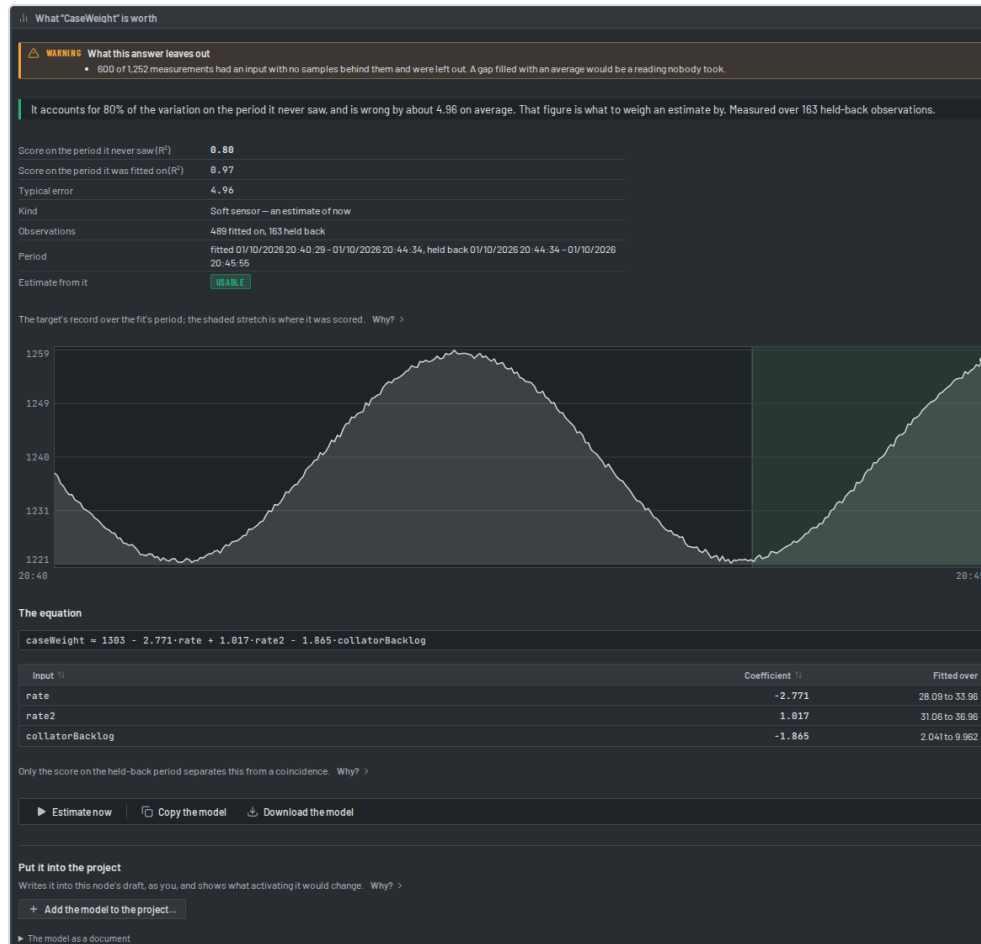
it is. The shortlist is always the arithmetic's. Where the site has configured an assistant, a button offers to show it the shortlist — the sentence beside it says what leaves: the signal paths, the name of the target, each candidate's correlation and delay, and no readings — and its notes arrive in a column of their own, in italics, under a line naming who wrote them. They never change the order.

A fit is read from the period it **never saw**: that score, the score on the period it was fitted on beside it, the typical error, and — for a forecast — what doing nothing would have got wrong, with a verdict sentence that says first when a forecast is no better than carrying the present forward. The server's warnings sit above the numbers. A usable fit can be estimated from, copied, downloaded, and given a **watch**: every threshold is priced against the record at the rate of interruption the site will accept, each row named for what it would have done — *within budget, too often, standing alarm, never fires* — and any row can be replayed to see its findings drawn on the record.

Keeping a model or a rule. Putting a fitted model, or a rule described from an investigation, into the project is a configuration change, and it is made the way every other one on a node is — in two presses, as you. **Add the model to the project...** (or **Add the rule to the project...**) writes it into the node's draft and shows what activating that draft would change, and activates nothing:

- first, whether it **adds** something new or **replaces** a model or rule the plant is already running — a refit moves an estimate with nothing else changing, so it is said before anything else;
- every setting that changes, before and after;
- anything **else** the draft already holds. The node keeps one draft for everybody, and activating it activates somebody else's edits with yours;
- what the node's validation refuses, in its own words. Anything refused and the draft cannot be activated.

Activate then makes it a new revision, with the sentence you give for the revision history. **Take it back out** puts back only this entry — as the running revision has it — and throws the draft away only when nothing else is left in it. Whether you may is the node's decision at every step: somebody it does not let change configuration is told *You are not permitted to change this node's configuration* and keeps the copy and download buttons, to hand the model to somebody who may.



A fit in the browser: the verdict first, the score from the period it never saw, and that period shaded on the record

Every word on these pages is in the reader's language, including the caveats the server sends with each answer; where a newer server has reworded one, the page shows the server's own words rather than a translation of what it used to say. What each analysis refuses to tell you, and why, is in the *Analysis walkthrough* that sits beside the Product Guide.

8. Administration in the browser

The **Administration** menu is offered only to an administrator, and every screen under it is refused to anybody else by the host. These are the screens for settings that were once reached only by editing a file.

Publication — Administration ▶ Historian in the browser

Where the node's samples go on their way past is set on the same screen as where they are stored, because both are one block of the node's activated configuration: **Save** makes a new revision that the node activates, and the revision history shows and rolls back the change like any other.

- **Publishing samples onward** — the MQTT or Sparkplug B broker. Two ticks at the top, *Publish alarms* and *Publish detected events*, apply to the broker and to Kafka alike. Under TLS the broker panel takes the **broker's certificate authority** (for a broker certified by

the site's own authority) and a **client certificate file** (PFX, or PEM with an optional key file) and its password; under Sparkplug B, a **primary host id**, so the node is born again whenever that SCADA host comes back.

- **OPC UA server on this node** — the port, the host name written into the endpoint URLs, the listen address, the root path, the folder of trusted client certificates, and the switches that open it further: an endpoint with no message security, a cleartext password on that endpoint, anonymous sessions and *accept any client certificate*. Each one that is on is said under the panel; the last is said in red, because a server that trusts every certificate authenticates no application at all.
- **Kafka producer** — bootstrap servers, security protocol, SASL mechanism, user and password, certificates, the topic mode and the three topics, compression and acknowledgement. The pickers offer only the values the node accepts.
- **MQTT broker on this node** — a broker on the node holding a governed unified namespace. The switch, the listen address (empty is this machine only; anything else is served over TLS with the site PKI's certificate, or not at all), the plain and TLS ports, anonymous clients, whether the node publishes its own signals (retained, at their place in the asset model), the enterprise level, the level names (ISA-95's by default) and the root path. **Access rules** and **Payload contracts** are text, one per line; a line that does not read refuses the save and is named by its number. **Show what the broker is doing** lists where it listens, its counters, the connected clients with their accounts and subscriptions, the topics, and the most recent refusals.

Every password on the screen — the broker's, the client certificate's, Kafka's SASL password — is sealed when it reaches the host and never shown again. An empty box reading *unchanged* — *type to replace* keeps the one held; forgetting one is its own tick.

Apart from the node's own MQTT broker, this screen does not yet show each publisher's live state or counters: no route the host serves reads them out. The node's log says whether a broker, a cluster or a client connected.

Signing in through the site's provider — Administration ► Single sign-on in the browser

Sets the OpenID Connect provider people may sign in through — Entra ID, Keycloak, Okta — beside this installation's own accounts, which keep working whatever the provider does.

1. Enter the **authority** — the issuer, exactly as the provider names it — and press **Test discovery**. The page reads the provider's metadata and says whether sign-in can use it, or why not, before anything is saved.
2. Register the **redirect URI** the page shows at the provider, and enter the **client id** and **client secret** it gives you. The secret is sealed by the host and never shown again; leave the box empty to keep the one held.
3. Under **Groups and what they grant**, add a row for each provider group: the service account its members read data as, their roles, screen permissions and area. Somebody in no listed group is refused.
4. Tick **Offer sign-in through the provider** and **Save**, then restart the browser host: the host reads these settings when it starts, and the page says whether the running host is offering the provider.

Dataset drops — Reports ► Datasets in the browser

Choose a saved dataset under **Datasets** on the Reports workbench. The *Drops to a lakehouse path* panel lists where the dataset is written on a schedule and how each drop is getting on: last run, next run, files written and the last error. If your account may manage datasets, **Add drop** opens the editor — destination, file format, schedule — and each row can be changed or removed. If the panel warns that *no drop will write on this node*, the machine has no `dataset-drop-roots.json`: an administrator of the server must list the permitted directories in that file first.

A report's webhook — Report Studio ► Settings in the browser

With the report's schedule turned on, the settings panel takes a **webhook** URL. Give it a **body template** to post your ticketing or maintenance system's own JSON instead of the file (`{{title}}`, `{{runs}}`, `{{fileBase64}}` and the other fields listed in the Product Guide), its **content type**, any **headers** (one `Name: Value` per line; they are not secret) and the **Authorization** header in its own box, which the node seals when you save the report. Afterwards the box only says a value is kept; leave it empty to keep it, or tick *Forget the stored Authorization*.

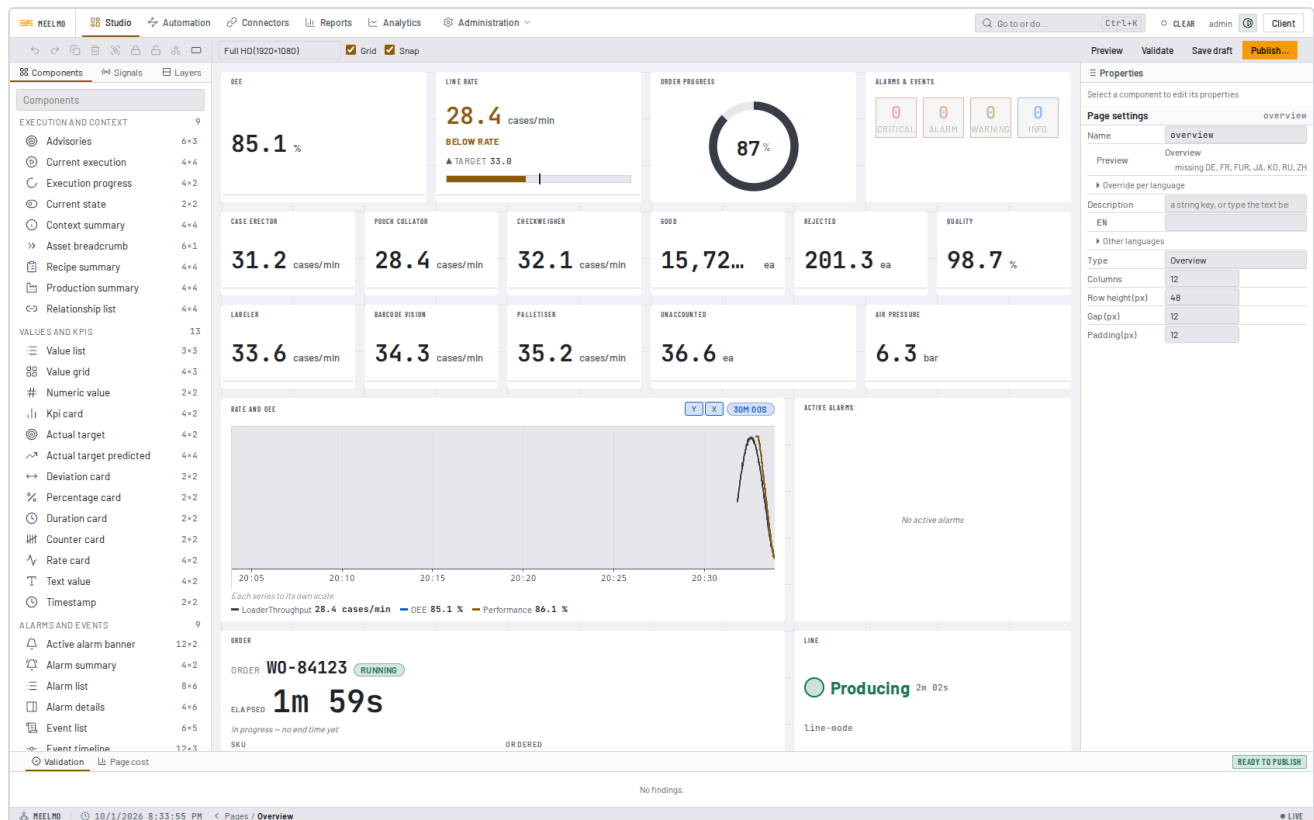
Azure OpenAI — Administration ► Studio Assistant in the browser

Choose **Azure OpenAI** as the provider, then enter the resource's address, the **deployment** name you created in Azure (not the model's own name), the resource's key and, if your Azure resource needs a particular one, the **API version**. Leave the version empty to use the build's default. **Check** asks the deployment one question before you rely on it.

Sending the log to a SIEM — Administration ► Audit in the browser

On **This host's log**, the folded *Forwarded to a SIEM* panel says in one line where the trail is going and how many entries were forwarded and dropped since the host started. Open it to set the collector's host, port (0 for the standard one), transport (TLS, plain TCP or UDP), format (RFC 5424 or CEF) and application name, and **Save forwarding**. The settings go into `forwarding.json` beside the log and are used from the next start of each program that writes the log. **Send a test entry** writes a real, labelled line on the log — look for `auth.audit.forwarding.test` at the collector — and the counts say whether it went.

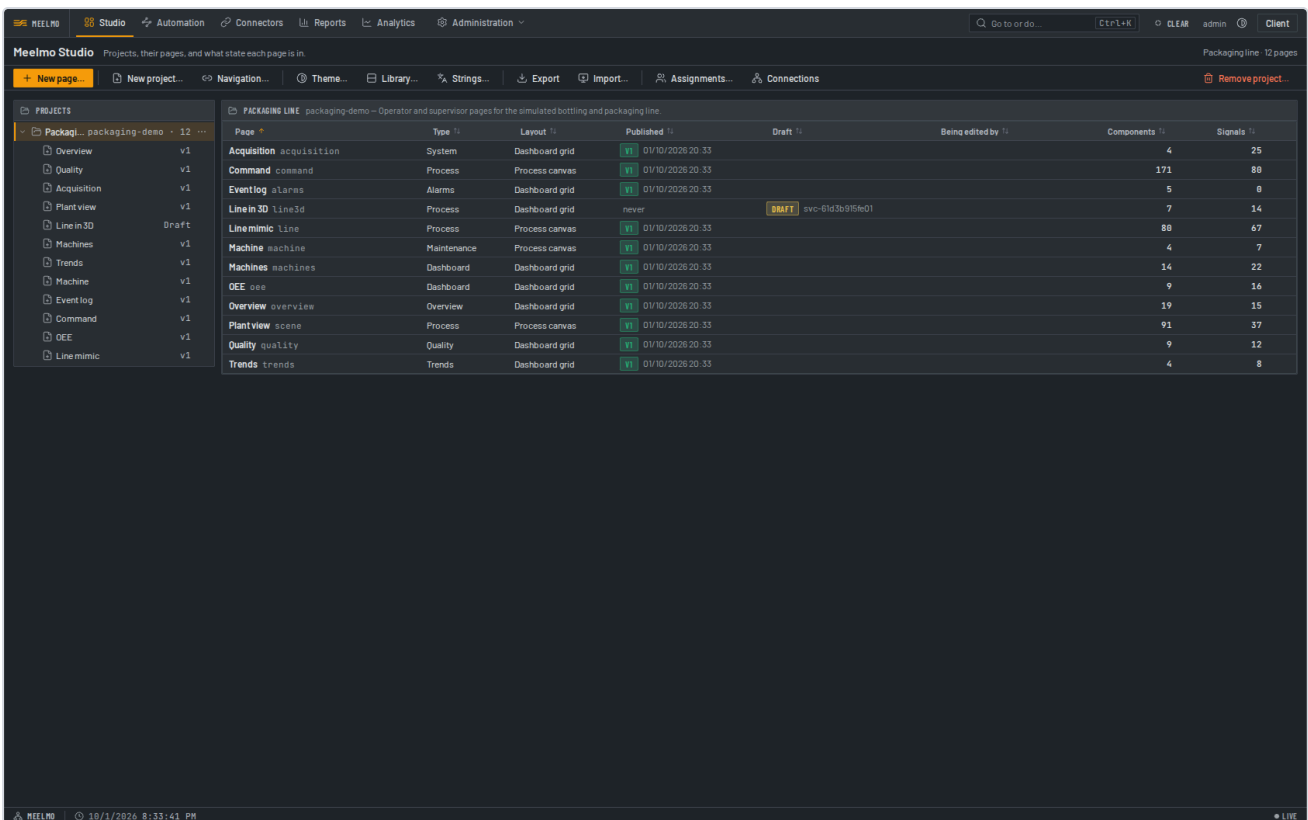
Part III — Authoring screens in the HMI Studio



The Studio workspace

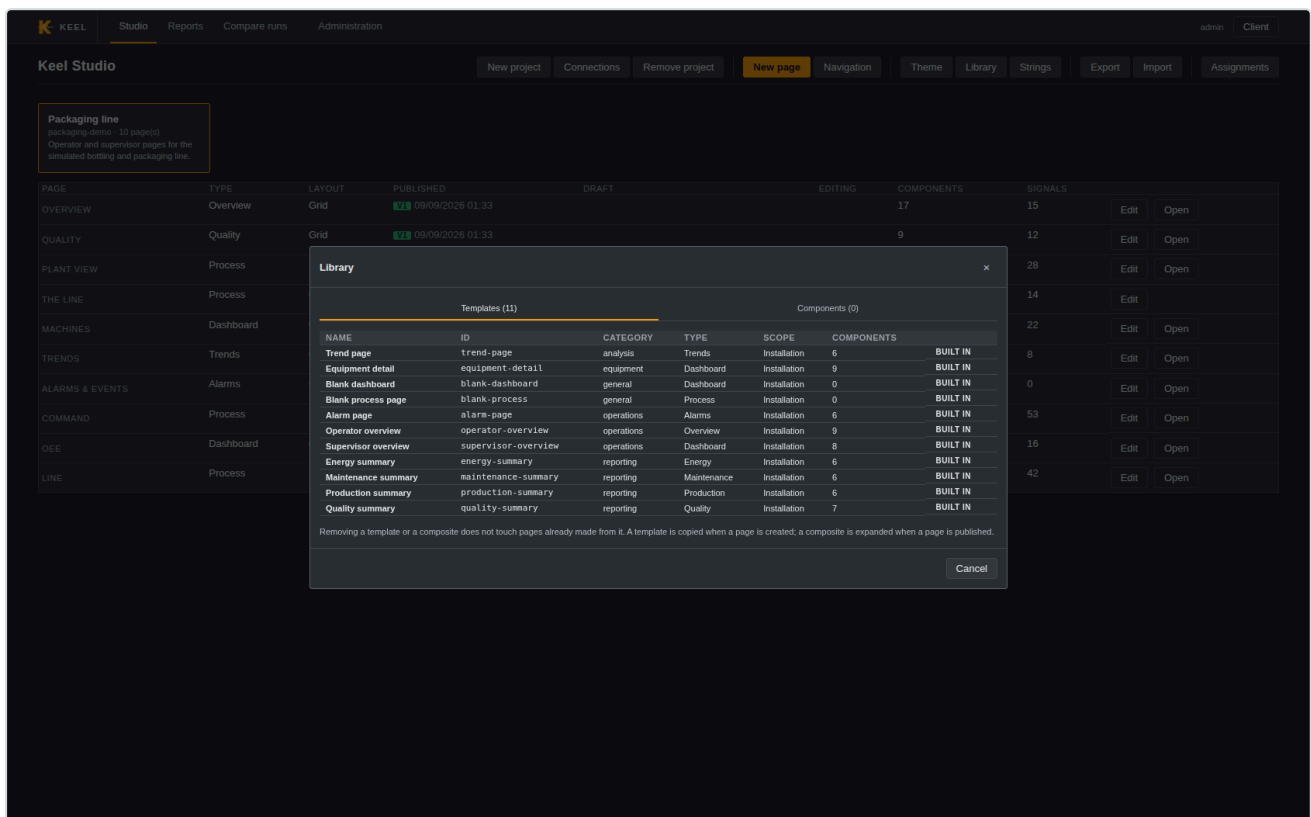
The Studio is where the screens Part II shows are drawn. It is an engineering activity, and it needs the `web-hmi` module.

1. Projects, pages and the canvas



Projects

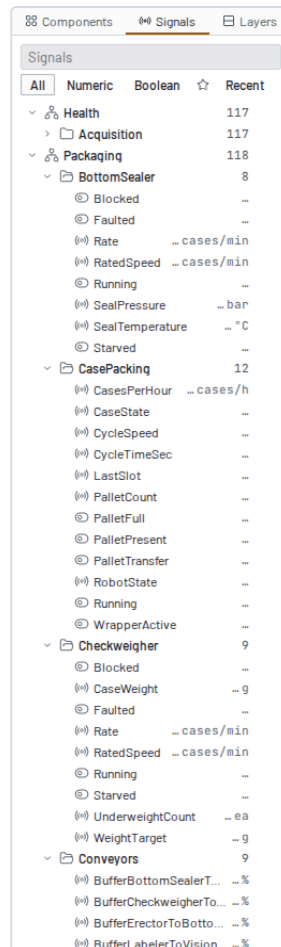
An HMI project is a `.meelmohmi` bundle of pages, components, bindings, animations and composites. A page is either a **12-column grid** or a **freeform canvas** — grid for dashboards, freeform for a mimic that has to look like the plant.



The component library

Drag components from the library; arrange them with the [layers](#) panel; adjust them in the [inspector](#).

2. Binding signals and choosing an update rate



Signals

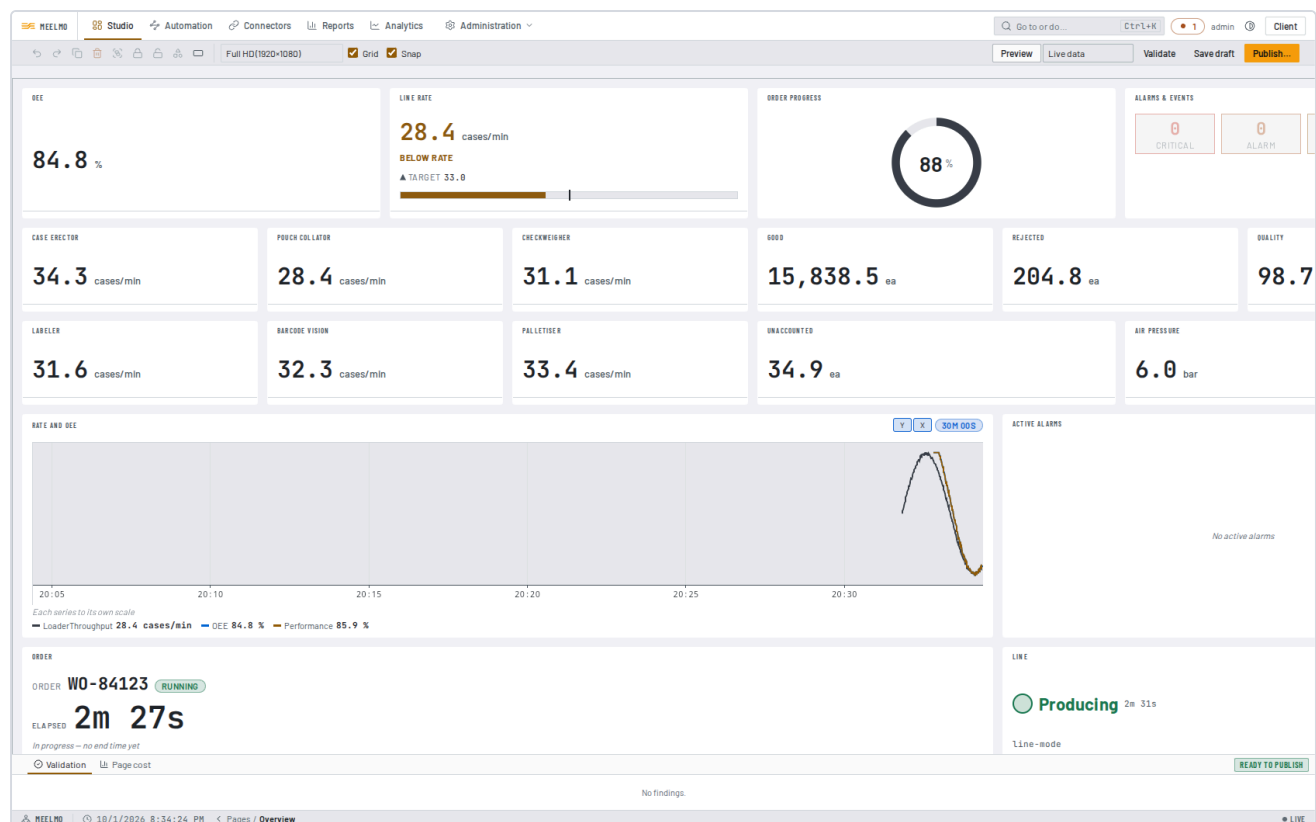
Bind a component to a signal, then choose its **update class**. This is the single decision that most affects how a page performs:

Class	Interval	Use for
OnChange	—	Discrete states, alarms, anything that sits still
Fast	≈250 ms	Values an operator reads continuously
Moderate	≈500 ms	Context that need not be instant
Detailed	≈100 ms, every sample	Scopes and loggers — the lossless path
Static	—	Labels and decoration

Do not put a whole page on Detailed. It is for the widgets whose value *is* the waveform.

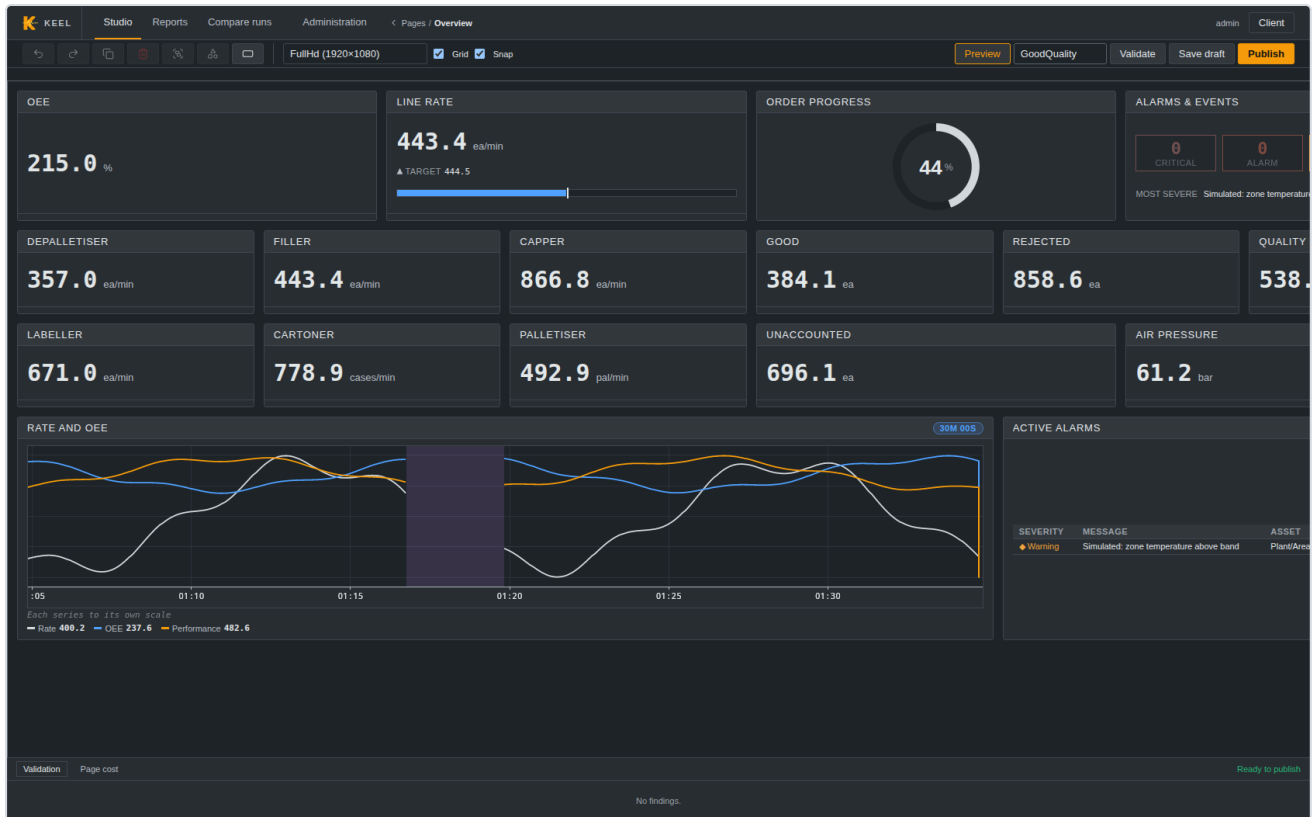
All fourteen binding source types **observe**. None writes.

3. Composites, templates and preview



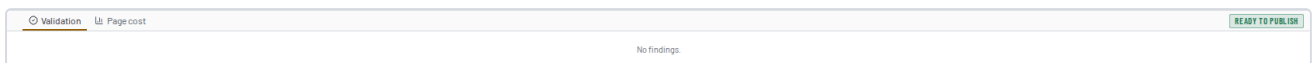
Preview

Turn a group into a **composite** and reuse it; a template with a parameter becomes one screen per machine. **Preview** ([tablet](#)) runs the page against a simulator through the same interface the runtime uses — so what you see in preview is what publishes.



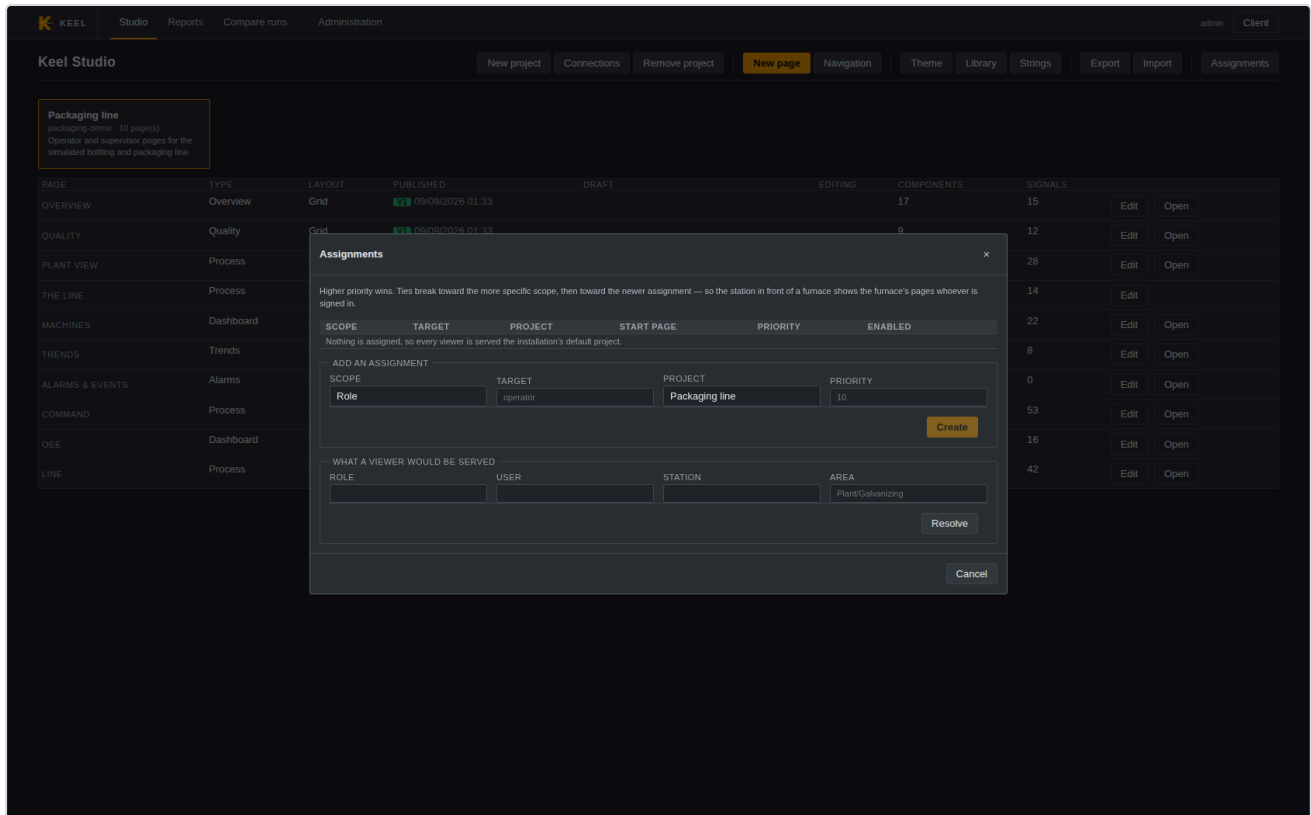
Simulate

4. Validation, publishing and assignments



Validation

Pages move **Draft** → **Published** → **Archived**. Publishing validates **server-side**, not just in the browser, and a rollback is itself a publish — so the history shows what was live at any moment.



Assignments

Assignments route a project to viewers: everyone, a role, or a station.

5. Commands and acknowledged

The only commanding constructs in a page are the `WriteTag` and `AcknowledgeAlarm` interaction kinds. Adding one is a deliberate act by a person, and it still runs behind the four layers in Part II §6.

6. Asking the Assistant

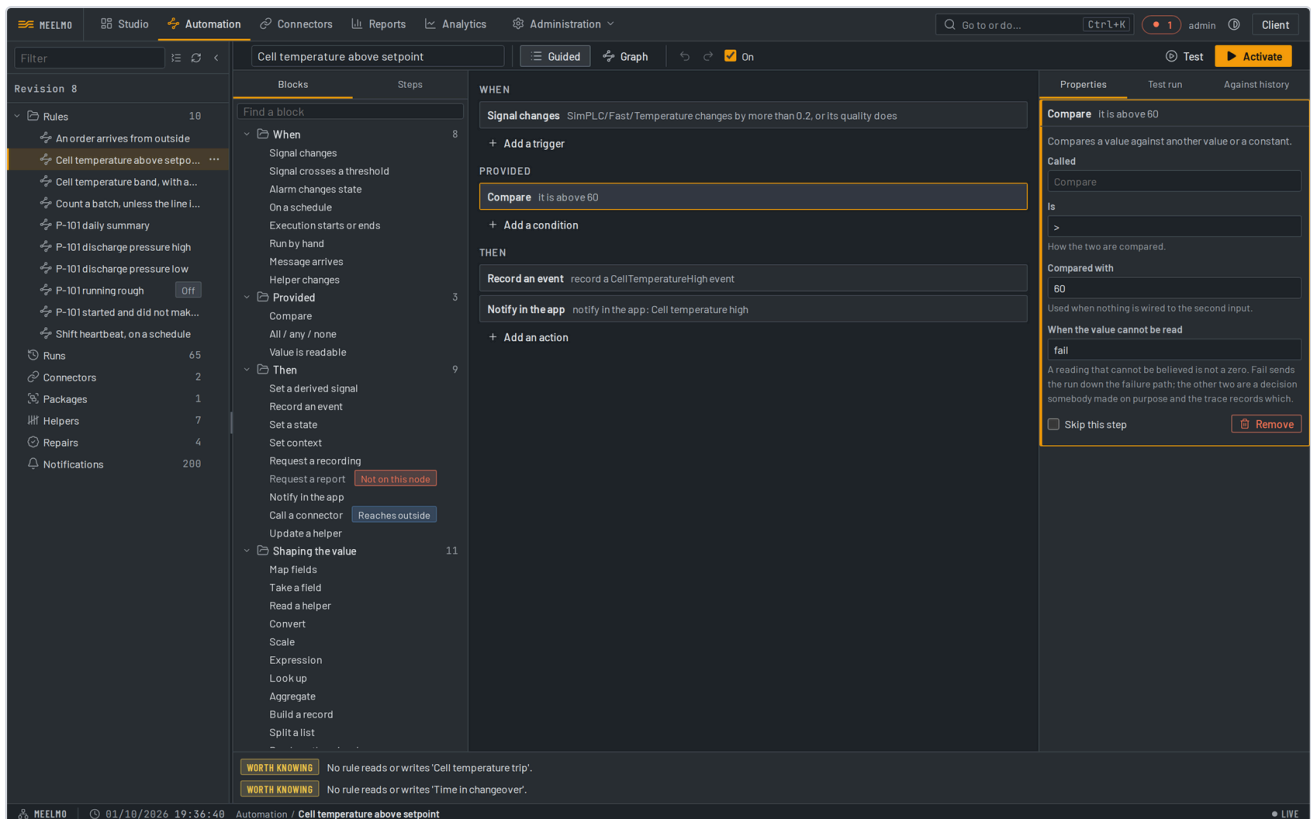
Where licensed and configured, the Studio Assistant takes a described change and returns a **proposal** — a list of typed operations against the open document, reviewable before it is applied, applied through the editor's own commands as **one undo entry**, and refused if the document moved meanwhile.

What it may propose is a closed vocabulary of operations, every one listed in the Studio Assistant manual: no property-path write, no document replacement, no query it wrote, no save. It reads the plant only through the read-only tool surface, **under your identity and your permissions**. It has **no operation that can set the `WriteTag` or `AcknowledgeAlarm` interaction kinds** — authoring a control is a person's act.

It is **off by default** and needs two separate things to switch on: the `StudioAssistant` capability, and a model endpoint the site configures. Point that endpoint at a machine inside the plant network and nothing crosses your boundary.

The endpoint may also be an **Azure OpenAI** deployment (provider `azure-openai`): the resource's address, the deployment's name and, optionally, the REST API version. The Desktop offers the same provider for the Models Studio's shortlist advice, under **Data ► Model for shortlist advice...**, with an endpoint, a deployment name, an API version and a sealed key, and refuses to save an Azure provider with no key.

Part IV — Automation rules



A rule in the guided view

A rule is what the plant does on its own when something happens: *when* the oven temperature goes above 60 °C, *provided* the line is running, *then* record an event and notify the shift. Rules run on the node, whether or not anybody has a screen open. They are written in the browser under **Automation**, and they need the automation permission to change. The full reference — every block, test and replay, helpers, connectors and limits — is the *Automation Studio Manual*; this part is what an author needs day to day.

1. Starting a rule

Automation ► New rule opens an empty rule in the **Guided** view, which reads it as a sentence in three sections:

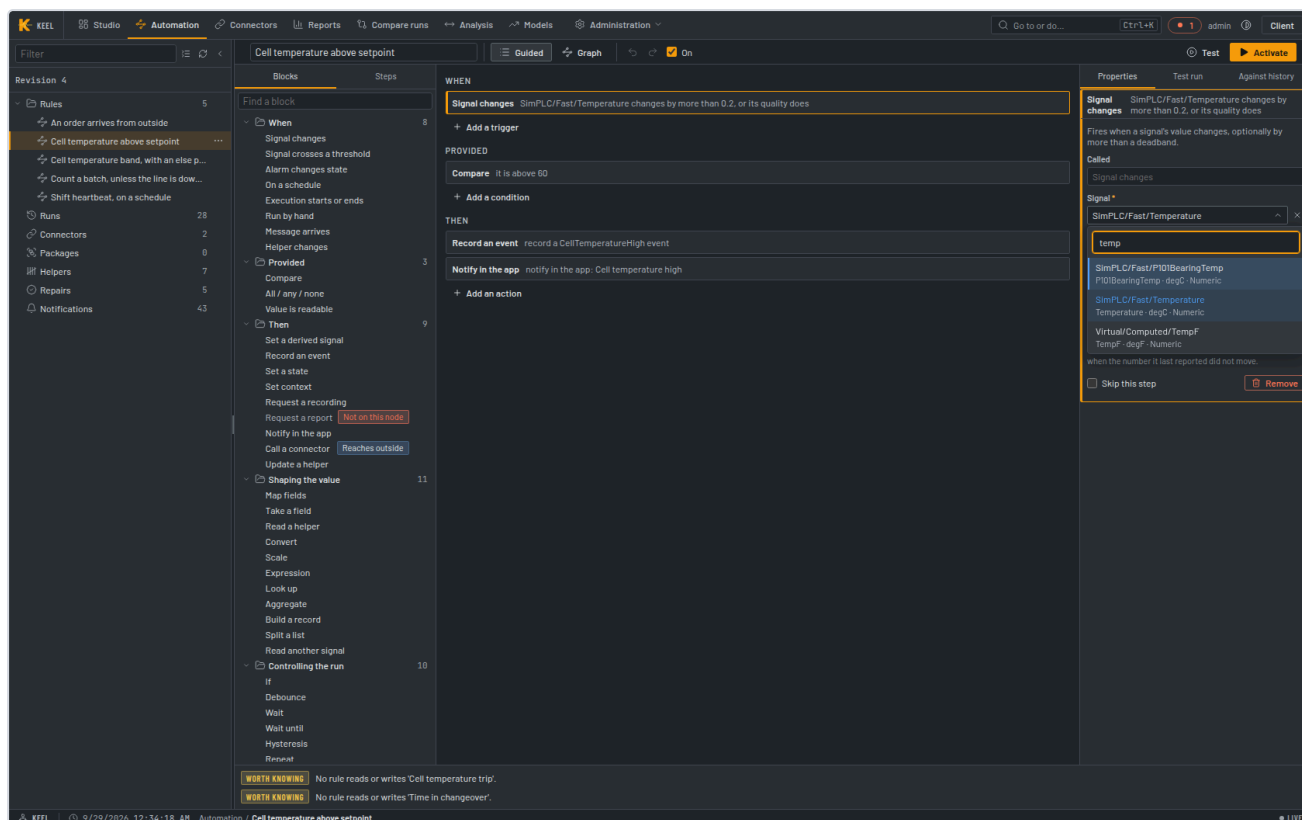
- **When** — what starts it: a signal changing, a threshold crossed, an alarm changing state, a schedule, a message from a connector, a helper changing.
- **Provided** — what has to hold for it to go on. None is fine: the rule then runs every time it is triggered.
- **Then** — what it does: record an event, set a state, notify, request a recording or a report, call a connector, update a helper.

A demonstration plant comes with example rules. Loading one (Help ► Demonstration, or Meelmo Setup's demonstration choice) puts three rules written for that plant into service beside it — two guided, one that needs the graph — so there is something to open, test and change.

Their ids begin `demo-`, and loading another plant replaces them without touching any rule you wrote.

Add a trigger, **Add a condition** and **Add an action** each offer the blocks for their section. A step added here is wired into the rule for you, in the right place: a condition goes before the actions even if the actions were added first. Click a step to open its settings on the right.

2. Settings are chosen, not typed



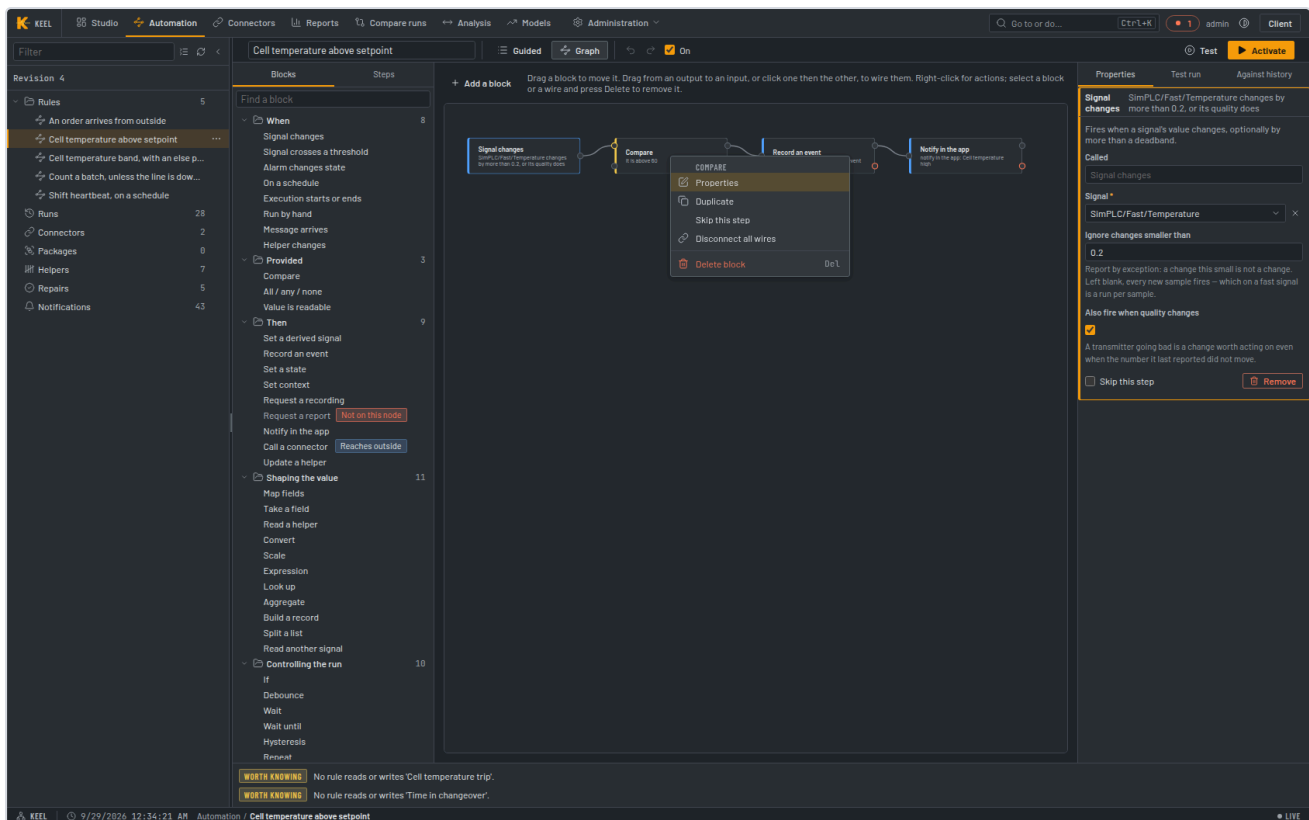
Choosing a signal

A setting that names something the node already has is picked from a list, so a misspelt name can never become a rule that silently does nothing:

- **Signals** open a searchable list. Type any part of the name — `temp` finds `SimPLC/Fast/Temperature` — and each entry shows its unit and type.
- **Alarm rules**, **connectors** and their operations, **helpers**, **reports**, **assets**, **event**, **state** and **execution types** and **time zones** are lists of what this node has. An alarm rule that is switched off says **Off** beside it.
- A **schedule's days** are seven tick boxes. None ticked means every day.
- A blank choice shows what the node does with it. An alarm trigger with **On** left empty fires when the alarm is **raised**, and says so.

A value the node does not have — a signal since renamed, a connector since deleted — is kept and marked rather than cleared, so opening a rule never changes it.

3. The graph view



The graph, with a block's menu

Graph draws the same rule as blocks and wires. It is one rule, not two: edit in either view and the other follows. A rule with branches, loops or blocks that shape a value in between (a *Map fields*, an *If*) can only be drawn as a graph; the **Guided** button is then greyed out and its tooltip names the block that needs the graph.

- **Add a block** from the palette on the left — click it, or drag it onto the canvas where you want it. Right-click empty canvas for **Add a block here**.
- **Move a block** by dragging it, or with the arrow keys (Shift for a longer step).
- **Wire two blocks** by dragging from an output (right-hand dot) to an input (left-hand dot), or by clicking one and then the other.
- **Select a wire** by clicking it; it is highlighted.
- **Remove** the selected block or wire with **Delete** or **Backspace**. **Escape** lets go of the selection.
- **Right-click** a block for **Properties**, **Duplicate**, **Skip this step**, **Disconnect all wires** and **Delete block**; a wire for selecting either end or **Delete wire**; empty canvas for adding a block, **Undo** and **Redo**. On a keyboard, **Shift+F10** or the Menu key opens the same menu.

Everything works without a mouse: Tab reaches every block, port and wire.

4. Testing and activating

- **Test** runs the rule as it is on screen against a value you type. Nothing reaches the plant: the result says what each action *would* have done, labelled *Simulated*.
- **Against history** replays the rule over a recorded period and lists when it would have fired.

- The bar along the bottom lists what is wrong. **Stops activation** must be fixed; **Worth knowing** is advice.
- **Activate** puts the rule into service. Undo and redo cover every edit until then.
- **Runs** lists every run afterwards — what started it, each step and the values it saw.
- **Delete a rule** from its row's menu (⋮, or right-click) ► **Delete rule....** It asks first, then saves a new revision without the rule: the node stops running it at once, its past runs are kept, and the change rolls back like any other. It needs the permission to activate rules. A rule a package made is removed with its package instance, under **Packages**.

Part V — When something is wrong

Organised by what you saw, not by which component said it. Most entries below are a **deliberate refusal with the remedy in it** rather than a fault.

It will not start

What you see	Why	Do this
A service starts, then stops, naming a bind address	A routable address with no TLS certificate. There is no option to serve without one	Add the certificate (Installation Guide §4), or bind loopback
"Unknown publisher" on the UAC prompt	The package was not signed	Return it to the vendor. Do not click through
The web host serves nothing on the address you expect	It started on loopback, which is where it starts	Add TLS and re-register the service with a routable bind

It will not connect

What you see	Why	Do this
The Desktop cannot attach to the plant server	<i>Administration ► Deployment</i> defaults its node control port to 8443; a recorder serves 8444	Change the port on that screen. This is the most common commissioning mistake
A driver connects but the password appears wrong	The configuration was copied from another machine. Secrets are sealed under a host-bound key and do not travel	Re-enter the password on this host
An OPC UA server refuses the credentials	The driver refuses to send cleartext credentials unless explicitly allowed	Use an encrypted endpoint, or set the override deliberately
A connection works from a laptop but not from the server	Segmentation, or a firewall on 9701 / 8443 / 8444	Check the port table in the Installation Guide §1
The browser says <i>Not secure</i> , and live values stay stale behind Live stream down	The browser does not trust the site's certificate. Clicking through the warning lets the page load but not its live connection	Trust the site's certificate: Browser warning about this server? Trust its certificate on the sign-in page (Part II §1)

The data is not what you expected

What you see	Why	Do this
The actual period does not match the requested one	Polling is best effort on a non-real-time OS	Open Administration ► Diagnostics . If the timing must be defensible, move to a source-synchronous mode

What you see	Why	Do this
Samples are being dropped	Best-effort capture under load, or a subscriber that stopped reading	Switch to lossless capture, or decimate the subscription. The counters say which
Two signals do not line up	They are in different acquisition groups, so they do not share a timestamp	Put them in one group
A trend has a hole in it	A gap is announced rather than hidden. <code>v: null</code> is missing, never zero	Check the diagnostics for sequence gaps and the spool for a historian outage
Values stop when the historian database reboots	They should not — samples spool and replay	Check <code>meelmo_spool_full</code> and <code>meelmo_spool_faulted</code> ; if either reached 1, the spool filled or its own disk refused a write, and that is where the loss starts
An automation rule has never run	Its trigger names a signal this node does not acquire, or a service this node does not have	Automation ► Repairs names which. The step also shows Not on this node in the editor

Something is unavailable

What you see	Why	Do this
A feature is missing entirely on this installation	Community does not <i>contain</i> the vendor drivers and four modules — they are absent, not disabled	Install a paid edition. A licence cannot conjure code that was never compiled
A module is licensed but unavailable	Six axes gate a module, and the screen names the first that failed	Usually the entitlement sidecar was not installed, or the module is installed but not enabled
HTTP 402 from the API	The licence lacks a capability. It is not a payment prompt and retrying will not help	A person has to change the licence
HTTP 429 from the API	This identity exceeded its per-minute budget	Back off and retry
HTTP 401 mid-session	The token was revoked; revocation takes effect on the next request	Issue a new token
The real-time endpoints answer <code>no_live_feed</code> (503)	Only a host that acquires has live data. A history-only server has none to serve	Point at the acquiring host
The Assistant panel says it is unavailable	One of licence, switch or provider withheld it — the panel says which	Fix the one it names

Recovering

- **A recording is more important than the application.** `.opslog` files are self-describing and readable while being written; a crashed session does not cost you the file.
- **Back up the state directory**, and `secrets/master.key` with it. A restore onto a different machine needs the secrets re-entered, because they are host-bound. Rehearse that restore before you depend on it.
- **A support bundle** (from the diagnostics screen) collects logs and configuration with credentials redacted, and deliberately excludes your project, the user store, the audit directory and the key store. Read it before sending it — you know your plant.